Algorithms

A tour d'horizon of the mathematics of
solving, optimizing, predicting and classifying.

Yves Jäckle
Version 2, date

Contents

1 Data

1.1 To do list

Things I still have to do for these notes.
First priority:

- Graph partitioning, clustering, and spectral graph methods
- Integer polyhedra, TDI systems and compuing integer hulls via Pfaffenhofer
- Disjunctive programming Balas
- MINLP (thesis)
- Tree decompositions, packing covering, chapters 19 to 20 of Graph Theory by Bondy Murty
- Group testing and compressed sensing (in math-bio folder)
- Matroid EA (EvolAlgo) methods from Skutella paper
- Enumeration algo papers
- Min cost flows in Hypergraphs and hypergraph-simplex algo, Hypergraph min cuts, Hypergraph coloring and connectivity, Hypergraphs in Schrijvers encyclopedia.
- Barahona cut polytope paper (plus chap on "Geometry of cuts and metrics")
- Cominatorics and machine learning and data science (TDA, "CombiDataScience", CDA)
- Discrete geometry in ML ("DG_ML", "VCdim_polytopes", "ml_polytopes" in AI folder)
- Frugal algorithms from thesis Sahil
- "Supply-Chain Optimization" by Veinott
- "Introduction to combinatorial designs" by Wallis
- GeometricProgramming
- OptiGroups
- QuadFlows
- CombiOptUncertainty
- AlgotithmicCombinatorics
- ConvexIntersection
- ConvexPacking
- p-adicNewton (and other computational algebra methods)
- EigenvalueAlgo
- Graph_Sparsification
- "Non-sep-non-disc-path" for non separating, non disconnecting paths

- Gradient descent material (relation to dynamics and geometry: "GradDesc..." in folder), including "ConditionalGradients" monograph
- Projected Newton and homotopy methods ("ProjNewton...")
- Computational geometry chapter (Convex hulls, triangulations (book chap 1 to 4 and 8), Square packing papers, "DiscreteCompGeo_skeletons", "Knots_NPcomplete")
- Algorithms in structural biology
- Distance geometry and molecular models, also "DistGeo_DataAna" in DG folder
- Monte-Carlo methods for optimization (CRC book for data science)
- Infinite dimensional linear programming (Barvinok chap 4.3, 4.4, 4.6, 4.7, 4.12)
- Perfect matching via polynomial identity ("PerfectMatch-SchwarzZippel")
- Held-Karp for TSP, including the paper with constraint programming improvement
- Numercal LA: Conjugate gradient descent (Shevchuck paper); Power method for largest eigenvalues, Gaussian reduction (quadratic, "ReductionGauss" in folder)
- Numerical methods for ODE and PDE: Runge-Kutta and FEM, as well as connection of gradient flows to optimization
- Optimal control (Evans notes and Numerical methods book)
- Calculus of variation (Kot)
- Numerical methods for optimal control: "OptControl_IntPtsMethod"
- "Algebraic Stats for Comp Bio", more on phylogeny and strings
- Machine learning chapter (MSPs via Reinforcment learning book by greek guy and japanese guy; Neural network books Springer (Calin O., Rojas ?), chapter in data science book by CRC; HMMs via script and chapters of "Algebraic Stats for Comp Bio", Hypergraph Neural Nets; Grosse lectures; Geometric Learning "GeometricLearnin-1" ; Machine Learning by Venkatesh ; "Complexity_NN"; GNN stuff; "NN_Topology"; "NN_Algebra"; "NN_NAS_MaxFlow")
- Interactions of statistical physics and combinatorics ("StatMecha..." in combinatorics folder, Perkins)
- Combinatorial statistics
- Interactions between combinatorics and numerics ("CombiNum...")
- MechaDesign and DistribComputing lectures, and Roughgardens AGT notes, and Votingalgo and MultiagentSystems notes, Ulam Game, parity game, reachability switching games, diffusion games, "VotingBribing", Computational social choice (survey), shelling games, financial network games, domination games, maker-breaker, pursuit-evasion, fair division, ...
- Differential game theory
- "FeatureRanking_wGameTheo" in AI folder
- Graph mining stuff in AI folder (including links to combinatorial optiization)
- Topological Data Analysis and HodgeRank (AI folder, "TDA...", "Ranking_CombiHodge" and "CombinatorialHodgeTheory")

- Convex relaxation for non-convex problems ("ConvRelax..." in DOpt folder)
- Non-convex optimizatiom (NonConvexOpt)
- Optimization on manifolds (OptiManifolds)
- Matroids: Hungarian Algo intersection (Papadimitriou), "FastMatroidInersection"
- Microeconomics of game theory chapter from Reny
- ParetoFront papers (maximal vectors), also "ParetoSets"
- Art gallery problems ("Art Gallery Full Book")
- Combinatorial optimization for geometry "CombiOpt-Geometry"
- Evolutionary algorithms (AI folder, files EA...)
- Particle swarm optimization ("PSO..." has a run-time analysis) and neural computing ("Neural-Comp..." ML for AI and more)
- Multi-criterion optimization (MOCO mst and slides)
- Inverse Problems
- Information theory (entropy chapter of extremal combinatorics book; Berkley lecture ("B_info...); books)
- Discrete differential geometry, in particular discrete minimal surfaces
- Symbolic integration
- Interpolation: Multivariate in Zippels book, "Approximation theory" books
- Reed Solomon codes and coding theory in general (Algebra)
- "BooleanAnalysis" paper with a proof of Arrow's theorem using Boolean Fourier, and all of O'Donells book "BooleanFunction..." in combinatorics folder
- Application of combi opt to bio (Gusfield book "OptLabBio...")
- AKS primality testing algorithm via the finite field AMS book
- Algorithms in Galois theory ("Algo_Galois")
- Fill out stuff left half done, fix errors

Second priority:

- Global minimum cut and Gomory Hu trees
- Simplex chapter (plus low dim LP (comp geo book), Klee-Minty, Sensitivity, active set algo, Randomized simplex algorithm from Kalai Paper)
- Quadratic and sequential quadratic programming (degeneracy, and paper on SQP and opt control, or other papers on the topic cause the ones I have are trash)
- Cycle Structure and feedback sets from the Digraph book
- Exact solution from interior point methods "Exact_IntPtLPsolver"

- Finish Rothvoss: integer conic programming and discrepancy theory
- CP linearisation, proximal algorithms
- Online algorithms

1.2 Books to read

Things I still have to read.
Sorted by priority:

- Wasserman's notes on statistical learning, in particular random forests, optimal transport, etc.
- Algorithmic toolbox by Roughgarden and Valiant
- the ML notes and graphical models notes
- the algorithmic game theory notes by roughgarden
- "MarkovSamplingHMM" for Monte Carlo and hidden Markov models, among other
- "NUMERICAL LINEAR ALGEBRA" by Rannacher
- "Numerical Optimization" by Wright and Nocedal
- "Numerical Optimization" by Bonnans, Gilbert, Lemaréchal and Sagastizábal
- "Linear and nonlinear inverse problems with practical application" by Müller and Siltanen
- "Inverse Problems and Data Assimilation" by Sanz-Alonso, Stuart and Taeb
- "Data Science and Machine Learning" by Kroese, Botev, Taimre and Vaisman
- "Geometry of cuts and metrics" by Deza and Laurent
- "Gems in combinatorial optimization"
- Williamsons lectures on mathematical programming
- "Digraphs Theory, Algorithms and Applications" by Bang-Jensen and Gutin
- "Algebraic Methods for Dynamical Systems and Optimisation" by Kaihnsa (stored as AlgOptBio)
- "Combinatorics on words" by Heubach and Mansour
- "Computational complexity of counting and sampling" by Miklos
- "Chemical graph theory" Wagner and Wang
- "Combinatorial scientific computing" by Naumann and Schenk
- "Exponential time algorithms for graph coloring" unknown, named "ColoringAlgo"
- "Local search in combinatorial optimization" Aarts and Lenstra
- "Randomized algorithms" by Motwani, Raghavan
- "Planning algorithms" LaValle
- "Triangulations..." book by Loerna and Rambau
- "Triangulations" notes on mesh generation by Shewchuk
- One of the discrete differential geometry notes and the discrete minimum spanning surfaces notes
- "Dynamic Programming" by Sniedovich
- "Combinatorial optimization" by Jungnickel

1.3 Sources and reading recommendations:

Things I read (partially) and worked with to make these notes, with some mini-reviews.

- *"Introduction to Algorithms" by Cormen, Leiserson, Rivest, Stein:* is mostly about combinatorial and graph algorithms and data structures, but also introduces other topics such as the FFT and computational geometry. It's motivated, rigorous, not too applied, and has nice doable but unsolved exercises.

- *"Algorithm design" by Kleinberg and Tardos:* is mostly about combinatorial and graph algorithms. It's well motivated, rigorous, shows some applications, and has a few doable and solved exercises, as well as nice hard unsolved ones. One of my **favorite** algorithm books.

- *"Exact Exponential Algorithms" by Fomin and Kratsch:* is about exact exponential algorithms. It's a bit motivated, mostly rigorous, not applied and has unsolved exercises.

- *"The Design of Approximation Algorithms" by Williamson and Shmoys:* is about approximation algorithms. It's motivated, rigorous, not applied, and has nice hard unsolved exercises.

- *"Combinatorial Optimization: Algorithms and Complexity" by Papadimitriou and Steiglitz:* is about optimization on graphs, linear and integer programming, as well as complexity theory. It's a bit motivated, mostly rigorous, not applied, and has nice hard unsolved exercises.

- *"A course in convexity" by Barvinok:* is about convexity, with a focus on geometry (polyhedra and ellipsoids) and combinatorics (discete geometry and Erhart theory) but is till of interest for optimization, as it tackles linear programming and the ellipsoid method. It's a bit motivated, has nice figures and examples, is rigorous, has some applications, and has nice doable unsolved exercises incorporated to the lecture material. One of my **favorite** discrete geometry books.

- *"Combinatorial optimization: Networks and Matroids" by Lawler:* is about shortest paths, network flows, matchings and lots of algorithmic topics on matroids. It's a bit motivated, mostly rigorous, not applied, and has unsolved exercises I didn't look at yet.

- *"Combinatorial optimization: Theory and Algorithms" by Korte and Vygen:* is about combinatorial optimization, LPs, IPs, and has lots of advanced topics such as semidefinite programming and multi-commodity flows. It's a not motivated, skips a lot of arguments, not applied, and has hard unsolved exercises. Makes for a good reference, but don't try to learn with it.

- *"A first course in the calculus of variations" by Kot:* is about the calculus of variations. It's motivated, mostly rigorous, has many applications, and has nice doable unsolved exercises.

- *"Discrete and computational geometry" by Devadoss and O'Rourke:* is about computational geometry with some aspects of discrete geometry (for example on triangulations). It is motivated, rigorous, has beautiful figures and some applications, and nice doable unsolved exercises. Makes for a good introduction. One of my **favorite** computational geometry books.

- *"Lectures on Polytopes" by Ziegler:* is about combinatorial aspects of polytopes, but contains some content linear-algebra aspects of them, as well as some content on oriented matroids. It's sometimes motivated, rigorous but skips way to much, not applied and has hard unsolved exercises.

- *"Convex Optimization" by Boyd and Vandenberghe:* is about convex analysis and optimization (in finite dimensional space) with algorithms. It's motivated, kind of rigorous, has many applications, and has unsolved exercises I didn't look at yet.

- *"Discrete Mathematics" by Aigner:* is about combinatorics, graphs and number theory but contains a chapter on linear programming. It's sometimes motivated, rigorous, not applied, and has nice **solved** exercises.

- *"Advanced microeconomic theory" by Reny and Jehle:* is about microeconomics, convex analysis and optimization and game theory. It is more mathematical then some math books. It's not very motivated, rigorous, has some applications, and exercise with hints/solutions.

- *"Algorithmic Game theory" by Nisan, Tardos, Roughgarden, Vazirani:* is about game theory and mechanism design, from an algorithmic viewpoint. It's not very motivated, not very rigorous, has some applications and unsolved exercises I didn't look at yet.

- *"Polyhedral Geometry and Linear Optimization"* and *"Lecture Notes on Lattice Polytopes"* by Paffenholz (free lecture notes available online) are about polyhedra, lattices, and optimization and some combinatorics regarding them. They're not too motivated but have nice figures and a clear approach, rigorous, not applied, and have no exercises.

- *"Geometric Algorithms and Combinatorial Optimization"* by Grötschel, Lovasz and Schrijver is mostly about the ellipsoid method and it's use in combinatorial optimization. It's not too motivated, rigorous, not applied and has no exercises.

- *"Integer Optimization and Lattices"* by Rothvoss (free lecture notes available online) is about lattice geometric methods for solving IPs. It's a bit motivated, rigorous, not applied and has unsolved exercises I didn't get to look at yet.

- *"Game theory: a playful introduction"* by Devos and Kent is about (classic) game theory. It's motivated, rigorous, not applied and has unsolved exercises I didn't get to look at yet. One of my **favorite** game theory books.

- *"Introduction to linear optimization"* by Tsitsiklis and Bertsimas is about linear and integer-linear programming, with content on interior point and ellipsoid methods. It's motivated, rigorous, not applied and has unsolved exercises I didn't get to look at yet. One of my **favorite** linear programming books.

- *"Linear programming"* by Chvátal is about linear programming. It's motivated, rigorous, not applied and has exercises which a partially solved with undetailed solutions.

2 Combinatorial algorithms and optimization

2.1 Shortest Paths

"Combi opt" Lawler: ch 3 sec 13

Recursion: Floyd-Warshall
We're interested in how a navigation device finds the shortest path from a starting point to a destination in a road network. To study this, we model the road network by a weighted directed graph G with edge-weight function w: the vertices represent intersections, the graph is directed as roads my only be driven in one direction and the weights represent travel distance or travel duration between intersections.

We which to find an algorithm that determines a path P from vertex i to vertex j that minimises the total weight $\sum_{e \in E_P} w(e)$ of all such paths.
Technically, we should ask for a walk instead of a path. In our context however, we assume that w is positive, so that eliminating the cycles in a walk yields a path of lesser weight then the walk: therefore, there is a minimum walk that is a path, so we keep the problem constrained to paths.
Positivity of weights also guarantees that a minimum walk exists! Without it, the graph may contain a cycle of negative weight. If a walk contains such a negative cycle, we can extend the walk by repeatedly cycle around it, getting paths of successively lower weight: in such a case, a minimum walk doesn't exist.

We now investigate a recursive approach to the problem. There are two variations of such an approach: either we consider recursion on the edges, or on the vertices. In both cases, we should try to see if one can recover a shortest i-j-path from the same knowledge, but on the graph with an edge or a vertex removed. Here, we'll study the recursion over the vertices.

Assume, we know how to find shortest u-v-paths in the graph $G \backslash x$ for some vertex $x \in G$: how can we find the shortest u-v-paths in G ?
In G, a u-v-path either passes through x or it doesn't. By recursion, we can assume that we know the shortest path of the second kind. To relate those of first kind to our knowledge from recursion, we split such a path P into two paths by cutting at x: if a and b are the predecessor and the successors to x on P, we obtain $P_{u \to a}$ and $P_{b \to v}$, two paths of $G \backslash x$.
We also split the weight of P into $\sum_{P_{u \to a}} w(e) + w(a,x) + w(x,b) + \sum_{P_{b \to v}} w(e)$. By recursion, we can assume that we know minimal u-a-paths and b-v-paths. So all possible candidates for shortest u-v-paths in G are the minimal paths of their respective category: those that don't cross x and those that use a minimal path in $G \backslash x$ from u to an a in the in-neighbourhood $N_{in}(x)$ of x in G, cross x through $a \to x \to b$ and reach v from b with a minimal path in $G \backslash x$, for some b in the out-neighbourhood $N_{out}(x)$ of x in G. So the task reduces to finding the candidate of minimal weight.

To clean this idea up, we take a bottom-up dynamic programming approach. We'll label the vertices in some order, associate to them two types of data and iteratively update this data, iterations corresponding to the addition of vertex k to be considered in the paths, for k climbing $[|V|]$. The first data associated to a vertex will be the length of a shortest path from i to j in the sub-graph induced by the vertices i, j and 1 to k, denoted by d_{ij}^k (so we iterate over the possible internal vertices of i-j-paths).

We also want to keep track of the paths of minimal weight. One way to do this is by storing the path as a list/tuple, one for each (i,j). However, there is a much more efficient way of doing this that requires a little theory:

Sub-structure optimality of optimal paths:

If a path P minimises (or maximises) a weight $\sum_{e \in E_P} w(e)$ under the constraint of fixed endpoints, then its subpaths minimise (or maximises) this weight for fixed endpoints too.

Proof: If a minimal i-j-path P had a u-v-subpath Q that wasn't a minimal u-v-path, then we could modify P so as to have less weight, contradicting its minimality. If R was a lighter u-v-path then Q, then we could build a new i-j-path by following P from i to the first vertex of R on P, then follow R until the last vertex that R and P have in common, and finish by following P until j. This is a path (not just a walk), as repetition of vertices would lead to contradictions in our construction. It's weight is less then $w(P) - w(Q) + w(R)$ since we never follow Q as the switch to R happens at u and v in an extreme case. If $w(R) < w(Q)$, this will yield a lighter i-j-path.

By this sub-structure optimality, we only have to keep track of predecessors of vertices on optimal paths. For example, to find an optimal i-j-path, we would store the predecessor k of j on such a path, because the subpath from i to k is also optimal, so that if we had already stored this optimal path as a list indexed by (i, k), then concatenating j to this list yields a vertex-list of an optimal i-j-path.
So our second data will be p_{ij}^k, the predecessor of vertex j on an optimal i-j-path in the kth iteration.

Summing up:

Floyd-Warshall (assp):

We consider the distance d_{ij}^k from i to j using the first k vertices.
We update distances with $d_{ij}^{k+1} = \begin{cases} w_{ij}, \infty, 0 \ for\ the\ start\ k = 0 \\ min\left(d_{ij}^k, d_{i,k+1}^k + d_{i,k+1}^k\right) \end{cases}$ (compare current distance to that when passing the new node k+1). We update for all pairs of vertices for $|V|$ iterations.

Runtime $O(V^3)$

Ex.kSP: In a digraph with positive edge weights, we seek the k successively shortest paths between two endpoints, in the sense that the ith path is different from the $j < i$ first ones and has minimum weight among all paths, other then the $j < i$ first ones. So the some of the paths in this list may have equal weight (there may be multiple paths of minimum weight), but they are all different. Give an algorithm that solves the problem with runtime polynomial in the graph size and in the parameter k.

Ex.DMP: We given a connected graph and edge weights $w : E \to \mathbb{R}$. For vertices s and t, how do we find a s-t-path P such that the most expensive edge $\max(w(P))$ is minimum over all such paths ?

2.2 Spanning trees

Ex.FLST:
(i) For a subset $T \subseteq V$ of the vertices of a graph, is there a polynomial time algorithm for telling if there is a spanning tree of G in which all vertices of T are leaves ?
(ii) Same question, but for the existence of a tree for which all leaves are in T.

Ex.NT: We wish to find a spanning tree in a connected graph with given edge weights that has the globally narrowest edges. By "globally narrowest edges", we mean that the spanning tree T minimises $\max_{e \in T} w(e)$, over all spanning trees. How do we find such a spanning tree ?

Ex.PST: We're given a connected graph with an (not necessarily proper) edge coloring $c : E \to [k]$ and a preference on the colors: wlog. we prefer k to $k-1$ and so on. We seek a spanning tree of the graph such that the number t_k of edges with color k is maximum among all spanning trees, and the number t_{k-1} of edges with color $k-1$ is maximum among all spanning trees of the previous kind, and so on until 1. Find an algorithm that finds such a tree.

MST with degree lower bound:
We seek an MST among a special subset of spanning trees of a connected graph G: for some fixed $v \in V$ and some $k \leqslant \deg_G(v)$, we restrict our attention to trees for which $\deg_T(v) \geqslant k$, and seek the tree of least total weight of this kind.

We can switch weights to $w_\lambda = w - \lambda\chi_{\delta(v)}$ for a reduction $\lambda \geqslant 0$, which makes edges in $\delta(v)$ more desirable to Kruskal. If for a given λ Kruskal selects q edges of $\delta(v)$, then the output tree T_λ has weight $w(T_\lambda) = w_\lambda(T_\lambda) + \lambda q$. If we denote by T_k the MST among those for which $\deg_T(v) \geqslant k$, then we have $w(T_k) = w_\lambda(T_k) + \lambda \deg_{T_k}(v) \geqslant w_\lambda(T_k) + \lambda k$.
Now, if we could find a λ such that for it (and weights w_λ) Kruskal selects $q = k$ edges of $\delta(v)$, then we'd have $w(T_k) \geqslant w_\lambda(T_k) + \lambda k = w(T_\lambda)$ so that this tree T_λ is an MST among those for which $\deg_T(v) \geqslant k$ as well.

The problem is that its a priori unclear how Kruskal will select edges for a given λ and in particular how many of $\delta(v)$ will be selected. Even if we manage to find a λ such that the cheapest k edges are in $\delta(v)$, it is possible for more egdes of $\delta(v)$ to be added later by Kruskal.
The only case in which this is easy is the case $k = 1$ in which $\lambda = 0$ does the job. This is when the recursion idea may come back into play. If we know a λ_k and such that a T_{λ_k} outputed by Kruskal (for a specified tie braking) is an MST among those for which $\deg_T(v) \geqslant k$, can we find a λ_{k+1} such that the same thing is true for $k+1$?
Note that the entry candidates $\delta(v) \backslash T_{\lambda_k}$ each close a unique cycle in T_{λ_k}, on which they have maximum weight (otherwise, we could delete the maximum weight edge on that cycle and get a tree of lower weight verifying the degree lower bound). If for each $e \in \delta(v) \backslash T_{\lambda_k}$, we denote by f_e an edge not in $\delta(v)$ achieving the maximum over the cycle closed by e, then by increasing λ, it will at some point be more lucrative to swap out f_e for e. More precisely, if C_e is the cycle closed by e then for λ such that $w(e) - \lambda = w(f_e) = \max_{C_e \backslash \delta(v)} w$, this is the (critical) case, where we started with $w(e) - \lambda_k \geqslant w(f_e)$, for otherwise the swap would have been luctrative at that stage, contradicting that T_{λ_k} was an MST among those for which $\deg_T(v) \geqslant k$.

We therefore seek the maximum λ for which $w(e) - \lambda \geqslant w(f_e)$ for all $e \in \delta(v) \backslash T_{\lambda_k}$, knowing $\lambda \geqslant \lambda_k$. We denote this value by λ_{k+1}. If only one of the inequalities $w(e) - \lambda \geqslant w(f_e)$ is tight at λ_{k+1}, say the one

for e' we run Kruskal on weights $w_{\lambda_{k+1}}$ by considering e' before $f(e')$, so the same edges of T_{λ_k} are chosen until e' is chosen before $f(e')$, after which no other edges of $\delta(v)$ are chosen as for them, $w(e) - \lambda > w(f_e)$, so that f_e is chosen first, preventing e from being chosen. Therefore Kruskal constructs a tree $T_{\lambda_{k+1}}$ for which $\deg_{T_{\lambda_{k+1}}}(v) = k+1$, which we know to be an MST among those for which $\deg_T(v) \geqslant k+1$.
If multiple inequalities of form $w(e) - \lambda \geqslant w(f_e)$ are tight at λ_{k+1}, then we chose one to do the same as before and make sure to consider the $f(e')$ before e' for the other tight ones, to make sure the e' aren't added, this time due to tie breaking.

Minimum forest partition:
In this problem, we're given a graph $G = (V, E)$ and we're asked to find the smallest k such that E can be partitioned into at most k subsets I_i, each forming a forest on V. Doing this is always possible, as we can partition E into singleton partition sets made of single edges, which are forests, so that $k \leqslant |E|$. So if we can answer the decision version of the problem, we have to answer it at most $|E|$ times to find the optimal solution.

We'll solve this problem by specifying an algorithm of Edmonds for matroid partitioning.
We start by introducing k sets $I_1, ..., I_k$, initially empty, which we'll fill up until we can either tell that partitioning is impossible, or until we get the desired output, maintaining the invariant that they are disjoint forests.
A necessary condition for G to be partitioned into at most k forests is that G has not too many edges. More specifically, if we consider a partitionable graph and consider set of edges $A \subset E$, then if $r(A)$ denotes the size of the largest spanning forest of the graph induced by A, since $|A| = \sum_{i \in [k]} |A \cap I_i|$ and the $A \cap I_i$ are forests (as subsets of acyclic graphs are acyclic) in A, so that by definition $|A \cap I_i| \leqslant r(A)$, we have $|A| \leqslant k.r(A)$. We'll see that our algorithm for constructing such a partition works if the inequality $|A| \leqslant k.r(A)$ is satisfied for all $A \subset E$, and returns a set violating the inequality otherwise.

At each step of the algorithm, we consider an $e \in E$ not assigned to one of the partition sets, so that $x \notin \cup_i I_i^{(n)}$, where n is the number of iterations performed this far. We then look for an I_i that still has room to accommodate x, which is a priori the case when $|I_i| = r(I_i) < r(E)$ (where $r(E) = |V| - 1$, the size of a spanning tree if G is connected) . Such a partition-set must exist, for otherwise $|E| \geqslant |e \cup_i I_i^{(n)}| = 1 + \sum_{i=k} |I_i| > k.r(E)$ (assume the contrary and use dijointness), so that E violates the necessary condition for partition.
If for such an I_i, $I_i \cup e$ is acyclic, we can put e in I_i and move on. This might not be the case however. If we denote by $S(F)$ the graph of largest size that contains a forest F, which is the union of the graphs induced by the forests components/trees, then it could be that $e \in S(I_i)$. Since $|I_i| = r(I_i) < r(E)$ informed us that I_i still has room left, we'll look for a different I_j to accommodate e by looking for their part in S_1. We therefore proceed the same way, but focusing on S_1. We look for an $I_j \cap S_1$ (which is a forest as a subset) that still has room to accommodate x, which is a priori the case when $|I_j \cap S_1| = r(I_j \cap S_1) < r(S_1)$. Again, such a set must exist, for otherwise $|S_1| \geqslant |e \cup_j \left(I_j^{(n)} \cap S_1\right)| = 1 + \sum_{j=k} |I_j \cap S_1| > k.r(S_1)$, so that S_1 violates the necessary condition for partition.

TO CONTINUE (Edmonds papers or p.124 onward of "Combinatorial rigidity")

Minimum diameter spanning tree (MDST) problem:

Definition:

For a graph $G = (V, E)$ with positive edge weights $w : E \to \mathbb{R}_+$ and induced distance d_G, a **diameter** is the longest distance between two vertices in G, or a path achieving this distance, depending on context.

In the minimum diameter spanning tree (MDST) problem, we search a spanning tree of a connected graph G who's diameter is smallest among all spanning tree. It's a minimally connected sub-graph for which the longest path on it is relatively short.

We'll try to relate this problem to shortest path trees, as we know how to compute those. the connection is made with more attributes:

Definition:

For a graph $G = (V, E)$ with positive edge weights $w : E \to \mathbb{R}_+$ and induced distance d_G, the **eccentricity** of a vertex v is is the longest distance in G from v to any other vertex, or a path achieving this distance, depending on context. A vertex of a graph is a **center** of the graph if it has minimum eccentricity among all vertices.

Intuitively, a center should be at the middle of a diameter, which should be double the radius. We can solve the ASSP problem to compute the diameters, the eccentricity of each vertex, and from those the centers of the graph. Intuition tells us that on a MDST, the center should be close to all vertices, so that the paths from the center to the vertices should be shortest paths.

We'll try to find a tree with this structure by investigating the properties of a MDST. We consider a MDST T of G and one of its diameters D (of T) and look at the middle of this path, in the sense that we consider u_1 and u_2 so that if $D : d_1 \to d_2 \to ... \to d_n$, then $(u_1, u_2) = (d_k, d_{k+1})$ for the first k such that $w(D|_{d_1 \to d_{k+1}}) \geqslant \frac{w(D)}{2} = \frac{diam(T)}{2}$ (which exists as $(w(D|_{d_1 \to d_{k+1}}))_k$ is increasing from 0 to $w(D)$). Note that then $w(D|_{d_1 \to u_1}) \leqslant \frac{diam(T)}{2}$ (minimality of k) and also $w(D|_{u_2 \to d_n}) \leqslant \frac{diam(T)}{2}$, as $w(D|_{u_2 \to d_n}) = w(D) - w(D|_{d_1 \to d_{k+1}})$.

The graph $T \backslash (u_1, u_2)$ splits into two trees T_1 and T_2, so that $u_i \in T_i$.

We now build SPTs T_1' the subgraph of G induced by vertices $V(T_1)$ and T_2' on vertices $V(T_2)$ and consider the tree $X = T_1' \cup (u_1, u_2) \cup T_2'$. This is a tree as (u_1, u_2) closes no cycles. How does it's diameter compare to that of T ?
If the path P achieves $diam(X)$, then by denoting its endpoints by a and b, we disjoint cases on whether the endpoints are in different splits of the tree. If $a, b \in T_i$, then all of P is contained in T_i and $diam(X) = d_{T_i}(a, b) \leqslant diam(T_i) \leqslant diam(T)$. This last inequality is not obvious, even if T_i is a subgraph of T: for $G = K_n$ and $w = 1$, we have $diam(G) = 1$ but for all spanning trees T of G, $diam(T) \geqslant 2$, as there is at least one triplet of vertices with only two edges among them. However, we have $d_{T_i}(a, b) \leqslant d_{T_i}(a, u_i) + d_{T_i}(u_i, b) \leqslant 2w(D|_{d_1 \to u_1}) \leqslant diam(T)$, by choice of (u_1, u_2) and the fact that for all $a \in T_1$ (and $b \in T_2$) we have $w(P|_{a \to u_1}) \leqslant w(D|_{d_1 \to u_1})$ (and $w(P|_{u_2 \to b}) \leqslant w(D|_{u_2 \to d_n})$).
Indeed, if $w(P|_{a \to u_1}) > w(D|_{d_1 \to u_1})$, the the path $P|_{a \to u_1} \cup (u_1, u_2) \cup D|_{u_2 \to d_n}$ would be a shortest path in T (as all paths from a to d_n must use (u_1, u_2)) with greater length then its diameter, a contradiction.
If we have $a \in T_1$ and $b \in T_2$, then P must use (u_1, u_2), and $P|_{a \to u_1}$ and $P|_{u_2 \to b}$ achieve $d_T(a, u_1)$ and

$d_T(u_2, b)$ respectively (sub-path optimality).
So in this case too, $diam(X) = w(P|_{a\to u_1}) + w(u_1, u_2) + w(P|_{u_2\to b}) \leqslant w(D) = diam(T)$, with the same bound as previously.
Finally, since T is an MDST, we must have $diam(X) = diam(T)$, so that X is an MDST.

We could use this observation to consider all edges $(u_1, u_2) \in E$, all bipartition of vertices so that $u_i \in V_i$ and compute SPTs for T_i, join them by (u_1, u_2) and compute their diameter. Among these trees is an MDST, as we go over the (u_1, u_2), $V(T_1)$ and $V(T_2)$ of the hypothetical MDST T eventually. Yet, this family is exponential due to the enumeration of all bipartitions.

We'll now show that a certain tree based entirely on (u_1, u_2) is also an MDST.
We couldn't find motivation for the following step, as our source doesn't motivate (and prove) much. We consider the quatity $q(v) = d_G(v, u_1) - d_G(v, u_2)$. This quantity has the property that on a shortest path P from v to u_1, $q(v)$ decreases along P. Indeed, if w is on P, then $d_G(v, u_1) - d_G(v, u_2) = d_G(v, w) + d_G(w, u_1) - d_G(v, u_2)$ and by using $d_G(v, w) - d_G(v, u_2) \geqslant -d_G(w, u_2)$ (via triangular inequality) we get $q(v) = d_G(v, u_1) - d_G(v, u_2) \geqslant d_G(w, u_1) - d_G(w, u_2) = q(w)$. Similarly (consider $-q$), one can prove that q increases along shortest paths to u_2.

We can use this to define sets of vertices V_i so that there is a shortest path in G or each $v \in V_i$ to u_i that uses only vertices of V_i. For V_1 of the form $\{v \in V : q(v) \leqslant k\}$, the vertices on a shortest path from $v \in V_1$ (if there are any, depending on k) to u_1 are still in V_1, and for $V_2 = V \backslash V_1 = \{v \in V : q(v) > k\}$, the vertices on a shortest path from $v \in V_2$ (if there are any, depending on k) to u_1 are still in V_2. To insure both sets are non empty, we can have $u_i \in V_i$ setting $k = d_G(d_1, u_1) - d_G(d_n, u_2)$ (providing $u_1 \in V_1$) and adding the conditions $d_G(d_1, u_1) < d_G(d_n, u_2)$ (providing $u_2 \in V_2$).

We now build Y by building SPTs on $V(V_i)$ with root u_i and connecting them with (u_1, u_2). Y is a spanning tree as the SPTs only had edges in V_i respectively, so no cycles are getting closed. A diameter of Y may either use (u_1, u_2) or not. In the first case, it has we can decompose it's length into $diam(Y) = d_Y(a, u_1) + w(u_1, u_2) + d_Y(u_2, b)$ for endpoint $a \in V_1$ and $b \in V_2$. But by construction, $d_Y(a, u_1) = d_{SPT(u_1, V(V_i))}(a, u_1) = d_G(a, u_1)$ and similarly for $b \in V_2$, so that since $d_G \leqslant d_T$, we have $diam(Y) \leqslant d_T(a, u_1) + w(u_1, u_2) + d_T(u_2, b)$ and finally $diam(Y) \leqslant d_T(d_1, u_1) + w(u_1, u_2) + d_T(u_2, d_n) = diam(T)$, using a previous property about T, and the fact that $a \in T_1$ and $b \in T_2$..... (remains to be proven).

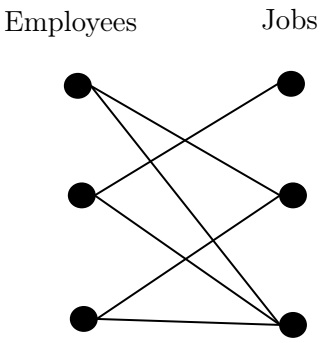
TO COMPLETE: case of diam = 2radius, spt of center.

2.3 Matching

min weight, blossoms, gale-shapley

Bipartite matching Suppose you're managing a team that's developing a video game. You have a set of employees and a set of intermediate jobs to be performed by them to make the game. In your team, employees have different skills and different levels of experience, so that some might not be able to perform certain jobs (or at least not well or not in a reasonable time span). For example, you could have two graphics designers that know two of the following skills: character design, landscape design and animation.
How could you get a better picture of the situation ?

On way to represent the situation is the following: we represent employees and jobs by the vertices of a graph, grouping employee vertices and job vertices together, and we add an edge between an employee and a job if the job can be performed by that employee.



This graph will be bipartite:

Bipartite graph:

A graph is bipartite if its vertices are partitioned into A and B and for all edges $\{u, v\}$ of the graph, $u \in A$ and $v \in B$ or vice versa.

We'll assume that one employee can work on one job only and that a job can be worked on by one employee only.
Your goal is to get all the jobs done, if doing so is possible, by a clever assignment of employees to jobs.
So when we assign employee u to job v, we can represent this on our graph by highlighting edge $\{u, v\}$. By our assumptions, each vertex can be incident to at most one highlighted edge. We are therefore looking for a:

Matching:

A matching M is a subset of edges E of a graph such that no two edges of M have a vertex in common.

To be precise, we are looking for a **maximum matching**, which is a matching of greatest size among all

possible matchings. If all employees and jobs are matched in such a maximum matching, then we'll have an assignment for which all jobs can be completed. Otherwise, there is no way of completing all jobs with our team. So how do we find a maximum matching in a bipartite graph ?

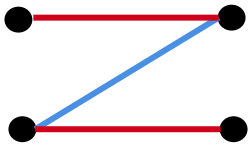
A first idea could be to add edges with different endpoints until all edges have been considered.

Maximal matching
Input: a graph (V, E)
1 Start with $M := \emptyset$
2 Take note of the matched vertices $C := \emptyset$
3 For $\{u, v\} \in E$:
4 If $u \notin C$ and $v \notin C$:
5 Add $\{u, v\}$ to M
6 Add u and v to C
Output: a maximal matching M

There are no spelling mistakes in this description: we output a **maximal matching**, which is a matching to which no edges from E can be added to produce another matching.

Before getting into the subtle difference between maximum and maximal, we'll check the algorithm.
The loop invariant is that M stays a matching: at each iteration, we add a edge only if its endpoints aren't matched already. After termination, when all edges have been considered, no more edge can be added to M and so that it remains a matching. To see this, recall the iteration in which the edge to be added was considered: we discarded it, as one of its endpoints was already matched. So it's endpoints are still matched in M and we can't add it without breaking the matching property.

Are maximal matchings also maximum ?
If they were, our previous algorithm would solve our problem. Unfortunately, they aren't. The problem lies in the fact that an unfortunate ordering of the edges in our algorithm can led to sub-optimal choices, as the following example shows:



In the case of this Z-shaped graph, the matching represented by the blue edge is maximal: adding any of the two remaining edges is impossible, as in both cases, one of their endpoints is already matched. The red edges represent a matching of greater size, so the blue matching is maximal, but not maximum.

Ex.MaMu: There is however an interesting relation between the sizes of maximal and maximum matching: if M_a is maximal and M_u is maximum, then $|M_u| \leq 2|M_a|$. Prove this (later).
So the maximal matching algorithm provides us with an upper bound on the size of a maximum matching.

If we analyse the previous counterexample, we may get an idea as to how to improve on a given matching. This way, we don't have to start solving the problem from scratch again.

If we had a way to detect that the blue edge's endpoints are connected to two different vertices that aren't matched by the blue matching, we could then perform a switch operation that discards the blue edge and adds the red ones. Let's attempt to generalise this.
To increase the size of the matching, we need to increase the number of matched vertices. So we can start

by looking for vertices that haven't been matched by our original matching. If we find one, we look at its incident edges and disjoin two cases: either the other endpoint of that edge isn't matched, in which case we can add that edge to increase the matching, or it isn't, in which case there is an edge from the matching incident to the other endpoint.
The previous counterexample shows that the second case isn't hopeless: we follow the edge from the matching and hope that the next endpoint on our path is connected to an unmatched vertex.
We can iterate this reasoning and hope to end up with the following structure:

Augmenting paths:

An augmenting path P for a matching M is a path who's start- and endpoint aren't matched by M and who's edges alternate between being in $E\backslash M$ and in M.
So in particular, augmenting paths start and end with edges from $E\backslash M$ and have odd length.

For example, the previous counterexample represented an augmenting path of length 3.

Given an augmenting path, we can improve the matching:

Augmenting matchings:

Given an augmenting path P for a matching M, the symmetric difference $M\Delta P$ (the edges that aren't common to both) is a matching of greater size then M.

Proof: To verify that we end up with a matching, we look at the vertices and disjoin cases. If the vertex isn't on P or is one of its endpoints, it will be incident to the at most one edge of $M\Delta P$ since M is a matching and the endpoints of P aren't matched by M. If it is in the interior of the path P, then it's incident to and edge that is common to M and P, which will be deleted, and also to an edge that is in P only, which will be added, but to no other edges, as M is a matching and P a path. So these points will still be matched in $M\Delta P$.
By the nature of augmenting paths, there is one more edge of $E\backslash M$ on them then from those of M.
So $|M\Delta P| = |M| + 1 > |M|$ and we've obtained a bigger matching. □

So we can look for augmenting paths in a given matching to improve on it. We can then repeat this step, knowing that at some point, there can't be any augmenting paths anymore, since the size of the matching keeps increasing, but is bounded by $|E|$.
Before clarifying how we can find augmenting paths (if there are any) in practice, we should address the elephant in the room. If we can't find any further augmenting paths, is the matching maximum ?

Characterising maximum matchings through augmenting paths:

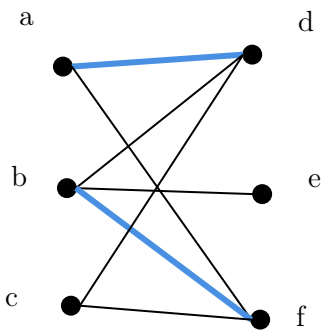
M is a maximum matching $\Leftrightarrow$ there are no augmenting paths for M.

Proof: The contrapositive states that M isn't a maximum matching $\Leftrightarrow$ there is an augmenting path for M, and it's what we'll prove. We've already taken care of $\Leftarrow$ in the previous result.
Proving $\Rightarrow$ requires more care.
We'll investigate the difference between matchings M and M' with $|M| < |M'|$ in hope of finding some way to augment M. In fact we'll study the difference quite literally: when looking at the edges of $M\Delta M'$

only, the vertices of the graph come in 3 categories: either they aren't incident to any edges, or they are incident to 1 or 2 edges, where in the last case, the two edges have to come from different matchings.
So if we walk along the edges of $M\Delta M'$, we will trace out nothing but disjoint cycles or paths, depending of whether the walk ends where it started or ends in a vertex of degree 1. All other walks are impossible, as one would have to encounter a vertex of degree at least 3.
The cycles have to be even, since they can only alternate between edges of $M\backslash M'$ or $M'\backslash M$. So on them we count as many edges of $M\backslash M'$ as those of $M'\backslash M$.
The same is true for even paths. But for odd paths, things start to get interesting. These paths are augmenting paths for the matching which leaves the paths endpoints unmatched. So if we find an augmenting path for M we have the result we looked for. Does there have to exist such a path ?
If there wasn't, either all paths would be even, so that by our counting $|M| = |M'|$, or all odd paths would their majority of edges in M, so that $|M| > |M'|$. But $|M| < |M'|$ by assumption, so this can't happen. There must exist an augmenting path for M. □

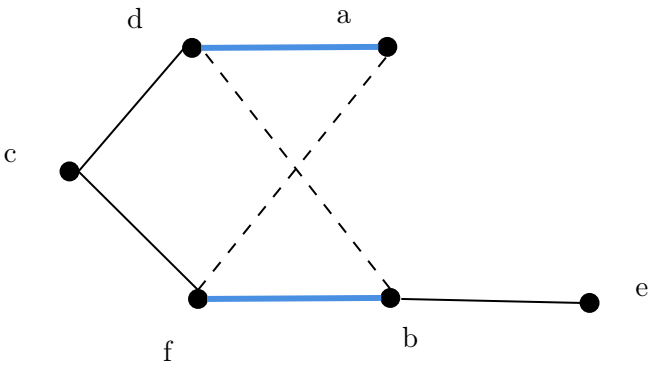
This is great news! It means that the approach of successively improving matchings with augmenting paths until there aren't any will output a maximum matching.
So all that's left for us to do is to develop a way of systematically finding augmenting paths or determining that none exist.

For example, we consider the following bipartite graph and matching (blue edges):



Augmenting paths start and end at vertices that aren't matched by the matching. So the first step will be to keep track of the un-/matched vertices. In the example above, we could start at c, which is unmatched. Next, we have to find a path that alternates between edges on and off the matching and ends at another unmatched vertex. We will do so with a graph search close to breadth-first search.

First, we consider the neighbors of c, d and f. They are matched, so we follow the unique edges to get to a and b respectively, ignoring the other edges. Finally, a only leads us to a vertex we've already visited, while b leads us to an unmatched vertex, e.



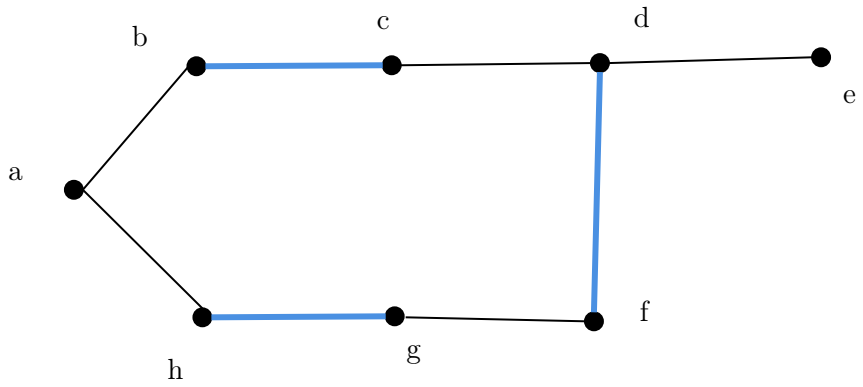
So we find the augmenting path (c, f, b, e). How do we proceed in general ?

Here's an algorithm that looks for augmenting paths starting at v.

Augmenting Paths
Input: a graph (V, E) and a matching M (and a set C of vertices matched by M)
1. Check if $V \backslash C$ has size at least 2 and take an unmatched vertex v from it.
2. Construct a tree routed at v as follows:
3. Add the neighborhood of v
4. For the leaves u of the tree, consider two cases:
5. If the the distance to v is even, add the unmatched edges of $\{u, w\}$ that lead to vertices not already in the tree. Then check if one of the w is unmatched. If it is, the w-v-path is augmenting, so we return it. If the distance is odd (and by iterative assumption, u is matched), add the unique edge of u in the matching M to the tree.
6. Iterate step 5. until all vertices reachable from v have been considered (that is, if there are no more unmatched edges leading to new vertices in step 5.)

We construct a tree, layer by layer, alternating between edges form the matching and edges not in the matching. The immediate question is: if an augmenting path starting at v exists, does our algorithm find one ?

To see that it does, notice that the tree we construct contains all the vertices that are reachable from v and that all paths on it are alternating between edges form the matching and edges not in the matching. The fact that we are building a tree requires a subtle justification of great importance.
The structure is surely connected, as we repeatedly get neighborhoods, so that all vertices have a path to v, by induction. Can there be cycles ?
When considering the edges that aren't in the matching, we exclude those that lead to vertices that already are in the tree, preventing a cycle from being closed. But we don't check this for the edges in the matching, which can lead to problematic situations, such as this one:



Here, our construction would fail once we reach d and f. The failure is due to the fact that the edge from the matching is connecting two vertices from our tree, thereby closing a cycle.
Even more embarrassingly, there is an augmenting path we fail to get: (a, h, g, f, e).
Can we fix this ? Or does this problem ever appear in our context ?

Will investigate how to fix this problem for general matchings in the next chapter. Now however, we're still in the setting of a bipartite graph. By investigating carefully, we notice that the cycles closed by edges of the matching must be odd, as the paths to the endpoints of the closing edge are augmenting, thus odd. But bipartite graphs can't contain odd cycles: if there was one, it would have to contain an odd number

of vertices, which belong to one of two partition sets. So two neighbors on the cycle would have to be in the same partition set, yet this is impossible by bipartiteness.

Ex.BOC: It turns out that the property of not having odd cycles implies that the graph is bipartite. Show how to construct a bipartition from this property.

So in the case of a bipartite graph, we don't close cycles when adding a layer to our tree corresponding to edges from the matching. We therefore conclude that we really are building a tree.

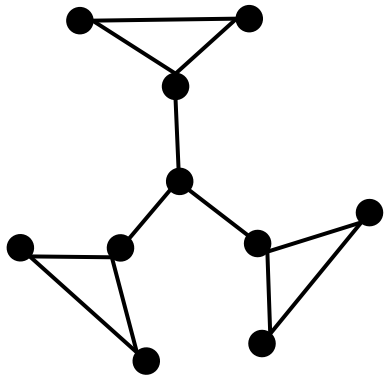
We now can settle the main question of whether we find an augmenting path starting at v, if one exists. An augmenting path P starting at v has its first edge in the tree, since the tree contains the neighborhood of v. We then show that all further vertices on P can be reached on our tree inductively. The induction step disjoins two cases. For the current vertex on P, if the previous edge on P is from the matching, then there is a unique predecessor, the vertex it's matched with, which is both on the tree and on the path. If the previous edge isn't from the matching, it may also be in the tree but if it isn't, it's only because there is another edge from the tree that connects to the endpoint. To be precise, the edge would have been added in step 5 of our algorithm, since it's a neighbour to a matched vertex, only if it didn't close a cycle: this implies that the vertex was already in the tree.

In conclusion, we can there fore use the following algorithm to find maximum matchings in bipartite graphs, as is guaranteed by the characterisation of maximum matchings by augmenting paths.

Bipartite Matching
Input: a graph (V, E)
1. Start with a matching, for example the output of **Maximal Matching**, or $\emptyset$,
2. Use **Augmenting Paths** starting from an unmatched vertex v
3. Augment the matching by the augmenting path (take the symmetric difference)
4. Reiterate 2. and 3. until no more augmenting paths can be found for all exposed vertices.

Output: a maximum matching M

As a final remark, note that it is possible that two unmatched vertices exist, but no augmenting path does. This may come a bit as counter-intuitive, but it is possible:



In a matching of this graph, we may take only one edge per triangle, leaving one vertex unmatched. The only possibility for that vertex to be matched is when we match it to the center of the "wheel", which we may do once at most. So there will always be at least two unmatched vertices in a matching, in particular in a maximum one, which doesn't have augmenting paths.

Ex.EV: When looking for augmenting paths, we've looked for one starting at all exposed vertices in

each iteration. But is this necessary ? If no augmenting path starting at an exposed u was found in one iteration, could it be the start of an augmenting path in the next ?

We conclude this section with some nice exercises on bipartite matching:

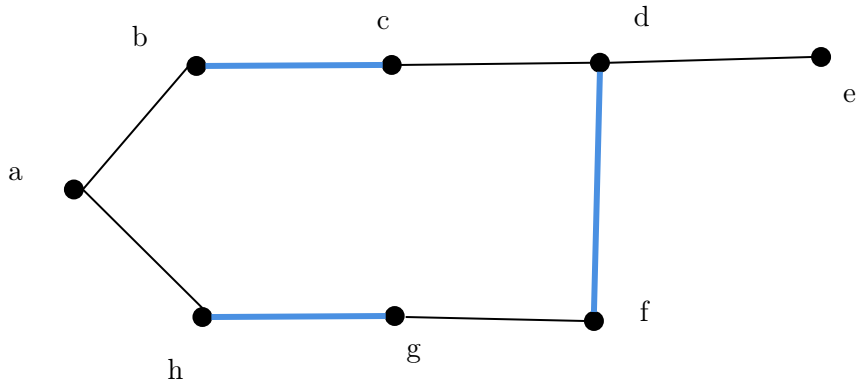
Ex.MD: We're given some planar shape made of identical square tiles: we can model this by a finite subset of $\mathbb{Z}^2$ by identifying squares with their lower left corner. We're asked to cover this shape with dominoes such that as many tiles are covered as possible. How do we proceed ?

General matching

We now investigate how to fix the problem we could afford to ignore in the previous section, because the graph was bipartite. This will allow us to find maximum matchings in non-bipartite graphs.

Matching in general graphs is of interest when the objects to be matched aren't in two categories (employees and jobs in our previous example). For example, consider a school class that is playing badminton in 1 vs. 1 matches. Student pairs inform the sports teacher that they wish to face each other in a match: based on these preference, how can the sports teacher get as many desired matches playing simultaneously ?
If we represent the students by vertices connected by an edge if the corresponding pair wishes to face off in a match, we are looking for a maximum matching in a general graph, which will correspond to the student pairs we can have playing simultaneously.

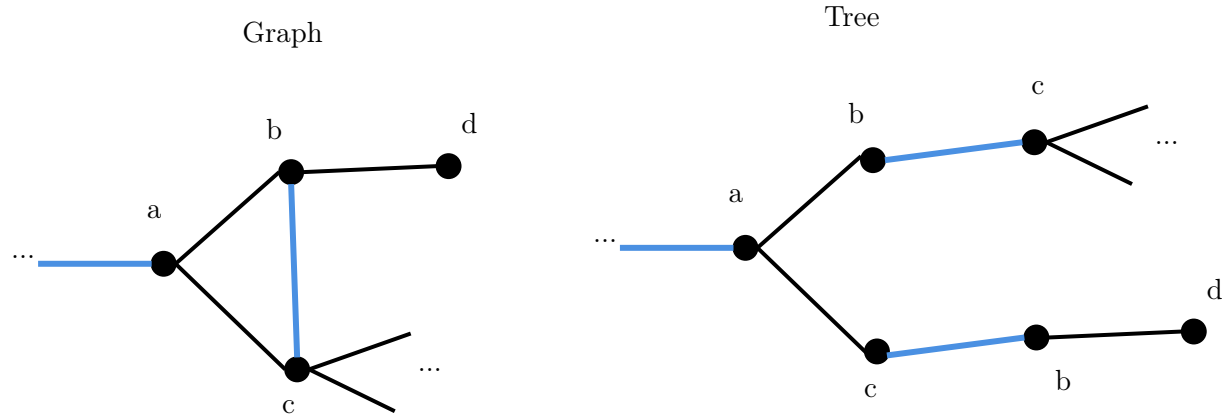
Recall the problem in our approach to find augmenting paths. We ran into trouble when an edge of the matching connected two vertices already in the tree. This edge of the matching closed a cycle in our tree, which is called a **blossom**. Blossoms have a particular structure: they are odd cycles that are alternating between edges from the matching and edges not in the matching, except at a single point, called the base of the blossom. The name comes from a particular way to deal with them, which we won't follow here: blossoms are shrunk at first, and then we un-shrink them (they "bloom") once we've found an augmenting path, and handle this un-shrinking so as to maintain the path.
In such a case, an augmenting path could still exist, as was shown in the example:



What do we do in such a situation ?
What can be said about the augmenting paths passing through such cycles ?

The augmenting paths passing through blossoms in the graph will "jump branches" in the tree we use to search for augmenting paths. For example, in the above figure, the alternating path of the lower branch, from a to f will jump to the higher one, that from a to d to continue.

One way to fix this is to "double" the edge in the tree, as we represent here:



This helps getting a clear picture. It also maintains the whole point of the algorithm: if the graph has an augmenting path, we can find it in the tree we construct, rooted at the start of the augmenting path. The argument given to see this is the same as in the previous section, since the matched edge will always be added to the tree, possibly twice if it connected two vertices already in the tree.

We can fix our algorithm "**Augmenting Paths**" with one line of pseudo-code:

5. If the endpoints of the matched edge are both in the tree, add it twice to the tree with the order of the endpoints reversed.

Ex.BlBl: Though our "doubling" of the edge in the tree fits in some way the image of the blossom blooming, we'll ask you to prove a property that has given the name to blossoms. To do so, we define G/B for a blossom B wrt. M to be the graph with the vertices of B repaced by a single vertex b, such that all edges inicident to vertices of B are now incident to b. Show that M is still a matching in G/B and that if G/B has an augmenting path wrt. M, the so does G.

Minimum edge cover:
Consider the problem of finding the minimum size set of edges of a graph, so that all vertices are incident to at least one edge of the set. A first observation to make about such a set is that it can't contain any path of length greater the 3: if it did, we could delete a middle edge of the path and retain the edge cover property, as the endpoints of the delete edge are still covered by the neighbouring edges of the path; this would contradict minimality. This means that a minimum edge cover is a node-disjoint union of stars.
If we remark further that by selecting one edge per star we obtain a matching who's size is the number of stars, we can develop the following intuition. Having stars with many edges seems inefficient, as all these edges are used to cover the same central node. Also, having many stars seems to decrease the number of edges per star. So if we had a way of turning matchings into edge covers, we could look for a maximum matching, and hope that the resulting edge cover has few egdes.
We can turn a matching into an edge cover, by doing the following for each uncovered vertex: we add an edge linking it to one of the matched vertices. To see that doing this is always possible, we must actually assume that the graph has no isolated vertices and that the matching is maximum. Then, each vertex has an edge incident to it, which leads it to a matched vertex, as otherwise, this edge could have been added to the matching, contradicting it's maximality.
If we denote by m the size (number of edges) of any matching and u the number of unmatched vertices,

then the graph has $|V| = 2m + u$. If M and U represent these quantities for a maximum matching then we have $M \geqslant m$. These in-/equations imply that $m + u \geqslant M + U$. Here $M + U$ is the size of the edge cover we built from the maximum matching. If we recall that we can obtain a matching of any minimum edge cover by selcting an edge per star, then if m is the size of this matching, $m + u$ is the size of a minimum edge cover and we've just found that $M + U$ is also minimum. So the edge cover we built from the maximum matching is a minimum edge cover.

2.4 Network Flows and Minimum Cuts

Ex.MCSA: Show that cuts are subadditive, in the sense that $u(A \cup B) + u(A \cap B) \leqslant u(A) + u(B)$, for positive capacities. Deduce also that $u(A \backslash B) + u(B \backslash A) \leqslant u(A) + u(B)$. What consequence does this have if A and B represent minimum s-t-cuts ?

Flow decomposition:
If we picture the flow as a fluid, each fluid particle of a s-t-flow should follow an s-t-walk in the graph. We have the following flow decomposition theorem:

Flow decomposition:

For an s-t-flow f in a digraph G, there are a collections P and C of s-t/t-s-paths and cycles respectively, in number $|P \cup C| \leqslant |E|$, and flows f_p and f_c with support in $p \in P$ and $c \in C$ such that $f(e) = \sum_{w \in P \cup C : e \in w} f_w$ and the flow value is $\sum_{p \in P_{s,t}} f_p - \sum_{p \in P_{t,s}} f_p$, where the first sum ranges over the s-t-paths, and the second over the t-s-paths. One can compute P and C in $O\left(|E|^2\right)$ time. If f is integral, so are the flows on the paths and cycles.

These may not be the augmenting paths from Ford-Fulkerson, since they were paths in the residual network.
Proof: We iterate the following procedure, that recovers paths and cycles and decreases the s-t-flow. We consider the graph induced by the edges for which $f(e) > 0$. We start at any such edge and extend it to paths of increasing length by adding edges to its endpoints. After at most $|E|$ addition of edges, we either close a cycle or have both endpoints such that there are no edges to continue along them. In the first case, we add this cycle c to C and set $f_c = \min_{e \in c}(f(e))$ (the bottleneck), and reiterate on flow $f - f_c \chi_c$ (it's a flow as balance is conserved and capacity satisfied). In the second case, we note that the endpoints can only be s or t since we're dealing with an s-t-flow, so that this is an s-t/t-s-path p that we add to P, setting $f_p = \min_{e \in p}(f(e))$. Again $f - f_p \chi_p$ is an s-t-flow and we iterate on it. Note that if f is integral, so are the f_p and f_c. At each iteration, the number of edges for which $f(e) > 0$ decreases by at least one, by the number of edges in which the bottleneck is attained, so that we iterate at most $|E|$ times. Finally, an edge is last considered when $f(e) = \sum_{w \in P \cup C : e \in w} f_w$.

All pairs minimum cut and Gomory-Hu trees:
We're interested in finding an algorithm that determines all capacities of minimum st-cuts and one such cut, for all pairs of vertices s and t. We'll try to do better then the brute force approach of $\binom{|V|}{2}$ minimum cut computations.

A special case where this problem seems easy are trees T. For A representing an s-t-cut so that $s \in A$, and for P the unique s-t-path in the tree, we see that $u(A) \geqslant \sum_{\{w,v\} \in P ; w \in A, \notin A} u(\{w, v\})$ by positivity. In particular $u(A) \geqslant \min_{\{w,v\} \in P}(u(\{w, v\}))$. But for the edge e achieving this minimum, the vertices of $T \backslash e$ splits into two connected components U and W, so that $u(U) = u(W) = u(e)$, one component containing s and the other t. So U is an s-t-cut, and $u(A) \geqslant u(U)$, so that it is a minimum s-t-cut.
So to find the minimum cut in a tree, we just have to find the minimum capacity on unique s-t-path and

consider the bipartition obtained by deleting a corresponding edge.

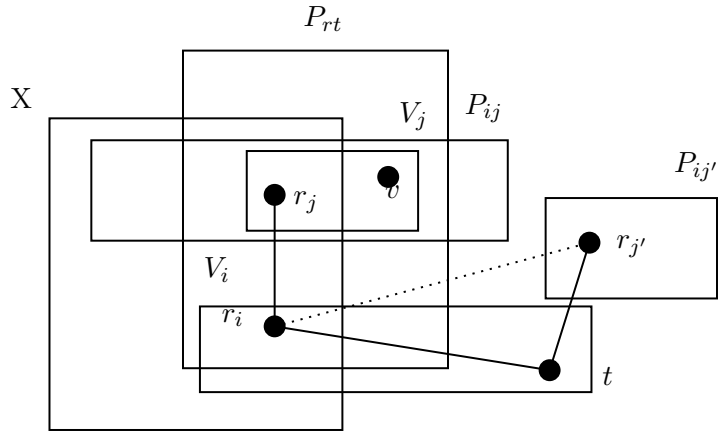
FIX FORM HERE: Williamson or Cook or the original paper, with figures like the one I attempted below to understand what the fuck is even going on in those proofs.

The idea behind the Gomory-Hu tree T_{GH} is to construct a tree on the vertices V, but who's edges E_{GH} are possibly different from that of E, that has the property that the biparitition obtained by deleting edge $\{u, v\} \in E_{GH}$ represents a minimum u-v-cut in the graph G.
We can show that this tree is enough information to determine any s-t-cut in G. Indeed, if $s = v_1, ..., v_k = t$ is the unique s-t-path in T_{GH}, then for A representing a minimum s-t-cut in G so that $s \in A$ and $u(A) = c(s, t)$, where $c(s, t)$ denotes the capacity of a minimum s-t-cut in G, then there comes an i so that $v_i \in A$ and $v_{i+1} \notin A$. Then A represents a v_i-v_{i+1}-cut in G and hence $c(s, t) \geqslant c(v_i, v_{i+1})$. In particular, we have $c(s, t) \geqslant \min_i(c(v_i, v_{i+1}))$. Now if we consider the bipartition given by the components of $T_{GH}\backslash(v_i, v_{i+1})$, with one partition set, say B, contains s, then by the property of the Gomory-Hu tree, this B also represents a v_i-v_{i+1}-cut in G, and since it represents an s-t-cut too, we have $c(v_i, v_{i+1}) = u(B) \geqslant c(s, t)$. So we actually have $c(s, t) = \min_i(c(v_i, v_{i+1}))$, and finding the minimum s-t-cut on a Gomory-Hu tree provides a minimum s-t-cut in G.

We will construct the Gomory-Hu tree by forming a sequence of trees, each of which has vertices representing sets of vertices of the original graph. In the k-th iteration, we'll start with vertices $V_1, ..., V_k$, which for a partition of the vertices V of the graph G. Each such set-vertex will have a representative $r_i \in V_i$, a special vertex. We start with $V_1 = V$ in the first iteration, choosing any ertex as representative r_1. At each iteration, we choose a set V_i of size $\geqslant 2$, and a vertex $t \neq r_i$ in it. We then compute a minimum r_i-t-cut in G, represented by X where $r_i \in X$. We then split V_i into two parts, which will be among the next set-vertices $V_1, ..., V_{k+1}$ of the next iteration. We split it into $V_i \cap X$ and $V_i\backslash X$. The representative of $V_i \cap X$ remains r_i, while we set t to be the representative of $V_i\backslash X$. We now handle the edges. The edges of the tree at the start of the k-th iteration can be represented by (r_i, r_j). To get the next tree, we start by keeping the edges that weren't incident to r_i. Next, we add the edge (r_i, t) with capacity the value of the minimum r_i-t-cut in G we computed. For the neighbouring r_j of r_i in the starting tree, we check if $r_j \in X$ in G, in which case we keep (r_i, r_j) in the next tree, or $r_j \notin X$ in G, in which case we take (t, r_j) instead of (r_i, r_j) in the next tree, and with the same capacity. Once all $V_1, ..., V_n$ are singletons, we're done.

This does indeed construct trees, as we subdivide vertices at each step, so that hypothetical cycles in the next graph would also be in the previous one, which is impossible as it was a tree. We'll now show that at each step, we have the Gomory-Hu property: the the union of the set-vertices, of a partition set of biparitition obtained by deleting edge $\{r_i, r_j\}$ in the k-th tree, represents a minimum r_i-r_j-cut in the graph G.
We have it after the first iteration, in which there is just a single edge which is by definition such that the property holds. To see that it's maintained by an iteration, we analyse the change in edges at each iteration. For the vertices r_j non-incident to r_i, in the bipartition of set-vertices obtained by deleting $\{r_i, r_j\}$, we note that $V_i \cap X$ and $V_i\backslash X$ are in the same parition set, as they should so that the corresponding r_i-r_j-cut in the graph G stays the same.
For the r_j that were previously incident to r_i, we now have edges (r_i, r_j) or (t, r_j). Since r_i and t are connected by (r_i, t) in the new tree, and all edges previously connected to r_i are now connected to either r_i or t, when we delet one of the (r_i, r_j) or (t, r_j), then r_i and t are still in the same component (the paths to r_i may now go over t in that component, and those to t over r_i). So we are in the same situation as for the edges that weren't incident to r_i: the cut corresponding to the the bipartition of set-vertices obtained by deleting the edge is the same, hence it's still minimal by the property for the previous iteration.
For the new edge (r_i, t), its capacity is by definition the value of a minimum r_i-t-cut in G as desired by

the property. It remains to see the the bipartion obtained by its deletion in the tree also corresponds to minimum a r_i-t-cut in G. We can see that in the new tree, with our costruction of egdes previously incident to r_i, the r_j that are in X are in r_i's component of the cut obtained by deleting (r_i, t), while the r_j not in X are on t's side. We thus have two cuts in G to compare, the one represented by X and the one corresponding to the components obtained by deleting (r_i, t), which we'll represent in G by P_T. All r_j that were in X are connected now to r_i through (r_i, r_j) in the new tree, so that by deleting (r_i, t) in it, r_j is in r_i's component. Similarly, all r_j that weren't in X are connected now to t through (t, r_j) in the new tree, so that by deleting (r_i, t) in it, r_j is in t's component.



Triangular property:

If $c(s,t)$ denotes the capacity of a minimum s-t-cut, then for any three $r, s, t \in V$, we have $c(s,t) \geqslant \min(c(s,r), c(r,t))$.

Proof: We let A represent a minimum s-t-cut so that $s \in A$ and $u(A) = c(s,t)$. We disjoin the cases of $r \in A$ and $r \notin A$. If $r \in A$, then A represents an r-t-cut, so that $u(A) \geqslant c(r,t)$, and if $r \notin A$, then A represents an s-r-cut, so that $u(A) \geqslant c(r,s)$. In either case $\min(c(r,s), c(r,t)) \leqslant u(A) = c(s,t)$.

How to properly exploit a monopoly:
Suppose you manage all cafés of your city and your goal is to squeeze as much money out of your fellow citizen. You're given a set $[b]$ of buyers j, for each of which you know the prohibitive price m_j for coffee (the most they're willing to pay for a cup of coffee) as well as the accessibility to the next café in the form of $S_j \subseteq [c]$ (the set of cafés consumer j is willing/able to go to), and a set $[c]$ of cafés i, that you own, each of which can sell a_i cups of coffee. Our goal is to determine x_i at which to sell coffee in café i so as to obtain highest income. We assume that for prices x, buyer j will get their coffee at the cheapest café in S_j.

If we sold coffee at price x in every café, we could model allocation as follows. We create a bipartite digraph with vertices representing buyers and cafés, and add edge (i,j) if buyer j has access to café i. We turn this into a flow-network by adding a source s that we connect to cafés i with and edge with capacity $x.a_i$, and a sink t which we connect buyer j to with an edge of capacity m_j. The edges between buyers and cafés have infinite capacity (in practice, capacity $1 + \max\left(\sum_i xa_i, \sum_j m_j\right)$ will do the job). An s-t-flow

will represent a possible allocation of coffee to buyers, as for a uniform price, the buyers are indifferent between where they get their coffee from. This representation requires buyers to buy a fraction of cup of coffee.
In this context, a minimum cut may not use edges between buyers and cafés (as cut $(s, V\backslash s)$ will be lower), so both endpoints must be in the same partition set,and more generally, the neighbourhoods of buyers and cafés must be in the same partition set.
The partitions therefore have form $(s \cup S \cup \delta(S), t \cup [c]\backslash S \cup \delta([c]\backslash S))$ where $S \subseteq [c]$ and so that $\delta(S)$ and $\delta([c]\backslash S)$ are disjoint (which thy have to be, for otherwise a buyer would have access to two cafés in different partition sets, so that at least one edges between buyer and café is used in the cut, which we prohibited). The cut has value $\sum_{i\in[c]\backslash S} xa_i + \sum_{j\in\delta(S)} m_j$.

In the corresponding maximum flow, the edges of the cut are used in full capacity: for the buyers, this means that the prohibitive price is reached, and for the cafés, this means that the local coffee supply is completely sold.
TO CONTINUE: example from Tardos and Vazirani (AGT). Might also be complete nonsense as allocation unclear.

Multi-commodity flows
In the multi-commodity flow problem, we ship K commodities k form sources s_k to sinks t_k (the source of one commodity may for example also be the source or sink of another). That is, we have K s_k-t_k-flows f_k for each commodity and we additionally require that $\sum_{k\in[K]} f_k(e) \leqslant u(e)$, so that the total flow of all commodities on an edge doesn't exceed its capacity.

Note that if non-zero s_k-t_k-flows exist for all k, then a non-zero multicomodity flow exists. This can be seen by scaling the s_k-t_k-flows f_k so that $\max_e(f_k(e)) \leqslant \frac{\min(u(e))}{K}$, as in that case $\sum_{k\in[K]} f_k(e) \leqslant u(e)$ will always hold. In fact, we can scale the them down further for them to all have equal value.

We now present the **Awerbuch-Leighton** algorithm, which gets a flow with high lower bound on the values of the f_k. The problem the algorithm approximates is that of finding a multi-commodity flow for which $\min(val(f_k))$ is maximum.

2.5 Connectivity

In a directed graph, we can ask how many edge-disjoint or vertex-disjoint s-t-paths there are, or more precisely for a collection of such paths of largest number. A related question is how many edges or vertices need to be deleted from a graph so that one can't travel from s to t, or more precisely we can ask for the minimum numer of edges to remove to separate s and t.
The relation is that for p edge-disjoint paths, a s-t-separating set must contain at least one edge on each path, so that the maximum number of edge-disjoint s-t-paths is less then the minimum number of edges in an s-t-separating set. A similar result holds for the vertex version. Theses numbers are actually equal and we can find paths and separators algorithmically.

Menger's theorem (digraph-edge-version):

The maximum number of edge-disjoint s-t-paths is less then the minimum number of edges in an s-t-separating sets. We can find such paths and separators in polynomial time.

Proof: We add capacities 1 to all arcs of the digraph and compute a maximum flow f for it. Then, we compute a flow decomposition to find the collection of s-t-paths. They are edge disjoint, as f is integral, in fact 0-1, so that $f(e) = \sum_{w \in P \cup C : e \in w} f_w$ implies that only one of the terms in the latter sum can be 1, preventing two paths from using the same edge. We now need to show that this collection of paths is maximum: if it wasn't then we could push unit flow through a bigger collection, resulting is bigger flow value, contradicting the maximality of f.
Finally, we turn to separators. Since we have unit capacity, an s-t-cut is the size of the corresponding s-t-separator. By max-flow-min-cut, the miniumum cut has value f. Since f is maximum, there are no t-s-paths in the flow decomposition, as we could decrease flow along these paths to get larger flow values. So in this case f is $\sum_{p \in P_{s,t}} f_p$ which is $|P_{s,t}|$. So also the minimum size of the corresponding s-t-separator is $|P_{s,t}|$. To find a separator explicitly, one can choose one edge per path of $P_{s,t}$.

2.6 Orientation

Given a graph G, we seek an algorithm that finds a strong orientation of G, which is an orientation of edges $\{u, v\}$ into (u, v) or (v, u), such that one can walk from any vertex to another along a dipath, when such an orientation exists.
Robbins theorem from graph theory tells us that G has a strong orientation $\Leftrightarrow G$ is 2-edge-connected. So we seek an algorithm that actually finds this orientation, assuming that G is 2-edge-connected.

DFS:
We'll exploit a property of a graph search algorithm known as DFS (depth-first search).
DFS starts at an arbitrary root r and maintains a queue of vertices $v_1, ..., v_q$. It then repeatedly considers the neighbourhood of v_1 and creates a new queue $v'_1, ..., v'_{q+1} = v_1, ..., v_q, u$ if it finds a neighbour not in the queue $u \in \delta(v_q) \backslash \{v_i : i = 1, ..., q\}$. One can think of DFS as building a directed tree by adding arc (v_q, u) at each such step. The path from root r to u will be $r = v_1, ..., v_q, u$. If no such neighbour exists, we label v_q as explored and proceed with queue $v_1, ..., v_{q-1}$. Once all vertices are explored, we're done.
To check that the edges "added" in DFS don't close cycles, we prove that it's a loop invariant. The in the step, no cycle is closed since u is not in the queue, nor explored. Its not in the queue since by definition $u \in \delta(v_q) \backslash \{v_i : i = 1, ..., q\}$, and it's not explored, for if it were, at the previous iteration in which u left the queue and was labeled as explored, we'd get a contradiction since the edge $\{u, v_q\}$ would have been considered, and it would have required that v_q had been added to the previous queue, but then v_q would have to tbe labeled explored before u. This is because, the queue has maximum size $|V|$, so that explored vertices will eventually appear, and the last vertices of the queue are labeled explored before the previous ones.

In particular we note that for the DFS-tree, if r is the root, v is a vertex, and P is the unique path from r to v, then for the set $D(v)$ of all vertices that are further in the tree then v in the sense that their path to r contains v, then all edges to from $u \in D(v)$ not in the tree lead to vertices of P. Otherwise, we'd contradict the order in which DFS visits vertices, by finding edges that wouldn't have been considered during DFS.

Orientation:
This property is what leads us to find a strong orientation of G if there is one. We construct a directed DFS spanning tree from an arbitrary root r and for all edges not in the tree, since they connect vertices on ta directed path (the one from r), there is a unique orientation with which to close this path to a directed cycle. By DFS, any vertex can be reached *from* r. Now, if there was a vertex u such that u isn't connected *to* r, then by choosing it closest to r on its path from r (wlog), and deleting the last edge on that path from r, we would split the tree in two components. By 2-conncetedness, there would be an edge (x, y) from u's component leading to r's component, that isn't in the tree. We now know that y is on the path from r to u. So by taking the path from u to x, then (x, y) and then the path from y to r, which exists by minimality of u on the r-u-path for that property, we get a path from u to r, contradicting the assumption on u.

Finally, we can extend this procedure to one that tells us if a strong orientation even exists. Note that 2-connectedness, which we now know to be sufficient is also necessary: if a stronge orientation exists, then deleting an edge $\{u, v\}$ can't split the graph into two components, for otherwise there wouldn't be a path from u to v if $\{u, v\}$ was oriented as (v, u) and mutatis mutandis. So once our DFS and orientation terminates, we can check if deleting any edge from the DFS tree disconnects the graph to check for 2-connectedness. If one such edge is found, then the graph can't have a strong orientation.

2.7 Coloring

Greedy vertex coloring, Vizing as in Williamson Shmoys, Hypergraph coloring and algos as in Voloshin

2.8 Algorithms on strings

RNA secondary structure:
The macromolecule RNA is a string of smaller base molecules denoted A, C, G,U, that folds on itself by pairing up bases A and U as well as C and G, so as to minimise a potential energy.
In mathematical terms, the RNA is a sequence $c_1, ..., c_n$ where $c_i \in \{A, C, G, U\}$, and we seek a matching, given by pairs $\{i, j\}$, with certain properties. First, if $\{i, j\}$ is in the matching, it must be that $(c_i, c_j) = (A, U)$ or $(c_i, c_j) = (C, G)$. Minimizing energy will be interpreted as maximising the size of the matching, so we seek a maximum matching. We have additional geometric/topologic constraints. RNA shouldn't be "knotted", which we'll interpret that for $i < j < k < l$, we may have only one of the pairs $\{i, k\}$ and $\{j, l\}$ in the matching (there is no crossing). Also, RNA can't bend to much, so that $i < j$ may be paired up only if $j - i \geqslant 5$.

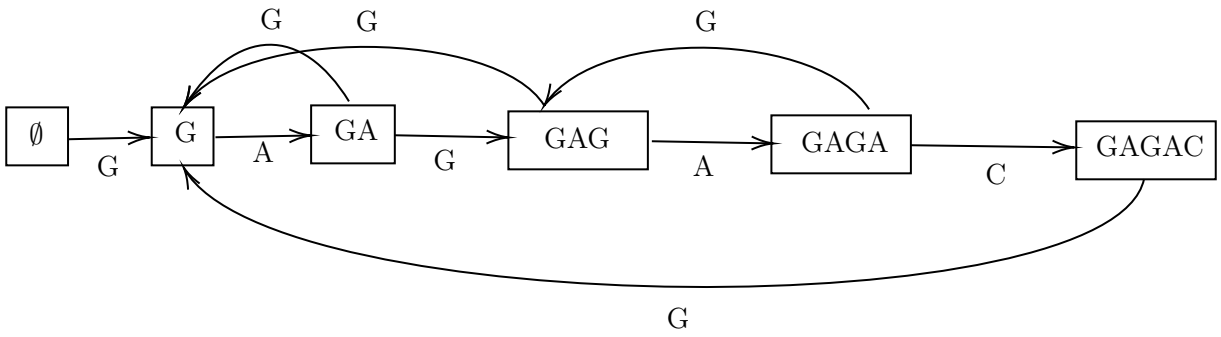
We attempt a recursive approach, by disjoining on the last c_n being paired up or not in a optimal secondary structure. In the latter case, the problem is that on $c_1, ..., c_{n-1}$, and in the former things get more interesting. If n is paired up with t, for $t \leqslant n-5$, then only pairs in the intervals $[1, t-1]$ and $[t+1, n-1]$ are allowed, as pairs crossing intervals would violate the "knotting" constraint with the pair $\{t, n\}$.
If we find the optimal structures for $c_1, ..., c_{t-1}$ and $c_{t+1}, ..., c_{n-1}$, then these are the sub-structures on the global optimum one. This argument shows that the maximum size of a matching $v(i, j)$ for the problem on $c_i, ..., c_j$ is the maximum of $v(i, j-1)$ (j not paired up), and $1 + v(i, t-1) + v(t+1, j-1)$ for all $i \leqslant t \leqslant j - 5$ such that $\{c_t, c_j\} = \{A, U\}$ or $\{c_t, c_j\} = \{C, G\}$.
We see that the recursive calls are on intervals of smaller length. By intialising with $v(i, j) = 0$ for all $4 \geqslant j - i$, and computing the values $v(i, j)$ for $j - i = k$ for $k = 5, ..., n-1$, we can find $v(1, n)$, which is the value of the total string that we seek.
This algorithm runs in time $O\left(n^3\right)$ since we have to comple n^2 values $v(i, j)$, each time by checking pair compatibility and finding a maximum in $O(n)$ time.

String matching with automata:
In the ananlysis of DNA, which can be represented as a sequence of bases A, C, G, T, we often want to solve the problem of finding all places (if any) where a gene might be located. A gene is a smaller sequence of bases that is included in DNA.
The brute force approach is to check at all n points of the DNA if the next m bases are that of the gene, for a gene of size m and a DNA of size n. This is takes $O(mn)$ time, but we can do better.

We generalize the problem to an alphabet of α characters, here $\alpha = 4$. The point is that the substring (gene) may have a certain self similarity, which makes the brute force approach inefficient. If the substring is $GAGA$, then the brute force approach on $GAGAGA$ will start one check on the second A, despite the-self similarity indicating that we could have started comparisons after it.

We will construct a thing called an automaton that will locate the appearances of the substring in the main string upon reading the main string. At each step, the automaton will indicate how close we are to having a complete copy of the substring and use self-similarity of the substring in a clever way.
We represent the automaton by a digraph with $m + 1$ vertices. Each vertex will represent the number of matching characters in the first part of the substring that we have found aligned in the main string. If at a current state, we're at a given vertex (the state of the automaton), we look at the next character in the main string. For each character in the alphabet, and each vertex, there will be one edge per character, pointing from that vertex to another one, that we will follow according the the next character in the main string we just considered, to go to the vertex/state of the next iteration. We'll explain how to build the automaton soon, but for now, we claim that the automaton handles self-similarity of the string by having edges point to the largest prefix of the substring. This is best explained by an example: in the figure

below, we've represented the automaton for the substring $GAGAC$. We have ommitted the edges leading to the empty state.



The efficiency can be seen in the case of a mian string $GAGAGAC$: arrived at state $GAGA$, we read G and instaid of moving to the empty state, we move to GAG. We then reach $GAGAC$ in the next two iterations.

Whenever the final vertex corresponding to the full substring is reached, we know that the substring has appeared, in the m last strings of the main string considered so far. Conversely, if the substring appears, then we can show that the automaton will detect it. We know that if the automaton is on vertex v, then the last strings of the main string considered so far are the string associated to that vertex (at same length, when considering the last characters of the main string). We can then argue by induction on the size of the main string. If the algorithm works for strings of size n, then for strings of size $n+1$, we've located the appearances of the substring in the first n characters of the string. The last state transition of the automaton will indicate us if the substring is also a suffix of the main string. So this treats the only possible appearance that wasn't in the first n characters, and concludes the induction.

We now discuss how to construct the automaton. We do this vertex by vertex, starting from the one representing the empty string to that representing the full substring. At each new vertex added, representing s and for each character c in the alphabet, we add an edge from that vertex to either the next vertex to be added if sc is a prefix of the substring, or to the previous vertex of greatest size, corresponding to z, such that z is a suffix of sc. This last step requires at most m comparisons, each being a comparison of at most m characters. Since we do this for each new vertex and each character in the alphabet, constructing the automaton requires $O\left(m^3\alpha\right)$ comparisons. After that, running the main algorithm consists in n elementary steps, so that the total cost is $O\left(m^3\alpha + n\right)$. For small α and m, compared to n, which is the case for the DNA example, this beats the brute force approach.

Knuth-Morris-Pratt algorithm:
Intro algo by Rivest et al, chap 32

Sequence alignment:
Algo design by Tardos, chap 6.6 and 6.7

2.9 Phylogenetic trees

In the perfect phylogeny problem, we're given n unordered sequences (sets) of characters called taxa (singular taxon), such as the DNA of n species, but in the representation of a set of genes. We can identify these taxa by ordering the m characters present over all taxa, and associating to each taxon t a row-vector m_t such that $(m_t)_i$ is 1 if character i is in taxon t and 0 otherwise.
We'd like to guess the evolutionary history of these taxa, assuming that they're related. We make the following model: taxa may have common ancestors, that posses the characters that their descendants have in common. At each generation, the ancestors gain a character, but never lose any, until we arrive at a taxon that is among our n real-life observed ones. We can therefore represent the history by a rooted directed tree, called a phylogenetic tree:

Phylogenetic tree:

A **(perfect) phylogenetic tree** is a rooted directed tree, where each vertex represents a taxon, and each edge represents a character. It must have n leaves representing the taxa from the problem input. Each character may be represented by a single edge only. The root represents the empty taxon. If, for any taxon t represented by a vertex on the tree, an edge of the unique path from the root to t is labeled with character c, then taxon t must posses character t.

The perfect phylogeny problem asks if a set of n taxa has a perfect phylogeny, and if so how many, and for an algorithm to find one when one exists.

Perfect phylogeny theorem:

A phylogenetic tree exists for n taxa if for any pair of the m taxa, one can not find 3 taxa who's indicated values for the characters are $(0,1)$, $(1,0)$ and $(1,1)$. If this is the case, we can find a phylogenetic tree in polynomial time.

Proof: To see that phylogenetic trees have this property, so that it's a necessary condition, note that the edges corresponding to any pair of characters c and d (we call edges by their character as the character appears only once on a phylogenetic tree) can appear in three ways:

- The unique path to the tail of c contains d: then, all descendants taxa of the head of c must contain both characters c and d, so that if in the order of characters $c < d$, then $(0,1)$ and $(1,1)$ may appear, but not $(1,0)$.
- The unique path to the tail of d contains c: then, all descendants taxa of the head of d must contain both characters c and d, so that if in the order of characters $c < d$, then $(1,0)$ and $(1,1)$ may appear, but not $(0,1)$.
- The paths leading to the tails don't contain the other character. In that case $(1,1)$ may not appear, otherwise a corresponding taxon could be reached by two different paths.

To see that these conditions are actually sufficient is the core of the theorem.
The idea is that characters that appear in many taxa must have appeared early in the evolutionary history, as in our model characters can't vanish once acquired. So we'll sort the characters in decreasing number of appearances in the taxa, in $O(mn)$ time. We then note that for a phylogenetic tree, the labels must increase along paths: if c is on the path to the tail of d, then all the leaves that are descendants of d are also descendants of c, hence also posses charcater c, so that c has more or equal number of appearances

then d.
Before building trees, we note that if two characters have the exact same appearances in the taxa, we can combine them to one character, find a perfect phylogeny for these modified taxa, and subdivide the edge corresponding to that character into two corresponding to the two original ones. So we may assume that no two characters have the same appearances.
We'll try to build the trees inductively. The tree for one taxon is just a path. Note however that we'll order the path according to the ordering of characters given by the number of appearance per character for all n taxa. If we have a perfect phylogenetic tree T for k taxa, how can we get one for an additional taxon (assuming the properties of the theorem) ? If t is the new taxon, we consider its characters that have already appeared in the tree. They must in fact for a path from the root to a taxon t' or we can conclude that the necessary condition is violated, and hence note phylogenetic tree may exist, as we'll soon show. With this property, we can build the next tree by adding edges corresponding to the characters previously not in the tree in the form of a path starting from t', in the order of the characters for the n taxa. This is indeed a phylogenetic tree, as it's a tree (we added a path), the path leading to t contains precisely the characters of t, and all characters appear once, as the new edges correspond to characters previously not in the tree.

We now prove the claim about paths made along the way. We first note that if a pair of the characters from t is on different paths from the root (so that there are are descendants with indicators $(1,0)$ and $(0,1)$), then there can't be a be a phylogenetic tree with t (which indicates $(1,1)$). We then stop the algorithm and mention infeasibility. If not, all pairs must be aligned on a same path P. We can then show that the characters on that path are precisely those that t has in common with T by a property called the shared prefix property.
To show the prefix property, we'll show that all characters of the path a also in taxa t. We'll consider the character l that is the last wrt. the ordering of characters that is on taxon t an on all other taxa of the tree T. We denote by t' the head of l. Since t' was already in T, we can follow it in T until we reach a leaf t''. Now, for any character c in P different from l, the indicator of c and l for t'' will be in configuration $(1,1)$. Since c appeared earlier on P then l, its number of appearances must be $\geqslant$to that of l. The tree T must contain another leaf-taxa which indicates $(1,0)$, for otherwise there are only indicators of form $(1,1)$, $(0,0)$, or $(0,1)$ among the k taxa, so that the number of appearances of l is greater or equal to that of c, equality appearing only if only indicators of form $(1,1)$ or $(0,0)$ are present, in which case the characters have the same appearances, a case we excluded in preprocessing, so that l would have more appearance then c, a contradiction to l following c on P. But now that we know that there is another taxa t''' indicating $(1,0)$, the new taxa t can't indicate $(0,1)$, aka posses l but not c as t, t'' and t'' would be a triple violating the necessary condition. So by checking the necessary condition in $O\left(nm^2\right)$ in a preprocessing step, we may assume that it holds, so that t can't indicate $(0,1)$, and since it has l, it must indicate $(1,1)$, aka posses c. Hence, all the characters of P are in t, as we desired to prove all along.

2.10 Enumeration algorithms and combinatorial generation

Spanning forests, and spanning trees via Prüfer codes:
Our goal is to find all the spanning forests of a graph G. We will do this recursively on the number of vertices. We consider the recursive procedure $A(G)$ which returns a list of all spanning forests of G. As base case, if the graph is a single vertex, we return a list with that vertex as a forest. $A(G)$ works by selecting a vertex $v \in V$, running $A(G\backslash v)$, and post-processing. From a spanning forest of the list from $A(G\backslash v)$, we can gain multiple spanning forests of G. We check the neighbourhood of v in G on which component they're in. To get a forest, the only condition is that no two edges leading v to the same component are added: in all other cases, no cycles are closed and the graph remains a forest. So from each forest of $A(G\backslash v)$, we obtain forests on selecting one or none of the edges leading to the same component. Conversely, any spanning forest of G becomes a spanning forest of $G\backslash v$ if we delete v from it.
This algorithm runs in time $O(|V|)$, but has exponential space complexity, as at most $2^{\deg(v)}$ forests can be added to the list of $A(G)$ for each forest of $A(G\backslash v)$.

In fact, we'll show that the problem of enumerating all trees is exponential by nature, as a complete graph has $|V|^{|V|-2}$ trees (possibly isomorphic). There is also a linear algebra proof of this, using the matrix-tree-theorem.
We will show this with Prüfer codes: each tree is in correspondence to a sequence of length $|V|-2$ with alphabet V. The idea is to construct the tree leaf by leaf. If we delete a leaf from a tree, we get another tree, and we can reconstruct the initial tree if we know which vertex was the trees neighbour. When deleting leaves one ofer the other, it's good to have an order in which to delete the leaves, so as to know in which order to glue them back. If we label the nodes from 1 to $n = |V|$, then we construct the Prüfer code of the tree by repeatedly deleting the lowest labeled leaf and noting its neighbour $p_i \in V$, until we arrive at a graph with a single edge, both endpoints of which are leaves, so that our Prüfer code is $p_1, ..., p_{n-2}$.
To reconstruct a tree from any Prüfer code, we repeatedly add edge (p_i, l_i) where l_i is the lowest labeled vertex that hasn't yet been added and which isn't i the Prüfer code, starting from index i. Once we've added (p_{n-2}, l_{n-2}), only two vertices haven't appeared as leaves, and we connect them by an edge.
To show that this construction always produces a tree, we use induction. For $n = 3$, there is just one pairing (p_1, l_1) and two vertices which haven't appeared as leaves, so that we get a graph with 2 edges, which is always a tree for $n = 3$. For the induction step, we note that after the addition of edge (p_1, l_1), we can relabel the graph by decreasing the labels larger then l_i by 1, so that the future steps are exactly the steps of the algorithm for the case $n-1$. By induction, we know that they produce a tree, and since we add l_1 as a leaf to that tree, we still get a tree.

Minimal dominating sets:

Minimal multicuts:

Cycle space of a graph:
When the numer of sub-structures in a structure is very large, then we seek a way of representing these sub-structures that requires little storage and simultaneously allows for efficient recovery of all the sub-structures. We give an example of this for cycles in a graph, based on topology.

Note that any circuit and any walk in the graph can be represented as an expression $\sum_{e \in E} c_e e$ where $c_e \in \mathbb{Z}$ is the number of times we crossed the edge on that walk. These expressions can be though of as a module. If we give the graph some orientation, then we can consider the so-called boundary map ∂ that

we define to be linear over the module of edges, mapping edge (u, v) to the expression $v - u$ on the module of vertices over $\mathbb{Z}$. The kernel of ∂ contains the (expressions of) circuits of the graph.
We seek a basis of this kernel. For a spanning tree T, each edge not in the tree closes a unique cycle C in that tree: we'll show that the elements $\sum_{e \in C} e$ form a basis of the kernel. A special case is that when the graph is a tree: we'll also need this case in the proof for general graphs. So for a tree, we have to show that the kernel is trivial (contains only 0). To see this, we use induction on the size of the tree. At each step we consider an expression $\sum_{e \in E} c_e e$ such that $\partial \left(\sum_{e \in E} c_e e \right) = 0$ and a leaf u of the tree, connected to it by $\{u, w\}$. By writing $\partial \left(\sum_{e \in E} c_e e \right) = \sum_{v \in V} k_v v$ for some $k_v \in \mathbb{Z}$ by linearity and summation, we note that $k_u = c_{\{u,w\}}$, as this is the only edge where u appears. Then $\sum_{v \in V} k_v v = 0$ implies that $c_{\{u,w\}} = k_u = 0$ as the $v \in V$ form a base of that module, and $\sum_{e \in E} c_e e$ is actually an expression in the modul of the smaller tree, the one where leaf u was deleted. So by induction, we get that the rest of the $c_e = 0$.

Now we turn to the general case. We denote by $e_1, ..., e_r$ the edges not in the spanning tree T and by z_i the expressions of the unique cycles C_i that the e_i close. IIf we are to write $\sum_{e \in E} c_e e = \sum_{i \in [r]} k_i z_i$ for a kernel element ($\partial \left(\sum_{e \in E} c_e e \right) = 0$), since we cross the edges e_i only in C_i, we should have $k_i = c_{e_i}$, as $e \in E$ form a basis of the edge module. By linearity of ∂, $\sum_{e \in E} c_e e - \sum_{i \in [r]} c_{e_i} z_i$ is in its kernel. Yet, all the coefficients for the e_i of this experssion are 0, so that the possibly non-zeor ones are on the edges of the tree T. As we've just previously established, this implies that the expression is 0, so we really do have $\sum_{e \in E} c_e e = \sum_{i \in [r]} c_{e_i} z_i$ for any $\sum_{e \in E} c_e e \in \ker(\partial)$. These coefficients of $\sum_{i \in [r]} c_{e_i} z_i$ are unique, as the edges e_i appear precisely only in the z_i respectively, so the z_i do form a base in $\ker(\partial)$ as a module.

To get enumerate the cycles of the graph, we can now do the following. Since a cycle corresponds to an expression $\sum_{e \in C} e$, which we now know can be written as some $\sum_{i \in [r]} k_i z_i$, where $k_i = 1$ if $e_i \in C$ and 0 otherwise, we can enumerate cycles by considering all 2^r expressions $\sum_{i \in [r]} k_i z_i$ where $k_i \in \{0, 1\}$.

TO FIX: replace $\mathbb{Z}$ by $\mathbb{F}_2$ so that in the last part the cycle comes from expressing $\sum_{i \in [r]} k_i z_i$ in the E and considering connected components and Euler-tour decompositions.

2.11 Sensitivity analysis and post-optimization

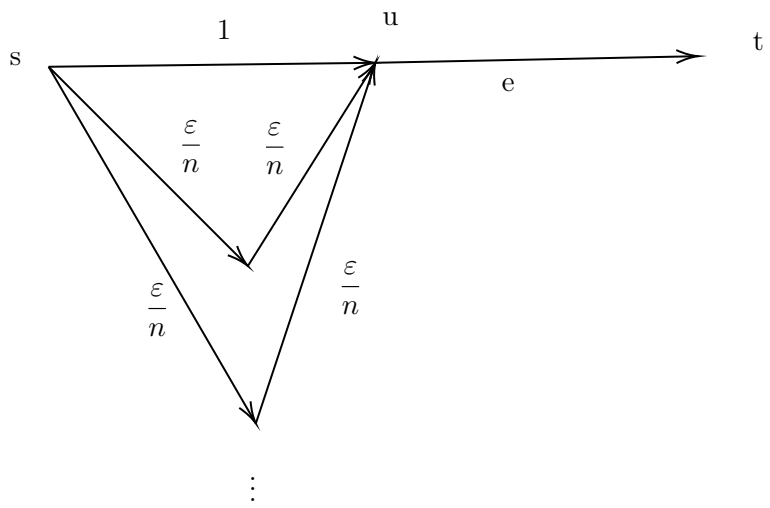
In this section, we give few examples of sensitivity analysis and post optimization processes. Sensitivity analysis asks how a small perturbation in the input of the problem can lead to a change in its solution. In a post optimization process, we try to find efficient ways of adjusting a current already computed solution for it to solve the problem for a slightly modified input instance. Since there are many ways to perturb an input, we only study particular types of perturbations in this chapter, but remark that other perturbations exist.

A sensitivity analysis of MST:
We want to know what happens to the optimal value of a MST if we perturb the edge-weights w of a graph G to w', where $|w(e) - w'(e)| \leqslant \varepsilon$ for all $e \in E$. If T is an MST for w and T is an MST for w', then we have the following. On the one hand, $w'(T') \leqslant w'(T) \leqslant w(T) + \sum_{e \in T} |w(e) - w'(e)| \leqslant w(T) + (|V| - 1)\varepsilon$ and on the other $w(T) \leqslant w(T') \leqslant w'(T') + (|V| - 1)\varepsilon$ with a similar argument, since trees have the same number of edges. In all cases of signs, we have $|w(T) - w'(T')| \leqslant (|V| - 1)\varepsilon$. This means that the optimal value is $(|V| - 1)$-Lipschitz in the perturbation, so in particular small perturbations shouldn't make a big difference on the optimal value (but the actual MST may differ on multiple edges).

A sensitivity analysis of BPP:
The situation is different for the bin packing problem. It asks, given a list of of weighted items $(w_i)_{i \in [n]}$ where $w_i \in [0, 1]$, for the smallest number of bins b such that we can partition $[n]$ into b sets B_j such that $\sum_{i \in B_j} w_i \leqslant 1$ for all bins (the bins have volume 1 and we can't let them overflow). For this problem, we can find a family of instances such that even for arbitrarily small perturbations, the smallest number of bins used for one perturbation is increasingly larger then that of the original instance. Indeed, consider the cases $(w_i)_{i \in [n]}$ (the family of instances is indexed by n) where $w_i = \frac{1}{2}$, where the optimal number of bins is $\left\lceil \frac{n}{2} \right\rceil$ (pack the items two-by-two; optimal since $\frac{n}{2} = \sum_{i \in [n]} w_i = \sum_{j \in [b]} \sum_{i \in B_j} w_i \leqslant b$ for any packing). For all perturbations $\varepsilon > 0$, we can find an instance w' such that $|w_i - w'_i| \leqslant \varepsilon$ for all i, which is $w'_i = \frac{1}{2} + \varepsilon$, such that the optimal value doubles! This is because we can only pack one item per bin, so that in particular the least number of bins is n. So the distance in optimal values is $\left\lfloor \frac{n}{2} \right\rfloor$, which grows with n and becomes arbitrarily large.

A post-optimization for network flows:
Assume we've computed an maximum flow in a network, but that one capacity $u(e)$ is perturbed to $u'(e)$: is there an efficient way of finding a maximum flow for the network with one perturbed capacity ?
If e was not in a minimum cut and $u'(e) > u(e)$, then this cut is still minimum (as the value of a cut can only have increased), and the flow for the unperturbed network is still feasible, and maximum, as it's value is still that of the minimum cut.
If e was in a minimum cut and $u'(e) < u(e)$, then this cut is still minimum (otherwise, the initial one wouldn't have been). We can then find a flow which has the value of this new minimum cut, and which will therefore be optimal. We look for an s-t-path containing e by DFS on the graph induced by flow-carrying edges (here, it's helpful to consider sources and sinks the are such also in the digraph sense, for it prevents cycles in the flow). The edge e has to be the bottleneck of this path, for otherwise, the maximum flow was less then the minimum cut. We can thus decrease the flow along this path from $u(e)$ to $u'(e)$, which will have the value of the minimum cut, and hence be optimal.
In other cases of perturbations, things aren't that easy. For an increase in capacity of an edge in a minimum cut, we may need to keep using augmenting paths, and possibly more then one of them. For example,

consider the following network:



There are n pairs of edges with capacity $\frac{\varepsilon}{n}$. For the network with $u(e) = 1$, we have an optimal flow in the 1-flow s-u-t. Note that e is in a minimum cut. If we increase capacity to $u'(e) = 1 + \varepsilon$, then the optimal flow is the one of full capacity. To get it, we have to augment along n paths, by flow $\frac{\varepsilon}{n}$ for each of them.

2.12 Solutions

Ex.kSP:
The idea is that of finding a way to exclude certain paths precisely, so that we can solve the problem by using the previous shortest path algorithms as routine, excluding the previous shortest paths at each iteration.
Consider the situation in which we which P is a shotest s-t-path of G, and we which to find a shortest s-t-path among those different then P. For such a path Q, there must and edge of P not used by Q: else, since Q has no cycles, for an edge enumeration of P, Q has to use e_1 first, as it would close a cycle from s when reaching it otherwise, then e_2 second, as it would close a cycle from the tail of e_1 to e_2, and so on, so that $Q = P$.
We can solve the shortest path problem on the graphs $G\backslash e$ for the edges e of P, one of which contains Q, and consider the shortest among these outputs, which will be shorter then Q, hence of weight equal to Q, by definition of Q. No s-t-path other then P will be excluded from the feasible solutions to the optimization problem, as its contained in at least one of the graphs, as it can't contain all edges of P. So this procedure yields a desired path.
To get the successive shortest paths, we will use a branching procedure, which we represent by a branching tree. The nodes of the tree are shortest path subproblems on subgraphs. The children of a graph G' with shortest path P' will be graphs $G'\backslash e$ for the edges e of P', together with their shortest paths. We'll show that a least weight path among the paths of the problems of the leaves (childless nodes) of the branching tree is a shortest path of G among the paths of G except for those present as solutions of non-leaf (the ancestors) nodes of the branching tree. So in order to find the successive k shortest paths, we have to branch $k-1$ times. A branching iteration requires computing a minimum over the leaves of the tree and then computing the children of a node achieving he minimum. Since the path P' has at most $|V|-1$ edges, the node corresponding to (G', P') can have at most $|V|-1$ children, so that branching adds a netto of at most $|V|-2$ leaves per iteration. So the minimum to be computed over the leaves can be done in $O(k|V|)$ time, as we branch at most k times. When determining the children, we run a shortest path algorithm on a subgraph of G at most $|V|-1$ times, so that this step can be done in $O\left(|V|^4\right)$ (with Floyd-Warshall, and better with Dijkstra). Since we branch at most k times, the total runtime of the procedure is $O\left(k|V|^4 + k^2|V|\right)$.

Now we show the property we mentioned by induction on the number of excluded paths. For 0 excluded paths, this become the conditions that the path be in G, which is true. A path R that is shortest among the paths of G other then those of the n nodes of the branching tree that we branched on is in particular a path among the paths of G other then the $n-1$ ones we branched on, before the nth branching. It will then be in contained in at least one graph of the leaves by induction. If it's not contained in the graph that is branch on in the nth branching, then we're good, and otherwise, it must be in one of the children, as it can't contain all of the edges of the nth path, for the same reasons we gave previously for Q, as it then would then have to be the nth path.
ERROR: It could be that the shortest path of a leaf $G\backslash e$ is also a shortest path of node $G\backslash e$ we branched on, in the case that the shortest path is contained in multiple leaves at a branching. Then our argument brakes. This is why Lawler uses a different branching on $G\backslash e_1$, $G\backslash e_2$ and the path uses e_1, $G\backslash e_3$ and the path uses e_1 and e_2, etc. This way, the paths are contained in exactly one of the children, as they can't be in multiple. Refer to paper "A Procedure for Computing the K Best Solutions to Discrete Optimization Problems and Its Application to the Shortest Path Problem". □

Ex.DMP:
There are (at least) two ways of solving this.
In the first, we adapt Dijkstra's algorithm. We denote by $l_i(v)$ the value label of vertex v in iteration i,

by S_i the set of settled vertices in iteration i and by $p_i(v)$ the predecessor of v on a optimal s-v-path at iteration i.
We preprocess the graph by adding $-\min(w(E))$ to the weights, so that wlog. $w \geqslant 0$. This won't affect optimality of the paths, as any maximum weight edges of two paths differ from the same amount, so that order is preserved.
We initialise with $S_i = \emptyset$, $l_i(v) = \begin{cases} 0 : v = s \\ +\infty : else \end{cases}$ and $p_i(v) = none$.
We loop on extracting a minimum labeled vertex u not in S_i, putting it in S_i and relaxing the labels of $v \in \delta(v) \backslash S_i$ by $l_{i+1}(v) = \min(l_i(v), \max(l_i(u), w(u,v)))$ and setting $p_{i+1}(v) = p_i(v)$ if the minimum is attained in the first argument and $p_{i+1}(v) = u$ otherwise. We use same labels and predecessors for the other vertices of $V \backslash S_i$. We stop when $S_i = V$.
We prove the loop invariant that from vertices v in S_i, the optimal s-v-paths are known and are in fact contained in S_i. This is true in the first iteration, in which $S_1 = \{s\}$ and $l_1(s) = 0$. In iteration i, we add v to S_{i+1}, where v has predecessor u on the s-v-path P. Assume for contradiction that there is a path P' from s to v, such that $\max(P') < \max(P)$. Consider the first edge $\{x, y\}$ on P' such that $x \in S_i$ and $y \notin S_i$. Since v was added, it must be the case that $l_i(v) \leqslant l_i(y)$. Now, on the one hand $l_i(v) = \max(w(P))$ as $l_i(v) = \max(l_j(u), w(u,v))$ for some previous iteration $j \leqslant i$ and $l_j(u) = \max(w(P|_{s \to u}))$ by the loop invariant. On the other hand, $l_i(y) \leqslant \max(l_{i-1}(x), w(x,y)) \leqslant \max(w(P'|_{s \to y})) \leqslant \max(w(P'))$, where the middle inequality comes from $l_{i-1}(x) \leqslant \max(w(P'|_{s \to x}))$, by the loop invariant. Butting the bounds next to each other, we get $\max(P) \leqslant \max(P')$, a case we precisely excluded. So the algorithm terminates with optimal paths.
The runtime is the same as Dijkstras (which can depend on the data-structure used), as all we did was replace an addition by a comparision (when relaxing labels, we compute the max).

A second way of solving this problem is by noticing the following. If the minimum maximum weight of an s-t-path is w, then there can be no s-t-path using edges e of weight $w(e) < w$. This can lead to checking connectivity in graphs induced by edges e of weight $w(e) < w$, where w ranges over all possible weigths $w(E)$. However, we can do even better: its turns out that an MST solves this problem for all pairs s and t.
Indeed, assume that the maximum weight edge on the s-t-path P of the MST T is w, achieved in e, and that there exists a better path P' such that all edges in P' have weight less then w, aka $w(P') < w$. Then $T \backslash e$ spolits into two components containing s or t, and there is an edge $e' \in P'$ that connects these components. Then $(T \backslash e) \cup e'$ is a tree and it has less weight then T as its weight decreased by w and increased by $w(e') < w$, contradicting T's MST nature. So P' can't exist and P is optimal. $\square$

Ex.FLST:
(i) It's in P. We check if the graph is connected and then if the induced graph $G[V \backslash T]$ is connected (dfs). In the positive case, $G[V \backslash T]$ has a spanning tree (found with dfs) which we can extend to a spanning tree of G in which all vertices of T are leaves by adding these vertices along any edge of their neighbourhood (which exists as G is connected). This forms a spanning tree as no cycles can be closed and all vertices not in T where already spanned in the spanning tree of $G[V \backslash T]$. Finally, since all vertices of T are connected to the tree by a single edge, they're all leaves.

(ii) It's NP-complete. It's in NP as we can verify a solution by checking if its a spanning tree (dfs and spanning) if its leaves (compute degrees in time $O(VE)$) are in T. It's NP-hard, as we can solve the Hamiltonian path problem with it.
We do the latter by looping over the following, for $u, v \in V$, so for $O\left(V^2\right)$ times. We check if there is a spanning tree who's leaves are in $T = \{u, v\}$: if so then there is a Hamiltonian path, otherwise, if none of the loops return a positive answer, then there isn't.

If G has a Hamiltonian path, say with endpoints u and v, then this path is a spanning tree with leaves in $T = \{u, v\}$, so that on that loop, we get a positive answer.
Conversely, any tree with leaves in $T = \{u, v\}$ is a Hamiltonian cycle. To see this, note first that the leaves must be $T = \{u, v\}$ exactly, as a tree has at least two leaves (the endpoints on a longest path on it). Then, start a walk at u and continue along previously unused edges of the tree of the walk. Each vertex on the walk (except for u, v) has degree $\geqslant 2$ in the tree, so its possible to always extend the walk at these vertices, along an unused edge of the tree. The edges of the walk can't lead to a previously encountered vertex, as this would close a cycle, which a tree doesn't have, so it's a path. The only way this path can end is at v. If this path didn't include all vertices of the graph, then deleting it from the tree results in a forest, each component of which is connected by a single edges to the path in the tree (otherwise, we'd have a cycle). Since each tree in this forest either has two leaves, not both of which could have been connected to the path in the tree, or is an isolated vertex, we get the existence of at least one additional leaf to u, v in the tree (the previously isolated vertex connected by a single edge to the path, or one of the leaves of the trees of the forest, the other being possible connected to the path in the main tree), contradicting the fact that the leaves are contained in $T = \{u, v\}$. So the path is a Hamiltonian one. □

Ex.NT: The worst-out-greedy algorithm for the MST problem provides us with a minimum spanning tree. But is this a narrowest spanning tree ?
The idea is that the worst-out-greedy algorithm will be leaving globally light weighted edges to form the tree. So if T_g is a tree from the worst-out-greedy algorithm and T_n is a supposedly narrower one, then by denoting T_g's heaviest edge e_g, then all of T_n's edges have smaller weight then e_g. This means that in the worst-out-greedy algorithm, they are all considered after e_g. But since e_g was included in T_g by the algorithm, deleting e_g disconnects the graph induced by the edges of lower weight than e_g. However, this graph is supposed to contain the tree T_n, since all of T_n's edges have smaller weight then e_g: in particular, it's connected ! Since we've arrived at a contradiction, we conclude that T_g is a narrowest tree, so that the worst-out-greedy algorithm solves our problem. □

Ex.PST:
The idea is to use weights so that the total weight of the tree encodes the edges used and then solve an MaxST problem. This can be done by assigning to edges e of color i the weight $w(e) = (|E|+1)^{i-1}$, so the code used is base $|E|+1$. If we solve the MaxST problem with these weights, the output tree will have the desired property. Indeed, assume for contradiction that there is some i for which the tree uses less then t_i edges of color i, but it uses t_j edges of color j for $j > i$. The weight of the tree is $\sum_{q\in[k]} \tau_q(|E|+1)^{q-1}$ where τ_q is the number of edge of color q used by the tree. We assumed that $\tau_q = t_q$ for $q < i$ and $\tau_q < t_q$ and we know that there is a tree for which $\tau'_q = t_q$ for $q \leqslant i$. By bounding $\sum_{q\in[k]} (\tau'_q - \tau_q)(|E|+1)^{q-1} = (\tau'_i - \tau_i)(|E|+1)^{i-1} + \sum_{q=1}^{i-1}(\tau_q - \tau'_q)(|E|+1)^{q-1} \geqslant (|E|+1)^{i-1} - \sum_{q=1}^{i-1}(|E|)(|E|+1)^{q-1} = 1 > 0$, we see that the output tree is not an MaxST, contradicting the nature of its construction. □

Ex.MaMu:
The symmetric difference $M_a \Delta M_u$ form even cycles or paths who's edges alternate between $M_a \backslash M_u$ and $M_u \backslash M_a$, for reasons explained in the proof of the characterisation of maximum matchings.
The paths can't be augmenting ones for M_u, as it's maximum, so each such path has at most one more edge in M_u then it does in M_a.
Next, each path must have at least one edge of M_a, otherwise the path is a single edge of M_u, which could

be added to M_a without breaking the matching property, contradicting maximality. So there are at most $|M_a|$ paths.
Thus there are at most $1 \times |M_a|$ more edges in M_u then in M_a.
Since the maximum of edges the matchings can have in common is $|M_a|$, since $|M_a| < |M_u|$, we can conclude with $|M_u| = |M_u \cap M_a| + |M_u \backslash M_a| \leqslant 2|M_a|$. □

Ex.BOC: First off, we can prove this for a connected graph, as bipartitions of the connected components of a graph can be merged to form a bipartition of all the graph.
So we may look at a spanning tree of the graph. The bipartition corresponds to the layers of the tree: we choose a root and partition the vertices on the parity of their distance to the root.
It could happen that when adding the edges not in the tree, vertices of the same partition become adjacent. But by the definition of our partitions, the cycle that this edge closes with the paths to the root in the tree must be odd, a possibility which we've excluded. So with all the edges considered, this bipartition remains valid. □

Ex.EV: Suppose an exposed u has no augmenting path in M, but we can augment M with augmenting path P. If u had an augmenting path P' in the next iteration, meaning in matching $M\Delta P$, then P' has to meet P, otherwise it would have been an augmenting path for u in M already. Let's call v the first vertex common to P and P'. It can't be an endpoint of P, because this would imply that P' restricted to from u to v was an augmenting path in M, as v was still exposed in M. So v is in the interior of P. If we then consider the path that follows P' from u to v and then then follows the path P to it's endpoint in the direction for which the edges at v are either both in the matching $M\Delta P$, or none of them are, then in the initial matching M, this would be an augmenting path for u.
In all cases, we reach a contradiction, so u won't have an augmenting path in the next iteration. Inductively, we seen that u won't have an augmenting path in any future iteration, so we may ignore it in the future iterations of the algorithm. □

Ex.MD: Dominoes cover two adjacent tiles, so if we model the problem by a graph who's vertices are the finite subset of $\mathbb{Z}^2$ and who's edges connect two vertices of from (n, m) which have one coordinate in common and the other coordinates at distance 1 (for example (2,3) and (3,3) are connected), then a cover by dominoes corresponds to a matching of this graph. So we're actually dealing with a matching problem. To see that the underlying graph is bipartite, notice that diagonal tiles can't be covered by dominoes. So we partition vertices (of from (n, m)) on whether the parity of their coordinates coincides or not and see that all edges are in the cut of the partition, in the sense that neighboring vertices are in different partition sets. Think of the coloring of a chess board. So we're actually dealing with a bipartite matching problem. □

Ex.BlBl: All vertices on the blossom are incident to an edge of the matching which is on the blossom, except for the basis of the blossom. So the basis is the only vertex which can be incident to an edge of M not on the blossom: when we shrink the blossom, all other edges incident to the blossom can't be in M, so b is incident to at most one edge of M, which therefore maintains the matching property.
If an augmenting path of G/B starts or ends in b, this means that b is unmatched, so that the basis of the blossom is unmatched too. Then we can always prolong this path until it reaches the basis, circling the path in the direction that maintains alternation of edges until we arrive at the basis. This will produce an augmenting path.
If an augmenting path of G/B has b as an inner vertex, the path passes through the basis and some other point of the blossom. Again, we chose circle the blossom from the basis until we meet the other incidence

vertex, circling in the direction such that the last edge on the blossom is one of M.
If an augmenting path of G/B doesn't pass through b or the entry and exit vertex on the blossom is the basis of the blossom, then no adjustments need to be made to the path for it to be augmenting in G. $\square$

Ex.MCSA:
To show this, we'll see that notation can have quite the impact on object studied. If we denote by $u(X,Y)$ the capacity of the edges going from a vertex of X to one of Y, then we have the property that $u(X,Y) = u(X \cap Z, Y) + u(X \cap Z^c, Y)$ and similarly on Y's side. We can then decompose the capacities into elementary pieces: $u(A \cup B, A^c \cap B^c) = u(A \cap B, A^c \cap B^c) + u(A \cap B^c, A^c \cap B^c) + u(B \cap A^c, A^c \cap B^c)$, $u(A \cap B, A^c \cup B^c) = u(A \cap B, A^c \cap B^c) + u(A \cap B, B \cap A^c) + u(A \cap B, A \cap B^c)$, $u(A, A^c) = u(A \cap B, A^c \cap B) + u(A \cap B^c, A^c \cap B) + u(A \cap B, A^c \cap B^c) + u(A \cap B^c, A^c \cap B^c)$, and $u(B, B^c) = u(A \cap B, A^c \cap B^c) + u(A^c \cap B, A^c \cap B^c) + u(A \cap B, A \cap B^c) + u(A^c \cap B, A \cap B^c)$. We see that the terms of the first two appear each exactly once in those of the last two decompositions, which have $u(A^c \cap B, A \cap B^c)$ and $u(A \cap B^c, A^c \cap B)$ in addition. Since capacities are positive, this means $u(A \cup B) + u(A \cap B) \leqslant u(A) + u(B)$.
Next, note that in general $u(V \backslash A) = u(A)$, as these vertices represent the same bipartition, so the same cut. So $u(A \cap (V \backslash B)) + u((V \backslash B) \cup A) \leqslant u(A) + u(V \backslash B)$ by submodularity, which says $u(A \backslash B) + u((V \backslash B) \cup A) \leqslant u(A) + u(B)$, and by $u((V \backslash B) \cup A) = u(V \backslash ((V \backslash B) \cup A)) = u(B \backslash A)$, we get the desired second inequality.

Finally, if A and B represent minimum s-t-cuts, then by noting that $A \cup B$ and $A \cap B$ represent s-t-cuts (they contain s and not t), that by minimality $u(A) = u(B) \leqslant u(A \cup B), u(A \cap B)$, and that assuming that one of the inequalities is strict leads to $u(A) + u(B) < u(A) + u(B)$, we conclude that $A \cup B$ and $A \cap B$ represent minimum s-t-cuts as well. In case A is a s-t-cut and B is an t-s-cut, then $A \backslash B$ is an s-t-cut and $B \backslash A$ is a t-s-cut,. Again, if A and B are minimum, then the second inequality implies that $A \backslash B$ and $B \backslash A$ are too. $\square$

3 Complexity and how to handle it

3.1 Exact exponential algorithms

"Exact exponential algorithms" Fomim and Kratsch: 3.2, 6.2, 9.2

Directed Hamiltonian path:

Definition:

In the **directed Hamiltonian path problem**, we're given a directed graph $G = (V, E)$ and we seek an algorithm to tell if the graph has a path P that uses every vertex of V.

We can reduce the problem to that of finding a directed Hamiltonian path from vertex s to vertex t, since if we can solve this problem with an algorithm, we can run it on the $\binom{|V|}{2}$ pairs of vertices of the graph and return a positive answer when one of the iterations returns a positive one.
A first approach to this problem is to check the $(|V|-2)!$ permutations σ of the vertices $V\backslash\{s,t\}$ on whether $s, \sigma(v_1), ..., \sigma(v_{|V|-2}), t$ forms a path (contains the arcs between vertices).

The problem can be relaxed further to that of finding the number of a directed Hamiltonian s-t-paths in the graph, as we can return a positive answer when this number is positive. This might seem more difficult at first, but it'll provide a faster algorithm for the initial problem!

The directed Hamiltonian s-t-paths are the s-t-walks of length $|V|-1$ that contain all vertices v_i. If we know the number W of s-t-walks of length $|V|-1$ in total, then finding the number of those using all v_i is equivalent to finding the number of those missing at least one of the the v_i.
We'll count these walks using inclusion-exclusion, which we state in the form most commonly found in combinatorics texts: $|A_1 \cup ... \cup A_m| = \sum_{j=1}^{m}(-1)^{m-1} \sum_{\substack{I \subseteq [m] \\ |I|=j}} |\cap_{i\in I} A_i|$
If we denote by A_i the set of s-t-walks of size $|V|-1$ that contain vertex v_i, then we can use $|\cup_{i\in[m]} \overline{A_i}| = \sum_{j=1}^{m}(-1)^{m-1} \sum_{\substack{I \subseteq [m] \\ |I|=j}} |\cap_{i\in I} \overline{A_i}|$, and by letting $|\cap_{i\in\emptyset} \overline{A_i}| = W$,
we get $|\cap_{i\in[m]} A_i| = \sum_{j=0}^{m}(-1)^{m} \sum_{\substack{I \subseteq [m] \\ |I|=j}} |\cap_{i\in I} \overline{A_i}|$, which is the number of directed Hamiltonian s-t-paths.
The only way this is useful is when there is a fast way of computing $|\cap_{i\in I} \overline{A_i}|$ for $I \subseteq V\backslash\{s,t\}$, the number of s-t-walks of length $|V|-1$ using none of vertices of I, aka. the number of s-t-walks of length $|V|-1$ in the graph $G\backslash I$.
The key point of our approach is that one can compute the number of s-t-walks of a graph G' of length k by considering entry (s,t) of A^k, where A is the vertex-adjacency matrix of G'. Since matrix multiplication takes $|V|^3$ multiplications, and we form $k-1$ of them (so $(k-1)|V|^3$ multiplications), we can find the number of s-t-walks of length $|V|-1$ in $|V|^4$ time.
Now, we can compute $|\cap_{i\in I} \overline{A_i}|$ for all $2^{|V|-2}$ subsets $I \subseteq V\backslash\{s,t\}$ in time $O\left(2^{|V|-1}|V|^4\right)$ and deduce the

the number of a directed Hamiltonian s-t-paths via inclusion-exclusion.

This is better then the first approach, as $2^{|V|-2}|V|^4 = O(|V|-2)!)$. Indeed, $\frac{2^{|V|-2}|V|^4}{(|V|-2)!} = \frac{|V|^4}{(|V|-2)...(|V|-5)} \times 2^5 \times \frac{2^{|V|-7}}{(|V|-6)!}$, where the first terms are asymptotically constant and the last is bounded by 1, as all $\frac{2}{k} \leqslant 1$ for $k = 2, ..., |V|-6$.
Note however that this approach is not constructive: it doesn't give us a directed Hamiltonian s-t-paths if there is one, it just tells us that there is one, unlike the first approach.

Maximum cliques or independence sets:

Definition:

We consider a graph $G = (V, E)$.
In the **maximum clique problem**, we seek a maximum size subset of vertices $W \subseteq V$ such that all pairs of vertices of W are linked by an edge, so $\forall v, u \in W, \{v, u\} \in E$.
In the **maximum independent set problem**, we seek a maximum size subset of vertices $W \subseteq V$ such that no pairs of vertices of W are linked by an edge, so $\forall v, u \in W, \{v, u\} \notin E$.

The two are related as follows: one problem is equivalent to the other, but on $\overline{G} = \left(V, \binom{V}{2} \backslash E\right)$.

A brute-force approach would be to check all $2^{|V|}$ subsets of V on being an independence set (in $O\binom{|V|}{2}$ time) and keep track of the largest one. We'll look for a more efficient way.

To find a maximum independence set in a graph, we study the properties of such sets. If v is in an independence set W, then none of its neighbours are in W, by definition. Conversely, if a vertex v has none of it's neighbours in a independence set W, then it must be in the independence set, for otherwise $W \cup \{v\}$ is a larger independence set.
This means that if we consider the closed neighbourhood $N(v) = \delta(v) \cup \{v\}$ of any vertex v, then there is at least one $u \in W$ also in $N(v)$, and for this vertex, $N(u) \cap W = \emptyset$.

This hints at a recursive way of finding a maximum independence set. Indeed, if we have found maximum independence set $W_{G\backslash N(u)}$ for the graphs $G\backslash N(u)$ for all $u \in N(v)$ for an initial vertex v, then the largest among the set $W_{G\backslash N(u)} \cup \{u\}$ is a maximum independence set of G. These sets are independence sets as the vertices of $W_{G\backslash N(u)}$ shared no edge and can't share an edge with $W_{G\backslash N(u)}$, being part of $G\backslash N(u)$. Since any maximum independence set W must contain a vertex in $u \in N(v)$ by our previous remark, and since $W\backslash u$ must be a maximum independence set of $G\backslash N(u)$ (it's an independence set as a subset of an independence set, and if there was a larger independence set U of $G\backslash N(u)$, $U \cup u$ would be a larger independence set of G, contradicting maximality), we're enumerating all candidates for maximum independence set, the largest of which is our output.
For the base case of the recursion, if $G\backslash N(u) = \emptyset$, we return an empty independence set of size 0.

We can prove that this algorithm terminates and with a correct output by induction on $|V|$. For the base case $|V| = 0$ we return an empty independence set of size 0. The more satisfying base case $|V| = 1$ returns the only vertex and is again correct. If we know the algorithm works on graphs with less then n

vertices, then for a graph with $n+1$ vertices, all (finitely many) recursive calls on graphs of form $G\backslash N(u)$, which have at most n vertices, terminate correctly by induction. So the algorithm terminates, and correctly by the above discussion, showing the induction step.

At each step of the recursion, we have to compute a maximum in $O(\deg(v))$, where v is the vertex we choose to "branch" on. So far we've chosen v arbitrarily, but we'll see that choosing v to be a vertex of minimum degree helps, even if this requires to keep track of degrees and compute their minima at each step.
So if $T(n)$ denotes the worst case runtime of the algorithm on a graph of n vertices, which occurs on some graph G. We have $T(n) \leqslant P(n) + \sum_{u \in N(v)} T(n - (\deg(u) + 1))$, where $P(n)$ is the polynomial worst case time required to compute the extrema at each step. By there worst case nature, P and T are increasing. By choice of v as a minimum degree vertex, we have $T(n - (\deg(v) + 1)) \geqslant T(n - (\deg(u) + 1))$, so that $T(n) \leqslant P(n) + (\deg(v) + 1)T(n - (\deg(v) + 1))$.
If we substitute this formula into itself, setting $s = (\deg(v) + 1)$, we get $T(n) \leqslant P(n) + s(P(n - s) + sT(n - 2s)) \leqslant (1 + s)P(n) + s^2T(n - 2s))$. Doing this until we reach $T\left(n - \left\lfloor \frac{n}{s} \right\rfloor s\right)$, we get $T(n) \leqslant P(n)\left(1 + s + ... + s^{\lfloor \frac{n}{s} \rfloor - 1}\right) + s^{\lfloor \frac{n}{s} \rfloor} T\left(n - \left\lfloor \frac{n}{s} \right\rfloor s\right)$.
This is where Fomin and Kratsch or the paper they got this example from get sloppy. There is no efficient way of bounding $T\left(n - \left\lfloor \frac{n}{s} \right\rfloor s\right)$ by a polynomial, as all we know is $n - \left\lfloor \frac{n}{s} \right\rfloor s \leqslant s$ and s depends on n in a non-exploitable way...
We have to restrain ourselves to graphs such that $\min(\deg(v) : v \in V) \leqslant k$ for a constant k independent of n. Then $s \leqslant k$, so that, by monotonicity and positivity, $T(n) \leqslant (P(n)+T(k))\left(1 + s + ... + s^{\lfloor \frac{n}{s} \rfloor - 1} + s^{\lfloor \frac{n}{s} \rfloor}\right) = (P(n) + T(k))\frac{s^{\lfloor \frac{n}{s} \rfloor} - 1}{s - 1}$ (when $s > 1$), where $(P(n) + T(k))$ has artificially been turned into a polynomial in n. Since $s \in \{1, ..., k\}$, we have bounds $T(n) \leqslant (P(n) + T(k))\left(\left\lfloor \frac{n}{s} \right\rfloor + 1\right)$ for $s = 1$ and $T(n) \leqslant (P(n) + T(k))s^{n/s}$. Since $s \mapsto s^{n/s}$ achieves its maximum value $e^{n/e}$ when $k \geqslant 3$, we have $T(n) \in O\left((P(n) + T(k))e^{n/e}\right)$, which is an improvement over brute force, as $e^{1/e} \approx 1,44 < 2$.

The Held-Karp algorithm:
A travelling salesman has to visit n cities in a tour. Since he's a scammer, he can't return to a city after having visited it, except for his hometown, which will forgive him after he visited all other cities. By this, we mean that the salesman has to chose a cyle of cities, visiting each just exactly once, except for the first one, which he visits exactly twice. Travelling from city i to city j incurs a cost $d_{ij} \geqslant 0$, if it is possible. The problem is that of finding a cycle of all cities of minimum cost, if this is possible.

We can model this as a weighted graph and ask for a minimum weight Hamilton cycle. However, we can reduce the problem to a complete graph on n vertices. Indeed, if G is the graph and d its weights, then we can complete it to K_n, using weights $w = 1 + \sum_{e \in E_G} d_e$ for edges previously not in the graph. If we then solve the problem, and find a solution with value greater then $\sum_{e \in E_G} d_e$, then the only way this could have happened is if we used an edge that wasn't in G. Since the cycle is minimum, we'll then know that finding a Hamiltonian cycle in G is impossible, as it would have value less then $\sum_{e \in E_G} d_e$, contradicting the previous minimality.

Travelling salesman problem (TSP):

For an complete graph K_n with edge weights $d \geqslant 0$, we seek a minimum weight cycle containing all vertices of the graph.

The problem reduces to that of finding a minimum weight Hamiltonian path. Indeed, fix a vertex v. The TSP cycle will use some edge $\{v, u\}$. Deleting that edge yields a minimum Hamiltonian v-u-path (if it wasn't minimum for the Hamiltonian constraint, we'd contradict optimality of the TSP cycle by closing the smaller path to a Hamiltonian cycle with $\{v, u\}$). So by fining the minimum Hamiltonian v-u-path for all u other then v, we can find a TSP cycle by taking the smallest weighted cycle among those obtained by adding $\{v, u\}$ to the paths.

We'll try to find a minimum Hamiltonian v-u-path recursively. The problem will be that the is to "scale" the Hamiltonian property to sub-problems. We can start by considering a minimum Hamiltonian v-u-path. On it, we consider the predecessor p of u, and the subpath from v to p. The subpath has to be a minimal v-p-path, for the property of containing all vertices except for u. If there was a ligther one, we could add edge $\{p, u\}$ to it to get a Hamiltonian v-u-path (since u wasn't on the path already, and the subpath used all vertices except u (for the Hamiltonian property)) of lesser value, contradicting minimality. Hence, if for any p, we have a method to find all v-p-paths minimum for the property of using all vertices but u, we can find the minimum Hamiltonian v-u-path as the path of minimum cost among the paths obtained by adding $\{p, u\}$ to the previous minimum paths, ranging over p.

Can you see a pattern emerging in the recursion ?
To find a minimum v-p-path $P_{S,p}$ using all vertices of a set $S \subseteq V$ and no other vertices, who's length we'll denote by $D_{S,p}$, we note that it is among the paths of form $P_{S\backslash p,j} \cup \{j, p\}$ for j ranging in $S\backslash v$, since by splitting $P_{S,p}$ into $P_{S\backslash p,j} \cup \{j, p\}$ where $j \in S$ is the predecessor of p on $P_{S,p}$, we see that $P_{S\backslash p,j}$ must be minimal wrt. the property of using all vertices of a set $S\backslash p \subseteq V$ and no other vertices, for similar arguments as before (otherwise, we can find a lighter path then $P_{S,p}$ for its defining properties, contradicting its minimality). In fact $P_{S,p}$ will be the least weight path among $P_{S\backslash p,j} \cup \{j, p\}$ for j ranging in $S\backslash v$, so that $D_{S,p} = \min\limits_{j \in S\backslash\{v,p\}} (D_{S\backslash p,j} + d_{jp})$.
This holds if $|S| \geqslant 3$. The base case $|S| = 2$ is just $D_{S,p} = d_{vp}$. We see that the recursion progresses in the sizes of the set S. We start with the base case $|S| = 2$ with $D_{S,p} = d_{vp}$, and for $O(|V|)$ iterations ranging over increasing $|S|$, we compute the $D_{S,p}$ and their corresponding paths by a minimum $D_{S,p} = \min\limits_{j \in S\backslash\{v,p\}} (D_{S\backslash p,j} + d_{jp})$ in $O(|V|)$ time, using the $D_{S\backslash p,j}$ values of the previous iteration. There are at most $2^{|V|}$ possible $S \subseteq V$ that need be compute. So the Held-Karp algorithm can be bounded by $O\left(|V|^2 2^{|V|}\right)$ in time.

3.2 NP-completeness

Beyond NP:
Some tasks are not verifiable in time polynomial in the input data, depending on what one considers to be input data and a verification algorithm. A simple example of this phenomenon is the problem of deciding if a set $S \subseteq \mathcal{P}([n]^n)$ (power set) is the set of permutations of $[n]$. Any algorithm based on checking if a permutation σ is in S will have to make $n!$ checks, which isn't polynomial in n.
In general, enumeration tasks can be exponential in the input data, since the objects to enumerate are exponential in the input data. This is the case for checking if a polytope has at least k vertices, since the n-cube is defined by $2n$ half-spaces, and has 2^n vertices. The enumeration of spanning trees falls in this category too.

Non polynomial time reductions:
One can solve NP-complete problems with polynomial time algorithms. This may sound lie P=NPc, but the key point is that the reduction isn't polynomial. For example, on can solve SAT with graph search algorithms, which are polynomial. To do this, we construct a binary tree so that each layer corresponds to the choice of a truth assignment of of a variable, and add a target vertex related to the vertices of the last layer if the truth assignment satisfies the instance of SAT. Then a path from the to of the tree to the target vertex corresponds to a satisfying assignment. However, the graph has a number of nodes exponential in the number of variables, and the computation of the edges required us to solve SAT in the first place.

An interesting question for problems involving numerical values is weather the sign of these values affects the difficulty of the problem.

Ex.PsP: We consider two versions of the partition problem: in the first, we're given n positive integers $a_i \in \mathbb{N}$ and in the second, we're given n integers $b_i \in \mathbb{Z}$. In both cases, we seek to decide if there is an $I \subset [n]$ so that $\sum_{i\in I} a_i = \sum_{i\notin I} a_i$, or $\sum_{i\in I} b_i = \sum_{i\notin I} b_i$ respectively. Show that one version allows to solve the other.

The situation is entirely different for cut problems. The min-cut problem for positive capacities can be solved in polynomial time, as we've seen in the chapter on flows, whereas the min-cut problem for general capacities is NP-complete! Indeed, the problem is equivalent to finding a maximum cut in the graph with opposite capacities, and this problem is NP-complete, as we'll now show.

NP-completeness of Max-cut via Partition:
Max-cut is clearly in NP: given a bipartition of vertices of a graph, we can count in $|E|$ iterations the edges that belong to the cut, by checking the partition sets of their endpoints.
We'll reduce an instance of the partition problem to a max-cut problem.
The partition problem asks, given numbers $(c_i)_{i\in[n]}$, whether there is a bipartition of indices into $S \subseteq [n]$ and $[n]\backslash S$ such that $\sum_{i\in S} c_i = \sum_{i\notin S} c_i$. It's NP-complete.
We can reduce it to a decision version of the max-cut problem with the following tricks. We consider a complete graph on vertices $[n]$, with edge weights $w_{ij} = c_i c_j$, so that the value of the cut of the bipartition into $S \subseteq [n]$ and $[n]\backslash S$ is $V(S) = \sum_{i\in S, j\notin S} c_i c_j = \left(\sum_{i\in S} c_i\right)\left(\sum_{i\notin S} c_i\right)$. Introducing the total sum

$C = \sum_{i\in[n]} c_i$, we have $V(S) = \left(\sum_{i\in S} c_i\right)\left(C - \sum_{i\in S} c_i\right)$. Then, we can exploit the fact that $x \mapsto x(C-x)$ is maximum in $\frac{C}{2}$ (alone) together with the fact that the existence of an S for which $\sum_{i\in S} c_i = \sum_{i\notin S} c_i$ implies $\sum_{i\in S} c_i = \sum_{i\notin S} c_i = \frac{C}{2}$ (divide the total sum along S and apply equality to see it) to complete the reduction. Indeed, the graph has a cut of value at least $\frac{C^2}{4}$ precisely when an S for which $\sum_{i\in S} c_i = \sum_{i\notin S} c_i$ exists: if $V(S) = \frac{C^2}{4}$, the objective is maximum, which requires $\sum_{i\in S} c_i = \frac{C}{2} = \sum_{i\notin S} c_i$, providing the desired S, and the converse is obtained by computing $V(S)$ for an S with the partition property. This shows the reduction, and therefore also max-cut's NP-completness.

Ex.HypPath: Recall that a (loopless) directed hypergraph on vertices V has hyperarcs $e \in H$, where $e = (T_e, H_e)$ for non-empty and disjoint $T_e, H_e \subseteq V$, where T_e is the tail and H_e the head of arc e. We can try to generalise the notion of a path from a source-set S to a target-set F as follows: a superpath from S to F is a finite sequence of hyperarcs $e_1, ..., e_n$ so that $T_{e_1} \subseteq S$, $T_{e_j} \subseteq S \cup_{i<j} H_{e_i}$ for $j \leqslant n$ and finally $F \subseteq S \cup_{i\leqslant n} H_{e_i}$. In words: we keep adding hyperarcs with tails in previously visited vertices, starting from S, until all vertices of F have been visited. This is more of a generalisation of a walk than that of a path, so we define an S-F-hyperpath to be a inclusionwise minimal S-F-superpath, in the sense that deleting any subset of hyperarcs from it breaks the superpath property. We can give hyperarcs weights $w : H \to \mathbb{Z}$.
Given a directed hypergraph (V, H), source-set S and target-set F and weight $w : H \to \mathbb{Z}$, show that the task of deciding whether a hypergraph has an a path of total weight less then k is NP-complete.

3.3 Input specification and randomisation

Probabilistic methods for algorithmic discrete mathematics, Probabilstic Analysis of Algoithms, Chapter 2, Chapter 13 and 10.2 Tardos Kleinberg: distinguish approximation algorithms and input specification.

Some tasks we showed to be NP-complete can be solved in polynomial time, if the input has an additional particular structure, as we'll see now. If we have a probability distribution over the inputs, we can compute the probability of a polynomial time algorithm to terminate correctly, or even compute the expected runtime of a worst-case exponential time algorithm that terminantes correctly, and hope this expected time to be polynomial (which may happen when the distribution depends on the structure of the input, so that exponentially high runtime occurs with exponentially low probability).

Maximum independent set on forests:
This is an example where we show how a difficult problem has an efficient solution on a cetrain class of inputs. An idea for finding large indipendence sets is to add vertices of low degree, as this excludes a small number of vertices, the neighbourhood, from being added to the independence set. In the case of forests, we're guaranteed to have leaves, which seem good for our idea. The key remark is that one can obtain a maximum independence set containing a particular leaf of the forest!
Indeed, if we consider a maximum independent set S that doesn't contain leaf l, then by maximality, it must contain the unique neighbour v of l, as otherwise, we could add l to the set without violating independence, contradicting its maximality. In this case, we can which to $(S\backslash v) \cup l$ and obtain an independent set of equal, hence maximum size, that contains the leaf. So a maximum independence set containing all leaves must exist.

We can then consider the recursive search of a maximum independence set explored in the "exact exponential chapter". We include a the leaf in S and solve the problem on the forest obtained by deleting the leaf and its unique neighbour. The reason we delete the neighbour is to make sure that a maximum independence set of this subgraph yields a maximum independence set of the initial forest, when we add the leaf to it. Indeed, the only way that Independence could be destroyed is if the neighbour of the leaf was in the maximum independence set of the subgraph, hence our reason for deleting it.
As base case of the recursion, we have isolated vertices or empty trees: in the first case, we add the isolated vertices to the independence set.
The reason this recursion produces a maximum independence set is that for the maximum independence set containing the leaf (that we know exists), the part that is in the subgraph must form a maximum independence set of the subgraph for otherwise, we would have a larger one that we could add the leaf to, obtaining an independence set of larger size then the maximum one, a contradiction.
Searching for a leaf can be done in $O(|V||E|)$ time, and since at each iteration, we delete 2 vertices, we iterate at most $|V|/2$ times, so the runtime is in $O\left(|V|^2|E|\right)$.

Bin packing for random bins:
We consider the bin packing problem in the following form: given n items with item i of volume $v_i \in [0,1]$, we seek the smallest number of bins k of volume 1 so that we can place items in bins according to $m : [n] \to [k]$, so that the bins don't overflow, aka. $\sum_{\{i:m(i)=j\}} v_i \leqslant 1$ for all bins j.
We'll give a deterministic algorithm for this problem. We will then analyse its expected output and expected runtime under the following probability distribution on the inputs: each v_i is uniform on $[0,1]$ and independent of the other v_j. Note that this is not an approximation algorithm, and not a randomised one either: all we do is analyse its behaviour under randomisation.

The deterministic lower bound on the number of bins $\sum_{i=1}^{n} v_i \leqslant k$ now provides a lower bound on the expected number of bins $E(k) \geqslant \sum_{i=1}^{n} E(v_i) = \frac{n}{2}$.

We'll start our packing by preprocessing: we place each i with $v_i \geqslant \alpha$ in a separate bin, for a parameter α to be chosen later. By the binomial distribution, we expect $n\alpha$ items to fall in this category. We'll now try to fit the left-over medium sized items into bins, by pairing them up as follows. We first sort the N (with $E(N) = n(1-\alpha)$) remaining items by increasing volume $x_1 \leqslant ... \leqslant x_N$, and check if we can fit the small i and the large $N-i$ in a new common bin (if $x_i + x_{N-i} \leqslant 1$), which we do if possible, and place them in two new individual bins otherwise, for i from 1 to $\left\lfloor \frac{N}{2} \right\rfloor$. If N is odd, we place item $\left\lfloor \frac{N}{2} \right\rfloor + 1$ in a new individual bin.

The expected number of bins needed in the second phase is at most $1 + \sum_{i=1}^{\lfloor \frac{N}{2} \rfloor} 1.P(x_i + x_{N-i} \leqslant 1) + 2.P(x_i + x_{N-i} > 1)$. With a lot of computation and probability theory and the fact that we sorted the list, one can show that $P(x_i + x_{N-i} > 1) \leqslant \frac{1}{n}$. This way, bounding $P(x_i + x_{N-i} \leqslant 1)$ by 1, we get a bound of $1 + \frac{N}{2} + 2\frac{N}{2n} \leqslant \frac{n(1-\alpha)}{2} + 2$. So the total number of bins to expect as output is at most $n\alpha + \frac{n(1-\alpha)}{2} + 2 = \frac{n}{2}(1-\alpha) + 2$.
Preprocessing requires n checks, after which an expected amount of $n(1-\alpha)$ volumes have to be sorted and $\frac{n(1-\alpha)}{2}$ sums have to be computed and compared. So this algorithm runs in (expected) linear time.

3.4 Approximation algorithms

So for we've been trying to solve combinatorial optimization problems exactly. But for many purposes, it's satisfying enough to find feasible solutions with reasonably good objective value, if as trade-off we can compute these approximate solutions efficiently.

For example, consider the problem of coloring the vertices of a graph so that no vertices joined by an edge have the same color. We can consider the follwing "greedy" algorithm: order the vertices in some way and color the vertices with a color/label in $\{1, ..., |V|\}$ that is smallest among the colors not used on the neighbours of the current vertex that came before in the ordering. For a given vertex v, the worst case is that all its neighbours were ordered before it and received different colors, in which case we need $deg(v) + 1$ colors at this stage at least. Therefore, this algorithm produces a $(\Delta + 1)$-coloring in the worst case, where $\Delta = \max(\deg(v) : v \in V)$.
This is an example of an algorithm that may not produce an optimal solution, but one with a quality that can be estimated/bounded systematically. In what follows, we seek an even stronger seal of quality: we want to know how well the algorithms solution compares to the optimal solution for the instance. This might seem impossible to do without having a way of computing the actual optimal solution, but you'll see a large amount of examples in this section.
This is why we turn our interest to:

Definition:
An **approximation algorithm** for an optimization problem with attainable optimal value $z_{opt} > 0$ is a polynomial time algorithm that returns a feasible solution with objective value z_{approx}, such that there is a constant or function of the input data $\alpha > 0$, so that $1 \geqslant \frac{z_{approx}}{z_{opt}} \geqslant \alpha$ for maximisation problems and $1 \leqslant \frac{z_{approx}}{z_{opt}} \leqslant \alpha$ for minimisation problems.

We seek approximation algorithms with α close to 1, as this means that the approximate value we get is almost optimal.

As a first example, we give a 2-approximation algorithm to the bin packing problem, introduced in the introductory section of the integer programming chapter.
Greedy BPP heuristic:
We add the item to the first bin with unused space among those bins that can fit the item and add an new bin if no bin can fit the item. This is known as the "first fit" or "greedy" heuristic.
The key fact is that this algorithm can't terminate with more then two bins with less or equal then half capacity: the items of the latter bin could have been put in the first one, so there was no need to make a new one. This means that for the volume B_i packed in bin b_i and a number b of bins, $\sum a_i = \sum_{i=1}^{b} B_i > \frac{1}{2}(b-1)$ because for all $i \leqslant b-1$, $B_i > \frac{1}{2}$. Reformulating and recalling the that optimal number of bins OPT must contain all the volume, so that $OPT \geqslant \sum a_i$, we get $2OPT \geqslant 2\sum a_i > b-1$ and by integrality of the number of bins, $2OPT \geqslant b$. The algorithm runs in polynomial times, as there are at most n bins and we therefore check at most n bins in each iteration and recompute their left-over capacity once in such an iteration: this is $O(n^2)$ runtime. So this is a 2-approximation algorithm.

Ex.SAT2: Give a 2 approximation algorithm for Max-SAT which runs in linear time. That is, find an

algorithm that finds a truth assignment that satisfies at least half the clauses.

Ex:SteinerKcut: In the Steiner k-cut problem, we given a connected graph with edge weights $w \geqslant 0$, and a set of vertices X called terminals. We seek a minimum weight set of edges who's removal produces k connected components, each of which contains at least one terminal. Consider the following heuristic: build a Gomory-Hu tree of the graph and iteratively delete the cheapest edge so that two terminals that were in the same component of the tree are now separated for $k-1$ deletions. The union of the cuts represented by these edges in the Gomory-Hu tree will be our output. Show that this is a k-cut and that this algorithm is a $\left(2-\frac{2}{k}\right)$-approximation algorithm.

Ex:KnapsackDensity:
Imagine a knapsack problem where the objects are liquids that you can take fractional amounts of. How woould you solve th problem in that case ? Use this to create an approximation algorithm for the standard knapsack problem.

Is the bound tight?
An interesting question, when we have a α-approximation algorithm is whether α is the best possible ratio truely attained by the algorithm. Recall that α is obtained from bounds in the theoretical analysis of the algorithm. Perhaps, a better analysis may have given a tighter ratio ?
In some cases, we can show the the analysis is the best by giving an instance in which the bound is tight. For example, consider the greedy approach to maximum matching, which yields a maximal matching, that is at least as large as half of a maximum matching. If we consider the bipartite graph on 4 nodes u_1, u_2, v_1, v_2 with edges $\{u_1, v_1\}, \{u_1, v_2\}, \{u_2, v_1\}$, and an ordering so that the greedy algorithm selects edge $\{u_1, v_1\}$ first, then we get a maximal matching of size 1, whereas the maximum matching $\{u_1, v_2\}, \{u_2, v_1\}$ has size 2, so that the bound is tight in this case.
This does not mean that we can't find an approximation algorith with a better α for the problem, but only that the current algorithm can't perform better then with a ratio of α.

Next, we refine the heuristic, to get an approximation algorithm this a ration depending on the input data.
Sorted greedy BPP heuristic:
First, we sort the items from least volume to most. Then we use the previous first-fit heuristic.
The algorithm is polynomial in time, as the last one was and sorting is polynomial. Let's see what this small modification of sorting allows us to say.
The last item has volume $a_n = \max_i(a_i)$. We may assume that no item has value 1 as otherwise it's clear that it needs a bin to its own, so $a_n < 1$. By the first-fit heuristic, a_n can't have been placed in previous bins: otherwise, there still was enough space in the previous bin to accommodate a_n and in particular all the a_i in the last bin, as $a_i \leqslant a_n$, so that the first-fit rule was violated. This means $a_n > 1 - B_i$ for the volume B_i used by bin b_i, for all but the last bin. Thus, for b output bins, summing the inequalities gives $(b-1)a_n > (b-1) - \left(\sum a_i - B_b\right)$ and by rearranging and bounding we get $\frac{1}{1-a_n} OPT \geqslant \frac{1}{1-a_n} \sum a_i > (b-1)$ and by integrality $\geqslant b$. So this is a $\frac{1}{1-\max_i(a_i)}$-approximation algorithm: one with a ratio depending on the input data.
For items with small weight (less then 1/2), this ratio is better then the previous one.

Not all heuristics turn out to give approximation algorithms !

Example of an algorithm that isn't an approximation algorithm:

Recall the vertex cover problem introduced in the section on LP relaxations and total unimodularity. In it, we look for a set of vertices of smallest size such that all edges have at least one endpoint in that set. A greedy heuristic would be to add the the vertex of highest degree to the cover, then proceed iteratively on the graph with this previous vertex deleted. It turns out that this isn't an approximation algorithm for a constant α. To see this, we'll construct a family of graphs for which the ration of the greedy solution to the optimal one gets smaller and smaller, preventing the existence of a universal constant α bounding this ratio from below for all graphs that can be inputs to the algorithm.

This family of graphs are bipartite graphs with bipartition into L and R. The set L has r vertices, for a parameter r, and the set R can be subdivided into subsets R_i where there are $\left\lfloor \frac{r}{i} \right\rfloor$.

We number the vertices l_j of L from 1 to r and those from R_i as r_{ik} for k from 0 to $\left\lfloor \frac{r}{i} \right\rfloor - 1$. We connect r_{ik} to l_k, $l_{k+\left\lfloor \frac{r}{i} \right\rfloor}$, $l_{k+2\left\lfloor \frac{r}{i} \right\rfloor}$, and so on until $l_{k+(i-1)\left\lfloor \frac{r}{i} \right\rfloor}$. This way r_{ik} has degree i. To see that $deg(l_j) \leqslant r$, note that for all i, there's at most one r_{ik} connected to l_j, so that we get the result by summing this of the r sets R_i. We prove that there's at most one r_{ik} connected to l_j: otherwise, r_{ik} and r_{iq} are connected to l_j, meaning $k + x\left\lfloor \frac{r}{i} \right\rfloor = j = q + y\left\lfloor \frac{r}{i} \right\rfloor$ for some $x, y \leqslant i-1$, so that $\left\lfloor \frac{r}{i} \right\rfloor$ divides $k - q \in]-\left\lfloor \frac{r}{i} \right\rfloor, \left\lfloor \frac{r}{i} \right\rfloor[$, so that $k - q = 0$. In this graph L and R are covers.

For such graphs, the greedy algorithm could choose $r_{r,0}$ as first node instead of a node from L, since it also has degree r. By deleting it from the graph, all the nodes of L loose an egde, so that the new upper bound on the degrees of L is $r-1$. Next, the greedy algorithm could choose $r_{r-1,0}$ as next node, and it's deletion lowers the upper bound on the degrees of L is $r-2$ since all nodes from L have at most one neighbour in R_{r-1} and those that did have one have lost their neighbour after deletion. These two facts are maintained after selection and deletion of all nodes in an R_i, so that the greedy algorithm could select all of R, which is a cover.

This cover has size $ALG = \sum_{i=1}^{r} \left\lfloor \frac{r}{i} \right\rfloor \geqslant \sum_{i=1}^{r} \left(\frac{r}{i} - 1\right) = r\sum_{i=1}^{r} \frac{1}{i} - r \geqslant r\left(\sum_{i=1}^{r} \frac{1}{i} - r\right) \sim r\log(r)$.

So if this was an α-approximation algorithm, then $0 < \alpha \leqslant \frac{OPT}{ALG} \leqslant \frac{r}{r\log(r)} = \frac{1}{\log(r)} \xrightarrow[\infty]{r} 0$ as $OPT \leqslant r$ since L is a cover of size r. By taking $r_n = n$, we construct a sequence of graphs forcing the approximation ration to be closer and closer to 0, contradicting its positivity.

The question of how good an approximation algorithm can be is deeply related to the question P vs. NP. If we find an approximation ratio of 1, we've actually solved the problem in polynomial time. In particular cases, we can make better statements of this form. If one assumes that $P \neq NP$, then these statements provide lower/upper bounds on the possible approximation ratios a problem can have.

Example of an bound on the possible ratio of an approximation algorithm:

We'll show BPP solves the following NP-complete problem, called PARTITION: for weights a_i in a finite index set, can one partition this index set in two so that the sum of the weight over each partion set are equal ?

To solve PARTITION with BPP, we set $a_i = \frac{2c_i}{\sum c_j}$ which is in $[0,1]$ when all $c_i \leqslant \frac{1}{2}\sum c_j$. If the latter isn't the case, then there are no partitions, since such a partition has cost $\frac{1}{2}\sum c_j$, and one of the partition would have to contain the $c_i > \frac{1}{2}\sum c_j$.

If the a_i can be packed into less then 2 bins, then $\sum a_i \leqslant 2$ but since $\sum a_i = \frac{2\sum c_i}{\sum c_i} = 2$ the bins are full, and in particular each contains an equal amount of volum so that $\sum_{i\in b_1} \frac{2c_i}{\sum c_j} = \sum_{i\in b_2} \frac{2c_i}{\sum c_j}$ which simplifies and gives partition b_1.
Conversely, if such a partition exists, by packing the items according to this partition, which is possible as $\sum_{i\in S} a_i = \frac{2}{\sum c_j}\sum_{i\in S} c_i = \frac{2}{\sum c_j}\frac{\sum c_j}{2} = 1$, we need exactly two bins.
So BPP solves PARTITION and is therefore NP-hard.

If $P \neq NP$, there cannot be an approximation algorithm with ratio smaller then $\frac{3}{2}$ for BPP. If there was, the solution ALG would verify $\frac{3}{2}OPT \geqslant ALG$, and since $ALG > 3 \Leftrightarrow OPT > 2$ and by integrality and contraposition $ALG \leqslant 2 \Rightarrow OPT \leqslant 2$, the approximation algorithm would be a polynomial time algorithm that decides if items can be packed in less then 2 bins, thereby solving PARTITION. This would mean $P = NP$.

TSP inapproximability:
In the previous example, we showed that unless P=NP, there is a lower bound on the possible approximation ratio for approximation algorithms to certain problems. We now give an even more extreme example: we'll show that the TSP can't be approximated by approximation algorithms, unless P=NP.

We'll assume that we have a α-approximation algorithm for TSP (with $\alpha \geqslant 1$) and use it to solve the NP-complete Hamiltonian circuit problem with it. Recall that this problem asks to find a tour (visit all vertices exactly once) in a graph $G = (V, E)$. We imitate the redaction of TSP to the Hamiltonian circuit problem: for a given instance of the latter, we construct a TSP instance by setting weights $w_{uv} = \begin{cases} 1 : \{u, v\} \in E \\ 2 + q : \ else \end{cases}$ for some constant $q \geqslant 0$ which we'll choose later to make the whole thing work. The idea of this reduction is to penalise the use of edges not in G by the TSP solution. Indeed, G has a Hamiltonian circuit precisely when the optimal TSP solution has value $|V|$: any Hamiltonian circuit must have $|V|$ edges as it visits all vertices once, and the edge weights are all greater then 1, so any TSP solution has value at least $|V|$. If a Hamiltonian circuit exists in G, then it's a solution to the TSP with value $|V|$, so it's optimal. Conversely, the TSP has optimal value strictly greater then $|V|$, no Hamiltonian circuit can exist.

We'll now choose q depending on α to show that we can decide the latter question with an α-approximation algorithm. For the output value A of such an algorithm, $\frac{A}{OPT} \leqslant \alpha$. If the tour given by the algorithm has value $|V|$, then its a Hamiltonian tour, and otherwise, it uses at least one edge of weight $2+q$, so that $(|V|-1).1 + 2 + q \leqslant A$. This means $\frac{(|V|-1)+2+q}{OPT} \leqslant \frac{A}{OPT} \leqslant \alpha$ so for $q = (\alpha-1)|V| > 0$, we have $\frac{1+\alpha|V|}{OPT} \leqslant \alpha \Leftrightarrow 1 \leqslant \alpha(OPT - |V|) \Rightarrow OPT > |V|$, so that there can't be a Hamiltonian circuit.

Approximation schemes:
Sometimes, it's possible to get a family of approximation algorithms who's approximation algorithm gets increasingly good (close to 1). This is known as an approximation scheme. Possible trade-offs in this situation is that the runtime increases as the approximation ratio decreases, or that one of the parameters of the problem appears exponentially in the runtime.

Definition:
An **approximation scheme** for an optimization problem with optimal value OPT is a family of polynomial time (in the problem input data) algorithms indexed by $n \in \mathbb{N}$ that return a solution to the problem with value ALG_n, so that $ALG_n \geq (1 - \frac{1}{n})OPT$ for maximisation problems and $ALG_n \leq (1 + \frac{1}{n})OPT$ for minimisation problems.

A **fully polynomial time approximation scheme** is an approximation scheme who's algorithms are polynomial time in the problem input data as well as the index n.

We give an example of an approximation scheme for a job scheduling problem. This scheme will be based on a first algorithm for job scheduling, which we present now.
In this job scheduling problem, we're given m machines and n jobs to be performed on them. The problem consists in finding m lists, where the list of machine i lists the jobs to be performed on it in the order specified by it. Each job j has requires an amount p_j of time to be processed. If l_i is the list of machine i, then the time needed for it to process its jobs is $\sum_{j \in l_i} p_j$. The makespan $M = \max_i \left(\sum_{j \in l_i} p_j \right)$ of a schedule is the time necessary for all jobs to be processed, and it's the quantity we#d want to minimise.

We construct an approximation algorithm for this problem using the local search heuristic: we start with a given schedule (for example, all jobs on one machine) and improve it to a next schedule, until no more improvements can be done.
In our case, this improvement corresponds to finding a last job j to be processed and checking if we can lower the makespan by letting it be processed by a different machine. This corresponds to the existence of a machine i so that $\sum_{j \in l_i} p_j < M - p_j$. In fact, we'll search for the machine that finishes earliest, with total processing time $N = \min_i \left(\sum_{j \in l_i} p_j \right)$, and check if $N < M - p_j$. If it is the case, we make a new schedule in which j is processed by the machine that previously finished earliest. We keep performing these improvements, until the condition $N < M - p_j$ is violated, at which point we stop and return the last schedule.

Before investigating if this algorithm ever terminates, we investigate its raison d'être: is it an approximation algorithm ?
At termination, if the last job to be processed is j, $M - p_j \leqslant N$ so that all other machines need at least a time of $M - p_j$ to finish. This means that at this stage, the machines have together dealt with $m(M - p_j)$ of the total processing time, with none of them being done (or just finishing at that exact moment), so that $m(M - p_j) \leqslant \sum_{j \in [n]} p_j$, as $\sum_{j \in [n]} p_j$ is the time necessary to do all jobs. So $M \leqslant p_j + \frac{1}{m} \sum_{j \in [n]} p_j$. We can bound these last two terms of the right side of the previous inequality by the minimum makespan over all schedules, so that this shows that we're dealing with a 2-approximation-algorithm. Indeed, since j must be processed in an optimal schedule of makespan M^*, we have $p_j \leqslant M^*$, and by writing $\sum_{j \in [n]} p_j = \sum_{i \in [m]} \left(\sum_{j \in l_i^*} p_j \right)$ for an optimal schedule given by lists l_i^*, $\frac{1}{m} \sum_{j \in [n]} p_j \leqslant M$ is due to the fact that the average of a sequence is less then it's maximum.

We now turn to the question if the algorithm terminates, and if so, in polynomial time. This usually requires more work for local search algorithms. To answer the question, we point to the particular choice we made during the improvement steps. By selecting the machine that finished earliest to do the last processed job, it turns out that we're assuring that this job won't be rescheduled to another machine ever again by the algorithm. This means that after n iterations, the algorithm must terminate. To see this, we first not the the earliest completion time of a machine N_k increases (not necessarily strictly) with the number of iteration k. Indeed, the machine that finishes in N_k gets the additional p_j, the machine the finishes in M_k now finishes in $M_k - p_j > N_k$, and all other machines finished in time $\geqslant N_k$ by definition of N_k, so $N_{k+1} \geqslant N_k$. With this in mind, if job j has been rescheduled to machine i in iteration k, then if it were rescheduled to machine y in a later iteration $q > k$, then this would mean that $M_q = N_k + p_j$ (due to how we choose j in the iteration) and $M_q - p_j > N_q$, implying $N_k > N_q$. This contradicts the fact that N_k increases with the number of iteration k !

We now turn to our approximation scheme.
To motivate it, recall the bound on the makespan $M \leqslant p_j + \frac{1}{m}\sum_{j\in[n]} p_j$ we previously obtained. If we could relate the processing time p_j of the last job to be executed to the average processing time per machine $\frac{1}{m}\sum_{j\in[n]} p_j$, say with a bound of form $p_j \leqslant \alpha\frac{1}{m}\sum_{j\in[n]} p_j$, then we can bound $M \leqslant (1+\alpha)\frac{1}{m}\sum_{j\in[n]} p_j$. We can then conclude as before, by the fact that the optimal makespan must be larger then the the average processing time per machine to obtain a $(1+\alpha)$-approximation algorithm.
How could we obtain a bound of type $p_j \leqslant \alpha\frac{1}{m}\sum_{j\in[n]} p_j$, for small α ?
For $\alpha = \frac{1}{k}$ with $k \in \mathbb{N}$, if $p_j \leqslant \frac{1}{km}\sum_{j\in[n]} p_j$, then job j must have processing time much below the average. We can artificially make sure that such a job gets sorted last by partitioning jobs into short jobs $S = \left\{ j : p_j \leqslant \frac{1}{km}\sum_{j\in[n]} p_j \right\}$ and long ones $L = \left\{ j : p_j > \frac{1}{km}\sum_{j\in[n]} p_j \right\}$ and doing the following. We first schedule the long jobs in some way and then we add all short jobs to a machine and apply the local search algorithm we described in the previous paragraphs. If the last job to be scheduled is a short job, we obtain the desired conclusion. Otherwise, the last job scheduled is a long job. To handle this case, note that we can't achieve a lower makespan with only the long jobs then we can with all the jobs (in a schedule for all jobs, ignore the short jobs: this lowers the makespan and produces a schedule for long jobs). So in the case that the last job scheduled is a long job, if the makespan of this instance was lower then the best achievable one for long jobs only, we could conclude that this makespan is optimal for the schedule over all jobs. Since the local search algorithm only lowers the makespan, we can guarantee this if the way we scheduled the long jobs is an optimal one. Luckily for us, there can't be too many long jobs: $|L| \leqslant km$ (as otherwise, the total processing time of the long jobs exceeds that of all jobs). There are therefore at most m^{km} schedules that can be produced (one choice of machine per long job) and we can find the best one by brute force comparison.
In conclusion, this produces an approximation scheme for this job scheduling problem for a constant number of machines. If the number of machines m is considered as variable input data, this algorithm doesn't run in polynomial time, due to its first phase. Finally, even if we consider the number of machines constant, this isn't a fully polynomial approximation scheme, as the runtime is exponential in k.

Randomized approximation algorithms:
Using random number generators, we can create randomised algorithms. These algorithms use random

number generators to get (pseudo-)random numbers that are used in the algorithm. The output of the algorithm is therefor random as well. To analyse such algorithms, we make assumption on the distribution of the random number generators and compute the expected value of the output. In particular, we're interested in:

Definition:
An **randomised approximation algorithm** for an optimization problem with attainable optimal value $z_{opt} > 0$ is a polynomial time algorithm that returns a solution that is feasible for the problem with a certain probability greater then p and that has expected objective value $E(z_{approx})$, such that there is a constant or function of the input data $\alpha > 0$, so that $1 \geqslant \dfrac{E(z_{approx})}{z_{opt}} \geqslant \alpha$ for maximisation problems and $1 \leqslant \dfrac{E(z_{approx})}{z_{opt}} \leqslant \alpha$ for minimisation problems.

We want both α and p to be simultaneously close to 1.

Here's a basic first example, in which $p = 1$, so that all output solutions are feasible for the problem:
Randomised Max-cut:
In the max-cut problem, we seek a partition of the vertices V of a graph into sets A and B so that the weight for the cut $\displaystyle\sum_{u\in A, v\in B, (u,v)\in E} w_{uv}$ is maximum, for given edge weights w_{uv}.
It turns out that choosing the partition at random yields a descent approximation.
For each node we include it in the A or B at random, with probability $\dfrac{1}{2}$ each, indipendently for all nodes.
If X_{uv} is the random variable indication if (u, v) is in the cut and $W = \displaystyle\sum_{e\in E} w_e X_e$ the random variable representing the weight, the the expected weight is $E(W) = \displaystyle\sum_{e\in E} w_e \left(\frac{1}{2}\right)$ because the probability that (u, v) is in the cut is that of $(u \in A \cap v \in B) \cup (v \in A \cap u \in B)$, which is $\dfrac{1}{2}$.
The maximum cut of value OPT must have $OPT \leqslant \displaystyle\sum_{e\in E} w_e$ so that $2E(W) \geqslant OPT$. Since in this algorithm we use a random number generator $|V|$ times, this is a linear time algorithm. So this is a randomised 2-approximation algorithm.
Ex.Max-k-cut: In the Max-k-cut problem, we partition the vertices into $k \leqslant |V|$ sets and consider the total weight of the edges with endpoints in different partition sets: the goal is to find the partition maximising this sum. Find a randomised approximation algorithm for this problem.

3.5 Parallel/distributed algorithms

We consider the approach of solving hard problems with brute force or with known efficient exponential time algorithms by increasing computational power. The problem here is that not everyone who needs to perform these computations has a super-computer at disposal. An idea is to use multiple machines to do the job. Users can rent their computers computing power for periods in which they don't require it for themselves. The problem is then the following: how do we efficiently share the computational task ?
A parallel algorithm for a problem is a algorithm that splits the problem into multiple sub-problems, where each subproblem is to be solved independently of the others, using no or little communication between machines, and finally merges the result of the subproblems to find an exact or approximate solution to the initial problem. Generally, such algorithms are based on a divide and conquer heuristic.

A toy example for the Knapsack problem:
For illustrative purposes, we'll give a very basic example for the knapsack problem. We have k machines at disposal, and assume that for n items i of weight w_i and value v_i, and capacity W, the weights are sufficiently small, $w_i \leqslant \frac{W}{k}$, and can be split evenly among machines, in the sense that $k|n$.
We split the items among the k machines in some way, be partitioning $[n]$ into the S_j for $j \in [k]$, where $|S_j| = k$. Machine j will be told the items of S_j, they're weights and values, as well as W. It will then solve the knapsack problem for the items of S_j with capacity $\frac{W}{k}$ exactly, say with the dynamic programming algorithm in time $O\left(\frac{nW}{k}\right)$. It returns solution set $S_j^* \subseteq S_j$ to us, and our solution will be $S^* = \cup_{j\in[k]} S_j^*$, which is feasible since $\sum_{j\in[k]} \sum_{i\in S_j^*} w_i \leqslant \sum_{j\in[k]} \frac{W}{k} = W$.
We'll now estimate the value of S^*, when compared to the optimal value of the problem. For an optimal set of items S^o, we note that $v(S_j) \geqslant v\left(S_j \cap S^o\right)$ (assuming positive values) and that $v\left(S_j^*\right) \geqslant \max_{i\in S_j}(v_i) \geqslant \frac{1}{k} v(S_j)$ since the item attaining maximum value in S_j is feasible for the subproblem when taken on its own, since for all $w_i \leqslant \frac{W}{k}$, and $v\left(S_j^*\right)$ is optimal for the sub-problem, and the lasi inequality is max-over-mean. Therefore, $v\left(S_j^*\right) \geqslant \frac{1}{k} v\left(S_j \cap S^o\right)$, so that $v\left(S^*\right) \geqslant \frac{1}{k} v\left(S^o\right)$ by summing over the partition sets. We see that our algorithm is a $\frac{1}{k}$-approximation algorithm.
Parallelisation can now be seen from the fact that the total runtime of the algorithm is $O\left(\frac{nW}{k}\right)$. This is because the machines solved their subproblem in parallel. Only if we performed this algorithm sequentially on a single machine would the runtime have been $O\left(k\frac{nW}{k}\right) = O(nW)$. Note also that in this example, no communication between machines was necessary. We can therefore get a solution to the knapsack problem that is half as good as the optimal one, in half the time, for example.

3.6 Solutions

Ex.PsP:
If we can solve for weights in $\mathbb{Z}$ we can in particular solve for weights in $\mathbb{N} \subset \mathbb{Z}$.
For the converse, notice that for $N(J) : \{i \in j : b_i < 0\}$, we can rewrite $\sum_{i\in I} b_i = \sum_{i\in[n]\setminus I} b_i$ as $\sum_{i\in I\setminus N(I)} b_i - \sum_{i\in N([n]\setminus I)} b_i = \sum_{i\in([n]\setminus I)\setminus N([n]\setminus I)} b_i - \sum_{i\in\setminus N(I)} b_i$, where all terms are positive. So the existence of a partition for general signs implies the existence of one for the problem on the weights $|b_i|$: we set it as $J = (I\setminus N(I)) \cup N([n]\setminus I)$. Conversely, If we solve the problem on $|b_i|$ by $\sum_{i\in I} |b_i| = \sum_{i\in[n]\setminus I} |b_i|$, then with the same trick we get $\sum_{i\in I\setminus N(I)} |b_i| - \sum_{i\in N([n]\setminus I)} |b_i| = \sum_{i\in([n]\setminus I)\setminus N([n]\setminus I)} |b_i| - \sum_{i\in\setminus N(I)} |b_i|$ which after distributing the sign and noticing that $b_i = -|b_i|$ for $i \in N([n])$, means $\sum_{i\in J} b_i = \sum_{i\notin J} b_i$.

Ex.HypPath:
We can check that a sequence of vertices is a S-F-hyperpath in polynomial time by checking $T_{e_1} \subseteq S$, $T_{e_j} \subseteq S \cup_{i<j} H_{e_i}$ for $j \leqslant n$ and $F \subseteq S \cup_{i\leqslant n} H_{e_i}$ in time $O(n|V|)$, and checking minimality with a bit more work. We'll show that the hyperpath is minimal if the deletion of a single hyperedge breaks the superpath property. So we can check the superpath property for n paths obtained by deletion of an edge, leading to a S-F-hyperpath check in $O\left(n^2|V|\right)$.
If a subset of indices $J \subsetneq [n]$ formed an S-F-superpath, the the edges of $[n]\setminus \max([n]\setminus J)$ would form an S-F-superpath, where the required properties of hyperedges stem from the nature of $(e_i)_{i\in[n]}$ or from that of $(e_i)_{i\in J}$.
Now, we reduce the problem to a known NP-complete one.
We reduce the problem to minimum set cover, which is NP-complete. In a minimum set cover instance, we are given a set A and a collection of m subsets $C_i \subseteq A$ (such that $A \subseteq \cup_{i\leqslant m} C_i$) and the task is to decide if there are sets C_i from an index set $I \subseteq [m]$ such that $|I| \leqslant k$, so that $A \subseteq \cup_{i\in I} C_i$. We start by building a dihypergraph with vertices $[m] \cup A$ and edges $(\{i\}, C_i)$. Next, we introduce a source s and a target t. We connect s to $[m]$ with edges of form (s, i). For the reaching of target t to correspond to a cover of A we can make use of the requirement that all tail-vertices of a hyperarc of a hyperpath must be visited before the hyperarc may be used. By adding hyperedge $(A, \{t\})$, an s-t-hyperpath must have visited all of A before using the final edge, which is necessarily $(A, \{t\})$. It starts with some edges of form (s, i), then uses edges of form (i, C_i) for $i \in I$ for some $I \subseteq [m]$, and finishes in $(A, \{t\})$. By inclusion minimality, only edges (s, i) for $i \in I$ are present, as they are necessary for (i, C_i), and can be deleted without affecting the hyperpath property, if (i, C_i) isn't used.
Then I is a cover, $A \subseteq \cup_{i\in I} C_i$, as $A = T_{(A,\{t\})} \subseteq S \cup_{i\leqslant n} H_{e_i}$ so that the e_i of form $(\{i\}, C_i)$, who's union is the only part of $S \cup_{i\leqslant n} H_{e_i}$ that can contain A, have $A \subseteq \cup_{i\in I} H_{e_i}$. If we give edges (s, i) weight 1 and all other edges weight 0, the weight of the hyperpath is $|I|$. So finding a hyperpath with $|I| \leqslant k$ provides a set cover.
Conversely, if I is a set cover, then $((s, i))_{i\in I}, ((i, C_i))_{i\in I}, (A, \{t\})$ is a s-t-hyperpath of weight $|I|$, as edge $(A, \{t\})$ may be used since $A \subseteq \cup_{i\in I} C_i \subseteq \{s\} \cup I \cup_{i\in I} C_i$.

Ex.SAT2:
We start by picking a variable in the first clause, say x_1, and count the number of times it appears as x_1 and $\neg x_1$ in the clauses, starting from clauses 1 to m. Once we encounter a clause that contains neither x_1 or $\neg x_1$, we pick a variable x_2 in it, and count the appearances of x_2 and $\neg x_2$ in the clauses in which x_1 or $\neg x_1$ don't appear. In general, if we encounter a clause that doesn't contain literals x_1 to x_k, we select

a variable x_{k+1} in it and count the appearances of x_{k+1} and $\neg x_{k+1}$ in the clauses in which none of the literals x_1 to x_k appear. We stop when all m clauses have been considered. So we end up with at most m literals x_1 to x_m, in m considerations.
Finally, we set $x_i = true$ if the number of x_i is greater then that of $\neg x_i$, and $x_i = false$ otherwise, and output this truth assignment. Since we made at most m comparisons, the total runtime is $O(m)$, aka linear time.
If we partition the clauses into those C_1 having literal x_1, those C_2 not having x_1 but having x_2, those C_3 not having x_1 and x_2 but having x_3, etc., then our outputted truth assignment has at least half of the clauses of C_1 true, at least half of the clauses of C_2 true, etc. This is because the clauses of C_i contain either x_i or $\neg x_i$ and we chose the truth assignment of x_i so that the majority (which is always more then half in a dichotomy) is true. Thus, the total number of true clauses is $\geqslant \sum_{i \leqslant m} \frac{|C_i|}{2} = \frac{m}{2}$, as desired.

Ex:SteinerKcut:
To see that the output is a k-cut, recall the property of Gomory-Hu tree T, that the cut associated by deleting e is represented in G by the vertices of the components of $T \backslash e$. At each iteration, a component C of the tree is split into disjoint halves C_1 and C_2 by deleting an edge in them, so that both halves contain terminals, for otherwise the edge deletion wouldn't have separated terminals. The edges of the cuts represented by C_1 can be partitioned into those leading to C_2 and those leading into $V \backslash C$, the latter of which are already in the union of the previous cuts, since C was obtained by a previous split. So only the edges between C_1 and C_2 are added to the union of cuts and they separate C_1 and C_2 in C. Therefore we get a k-cut in the end.
We consider an optimal k-cut with edges A and components V_i for $i \in [k]$. Note that the edges of A are all part of exactly two cuts represented by two V_i, corresponding to the components of the edges endpoints, conversely, all edges A_i of a cut represented by V_i are part of A, as they lead to some vertex of some V_j. Therefore $2w(A) = \sum_{i \in [k]} w(A_i)$ from double-counting. The point of considering the A_i is that we can relate them to the Gomory-Hu tree. To do this, we will consider arbitrary terminals $t_1, ...t_k$ of $V_1, ..., V_k$, consider paths from t_i to t_k for $i \leqslant k-1$ in the Gomory-Hu tree, and pick edges e_i on these paths that are the first to have endpoints in two partition sets (one of them being V_i, as we start the path in V_i). If we write $e_i = (a_i, b_i)$ for $a_i \in V_i$, and if we notice that the edges of A_i form in particular a a_i-b_i-cut ($a_i \in V_i$ and $b_i \notin V_i$), then by definition of a Gomory-Hu tree, $w(A_i) \geqslant c(a_i, b_i) = c(e_i)$ where c are the Gomory-Hu weights, aka. the values of minimum cuts of the adjacent vertices.
Next, we will relate these edges to those we selected from the Gomory-Hu tree by our algorithm. We'll show that for the ith edge f_i selected in our algorithm, we have $c(f_i) \leqslant \max_{j \leqslant i}(c(e_j))$. First, note that the e_i are different from one another: otherwise, if $e_i = e_j$, the paths from t_i and t_j to t_k would meet, latest at an endpoint of $e_i = e_j$, but by definition of our paths, this would mean that the endpoint is inparition sets V_i and V_j which are supposed to be disjoint. Now, if we had $c(f_i) > \max_{j \leqslant i}(c(e_j))$, then after having removed the $i-1$ edges of $f_1, ..., f_{i-1}$ from the tree, there must be an edge among the i ones $e_1, ..., e_i$ that is different, say $e_j \neq f_q$ for all $q \in [i-1]$. Now since $c(f_i) > \max_{j \leqslant i}(c(e_j)) \geqslant c(e_j)$, we contradict our choice of f_i as a minimiser if e_j separates two previously unseperated terminals. To see that this is indeed the case, note that removing the $e_1, ..., e_i$ separates the $t_{j \leqslant i}$ from t_k. But if two t_j and t_q where still connected, since this path, together with those to t_k would contain a cycle, which is impossible, this path has to split into two disjoint subpaths that are also subpaths of the paths to t_k. EXPLAIN MORE. This will lead to a contradiction when we consider the common ancestor to paths to t_k, which will be in the supposedly disjoint V_j and V_q. In conclusion, removing the $e_1, ..., e_i$ creates $i+1$ components each with a terminal, so that.... god damn this paper and this proof.

Ex:KnapsackDensity:
In the fractional setting, we can consider the value density of the item $d_i = \frac{v_i}{w_i}$, where v_i is the value and w_i the weight of object i. If we take mass x_i of item i, we get value $x_i d_i$. To get the greatest value, we can order the densities so as to have $d_1 \geqslant ... \geqslant d_n$, take the most mass of 1, then the most of 2, etc. until the knapsack is full. This is optimal because taking mass from any other item leaves us worse of then taking the same mass from an available better item.
To get an approximation algorithm for the general knapsack problem, we take the items in decreasing order of density d until the knapsack is full, say at item j, so that $\sum_{i\in[j]} w_i \leqslant W$ and $w_{j+1} + \sum_{i\in[j]} w_i > W$. To get an approximation algorithm, we seek comparisons with an optimal choice of items, which we'll denote by S. We'll compare $v_{j+1} + \sum_{i\in[j]} v_i$ to $\sum_{i\in S} v_i$. This is equivalent to comparing $\sum_{i\in[j+1]\backslash S} v_i$ and $\sum_{i\in S\backslash[j+1]} v_i$ (we exclude common items).
We have $d_i \geqslant d_k$ for $i \in [j+1]\backslash S$ and $k \in S\backslash[j+1]$, and also $\sum_{i\in S} w_i \leqslant W$. Now with $\sum_{i\in[j+1]\backslash S} v_i = \sum_{i\in[j+1]\backslash S} d_i w_i \geqslant d_j \left(\sum_{i\in[j+1]\backslash S} w_i\right) > d_j \left(W - \sum_{i\in S\cap[j+1]} w_i\right) \geqslant d_j \left(\sum_{i\in S\backslash[j+1])} w_i\right) = \sum_{i\in S\backslash[j+1]} v_i$, we seen that $v_{j+1} + \sum_{i\in[j]} v_i \geqslant OPT$. This implies that one of v_{j+1} or $\sum_{i\in[j]} v_i$ must be greater then $\frac{OPT}{2}$ (otherwise, summing the two resulting inequalities contradicts $v_{j+1} + \sum_{i\in[j]} v_i \geqslant OPT$), so taking the maximium among v_{j+1} or $\sum_{i\in[j]} v_i$ yields a $\frac{1}{2}$- approximation algorithm for knapsack.

Ex.Max-k-cut:
We attribute the vertices to one of the partition sets independently with uniform probability of $\frac{1}{k}$. The probability of a given edge then being in the cut is that of having its vertices in different partition sets. The opposite of the latter event is easier to find: for a given partition set, the probability that both endpoints are in it is $\frac{1}{k^2}$, and summing over the different sets yields $\sum_{i=1}^{k} \frac{1}{k^2} = \frac{1}{k}$. So the probability that the edge is in the cut is $1 - \frac{1}{k}$, so that the expected weight of the cut is $1 - \frac{1}{k}$ times the total weight over all edges, which is less then $1 - \frac{1}{k}$ times the value of the maximum cut. Therefore we get a $\left(1 - \frac{1}{k}\right)$-approximation algorithm for max-k-cut.

4 Parametrized and exact exponential algorithms

With "param..." book in Algo folder

Parametrized algorithms can be described as efficient algorithms for the decision version of an optimization problem, in the sense that runtime is polynomial in the instance data, but may be exponential in the objective value (the "parameter") data. We present some techniques for producing such algorithms.

4.1 Kernelisation

In kernelisation methods, we reduce the problem with efficient reductions to a smaller one, the "kernel", which we then solve exactly with exact exponential algorithms. One can think of this as preprocessing the instance. We give a first example for the vertex cover problem.

We'd like to know if a graph G contains a vertex cover (a set of vertices such that each edge has at least one of its endpoints in that set) of size at most k. A first easy preprocessing step would be to delete isolated vertices from the instance, as no edges are incident to them so that having them in a vertex cover is useless. Formally, we create a subproblem of instance (G, k) in the form of $(G\backslash\{v : \deg_G(v) = 0\}, k)$, which we then seek to solve. If it's a positive instance with cover S, then S is also a cover of G, since the edges are the same in both graphs. If it isn't, then (G, k) must be a negative instance too, as for any cover S of G of size at most k, so is $S\backslash\{v : \deg_G(v) = 0\}$ in $G\backslash\{v : \deg_G(v) = 0\}$.
The next preprocessing step is based on a necessary condition on positive instances of the problem: if S is a vertex cover of G of size at most k and $v \in V$ has $\deg_G(v) \geqslant k+1$, then v must be in S. Otherwise, the edges of $\delta(v)$ would have to be covered by the other endpoint, which accounts for $k+1$ vertices in S, which we assumed to have size at most k. Thus, the vertices of degree $\deg_G(v) \geqslant k+1$ will have to be in a a vertex cover of G of size at most k, if there is one. Since vertex covers are stable under vertex deletion, in the sense that if S is a vertex cover of G, then $S\backslash v$ is one of $G\backslash v$ (indeed, an uncovered edge would have had to be incident to v), we can make the following reduction. Delete all vertices of degree $\deg_G(v) \geqslant k+1$ and solve the instance on $G\backslash\{v : \deg_G(v) \geqslant k+1\}$ with parameter $k' = k - |\{v : \deg(v) \geqslant k+1\}|$: if the subproblem is a positive instance with cover S', then $S = S' \cup \{v : \deg_G(v) \geqslant k+1\}$ is a set cover of size at most k of G, and if the subproblem is a negative instance, then so must the instance (G, k) bee as well, since for a set cover S of size at most k of G, $S' = S\backslash\{v : \deg_G(v) \geqslant k+1\}$ would be a positive insance on $(G\backslash\{v : \deg_G(v) \geqslant k+1\}, k - |\{v : \deg_G(v) \geqslant k+1\}|)$.

Recursively applying these reductions when they apply, we get a sequence of sub-problems (G_i, k_i) for subgraphs G_i or decreasing k_i. Producing the instances is done in polynomial time, as we have to check for at most $|V|$ degrees each time, and so is augementing positive instances. The question is what happens if none of the cases needed for these reductions apply, that is, G_i has no isolated vertices nor vertices of degree more then k_i. In that case, we would decide the problem with an exact exponential algorithm. For example, one can use brute force of trying all vertex sets and checking if all edges are covered, in time $2^{|V_i|}|E_i|$. We will now try to bound $|V_i|$, $|E_i|$ and the number of reductions we can make. It turns out that analysis will give us a criterion that avoids the brute force finish in some cases.

Lemma:

If (G_i, k_i) is a positive instance, and the reductions don't apply, then $|V_i| \leqslant k^2 + k$ and $|E_i| \leqslant k^2$.

With this lemma, once the previous reductions don't apply, we can check if $|V_i| > k^2 + k$ or $|E_i| > k^2$, in which case we'll know that the instance can't be positive, so the the previous and the initial instance

(G,k) can't be either. If however $|V_i| \leqslant k^2+k$ and $|E_i| \leqslant k^2$, we will have to use the exact exponential algorithm on the last instance to take our decision. For the brute force appraoch, we'll then have bound $O\left(2^{k^2+k}k^2\right)$ on its runtime. Since we perform at most $|V|$ reductions since each reduction deletes vertices (making $|V|$ inicidence checks each time), the runtime is $O\left(2^{k^2+k}k^2|V|^2\right)$. It is polynomial in the graph data, but exponential in the parameter k: this is our first example of a parametrized algorithm.

Proof of the lemma: If (G_i,k_i) is a positive instance with cover S and no isolated vertices or vertices of degree $\geqslant k_i+1$, then we use the following remarks. The vertices of $V_i\backslash S$ must be adjacent to one of S, since they're not isolated, and an edge they're incident with is covered by S. Now, any vertex of S can have at most k_i neighbours, so that there are at most k_i vertices of $V_i\backslash S$ per vertex of S, hence $|V_i\backslash S| \leqslant k_i|S|$, so that $|V_i| \leqslant (k_i+1)|S| \leqslant k^2+k$ as $|S| \leqslant k_i$.
For the edge E_i, each edge is incident to a vertex of S, each of which can have at most k_i incident edges, so that $|E_i| \leqslant k_i|S| \leqslant k_i^2$.

Before moving to the next example, we note that one can easily perform an input specification for hard problems, given a parameterized algorithm. For example, we can consider the class of graphs admitting a vertex cover of size at most $p\sqrt{\ln_2(|V|))}$ for some fixed $p>0$. On that class, we can run our algorithm with $k=p\sqrt{\ln_2(|V|))}$, in time $O\left(|V|^{2p}\ln_2(|V|^{2p}))|V|^2\right)$, and find a small cover. In fact, we can perform binary search on $k \leq p\sqrt{\ln_2(|V|))}$, where each query takes time $O\left(|V|^{2p}\ln_2(|V|^{2p}))|V|^2\right)$, to find a minimum vertex cover in time $O\left(\ln_2(p\sqrt{\ln_2(|V|))})|V|^{2p}\ln_2(|V|^{2p}))|V|^2\right)$. So on this class of graphs with small vertex cover, the minimum vertex cover can be found in polynomial time.

d-Hitting set with sunflowers
In the d-hitting set probelm, we're given a family $F \subseteq 2^U$ of classes (subsets) of an universe U, each of size at most d, and our goal in the decision version of the problem is to find a set $H \subseteq U$ of representatives, in the sense that for all classes $A \in F$, it's represented by $A \cap H \neq \emptyset$, so that $|H| \leqslant k$.

An interesting notion from combinatorics related to this problem is that of a **sunflower**. The sets $S_1,...,S_q \in F$ form a sunflower of size q if for any $i \neq j$, they have the same intersection $Y = S_i \cap S_j$, the **core**, which may be empty, and non-empty **petals** $S_i\backslash Y$ (which are disjoint). This notion allows us to make an observation similar to that in the previous example for vertex cover. Indeed, if F has a sunflower of size $k+1$ as well as a hitting set H of size at most k, then H must contain some element of the core Y of the sunflower, which must be non-empty. Otherwise, the representatives of the S_i lie in the disjoint petals $S_i\backslash Y$, so that $k+1$ are needed, contradicting $|H| \leqslant k$.

If we have an efficient way to detect sunflowers of size at least $k+1$, then we can make the following observation. If we find one with empty core, there are at least $k+1$ disjoint petals of F so that no hitting set can have size less then k. If it has non-empty core, we know that a potential hitting set of size less then k must contain an element of the core. In fact any element does the job, as all S_i contain the whole core. We can therefore reduce instance (F,k) to $((F\backslash\{S_1,...,S_q\}) \cup Y, k)$ (with $q \geqslant k+1$), replacing the sunflower by its core. Indeed, any hitting set of $(F\backslash\{S_1,...,S_q\}) \cup Y$ will hit Y, hence all S_i, so that it's a hitting set for F too. Conversely, if $(F\backslash\{S_1,...,S_q\}) \cup Y$ has no hitting set of size less then k, then nor can F have, as otherwise, we just observe that such a hitting set H would have to hit Y, so that H would also be a hitting set of $(F\backslash\{S_1,...,S_q\}) \cup Y$.

The question now is how to find such sunflowers, and what can be said about a reduced instances size when no such sunflowers can be found. We will make use of:

Sunflower lemma (Erdös-Rado):

Given a family $F \subseteq 2^U$ of subsets of an universe U, each of size at most d, then if $|F| > d!k^d$, it contains a sunflower with $k+1$ petals, that we can find in time polynomial in $|F|, |U|, k$ and d.

Proof: Let's start by looking for disjoint sets, as they form sunflowers with empty core. We find a augmentation maximal family $S_1, ..., S_q$ of disjoint sets of F as follows: start with some set $S_1$of F, and iterate over the $|F|$ sets of F, augmenting the current family $S_1, ..., S_l$ by S_{l+1} if it's disjoint from the previous sets, which we can check in time $d|F|$. This output family will be maximal in the sense that no other set of F can be disjoint from all $S_1, ..., S_q$, since this would contradict the fact we didn't add such a set to the family when it was considered by our algorithm.
If $q \geqslant k+1$, we found our sunflower. The question is how to recycle the case $q \leqslant k$ into something usefull. We consider $S = \cup_{i\in[q]} S_i$, which has size $|S| \leqslant kd$ (as $q \leqslant k$ and $|S_i| \leqslant d$), and investigate the fact that for all $A \in F$, we have $A \cap S \neq \emptyset$.
The idea is that we can expect an element that is is a lot of sets to be in the core of some sunflower with many petals. For each $A \in F$, we pick som $s \in S$ such that $s \in A \cap S$, and increase its "number of representations" n_s by 1, staring from $n_s = 0$. We then consider the s with the largest n_s. This is done in $2|F|$ time. Since $|F| = \sum_{s'\in S} n_{s'}$, we'll have $n_s \geqslant \frac{|F|}{|S|} \geqslant \frac{|F|}{dk}$ by max-over-average. We can then look for a sunflower in the family $F' = \{A \backslash s : A \in F, s \in A\}$, based on the intuition we mentioned. Note that $|F'| = n_s$. If $S'_1, ..., S'_{k+1}$ is a sunflower of F', then $S_1 = S'_1 \cup s, ..., S_{k+1} = S'_{k+1} \cup s$ is one of F. We will thus work recursively on family F', which will have elements of size at most $d-1$. The base case is that of singletons where $d = 1$, in which all collections of disjoint singletons are sunflowers.
The problem is that our recursion fails when a sub-case has no sunflower of size $k+1$. We will seek conditions on F such that this case never occurs. Following the idea that a large family should contain a large sunflower, we'll try to prove that a lower bound on $|F|$ (which we'll later find to be $d!k^d$) guarantees the existence of a large sunflower. To make this compatible with our recursion, the lower bound should be stable under $|F'| \geqslant \frac{|F|}{dk}$. This is the case of $d!k^d$, as then $|F'| \geqslant \frac{|F|}{dk} > \frac{d!k^d}{dk} = (d-1)!k^{d-1}$.
We therefore show that when $|F| > d!k^d$ for a family $F \subseteq 2^U$ of subsets of an universe U, each of size at most d, by induction on d. The step is precisely the one we developed. We find s, build F', note that $|F'| > (d-1)!k^{d-1}$, so that the induction hypothesis applies and we conclude with the existence of sunflower $S'_1, ..., S'_{k+1}$ of F', which we can augment to sunflower $S_1 = S'_1 \cup s, ..., S_{k+1} = S'_{k+1} \cup s$ of F.
The key question is if the initial step holds, for our bound $d!k^d$. Fortunately, if for $d = 1$, F is a family of singletons with $|F| > k$, we can take $k+1$ different singletons, which will form a sunflower. So when $|F| > d!k^d$, our recursion will indeed produce a sunflower, since the base case contains one. There are at most d recursions, since this parameter decreases each time. So this algorithm is $O\left(d^2|F|^2\right)$.

We now return to the kernelisation of the d-hitting set problem. At each step, we check if for the current family F, we have $|F| > d!k^d$. In that case, we run the previously described algorithm to find a sunflower of size $k+1$, which we use to make the reduction, moving to the next iteration. Since at each iteration, $|(F\backslash\{S_1, ..., S_{k+1}\}) \cup \{Y\}| = |F| - k$, this case can occur at most $\frac{|F| - d!k^d}{k}$ times, for the initial family F.
Once the sub-case family has size less then $d!k^d$, we have to solve the instance with an exact exponential algorithm. The brute force approach of testing all possible hitting sets H, $O(2^{d!k^d})$ of which there are, each in time $|F|$, will lead the main algorithm to run in time $O\left(\left(\frac{|F| - d!k^d}{k}\right) d^2|F|^2|F|2^{d!k^d}\right)$. Note that this time is polynomial in $|F|$, and that we are using two parameters d and k.

Max-SAT with crown decomposition

Feedback vertex set with the expantion lemma

4.3 Iterative compression

4.4 Tree decompositions and treewidth

4.5 Subset convolution and applications

Subset convolutions originate from combinatorics. We can phrase certain algorithmic problems in terms of computing such a convolution. The name is due to their similarity with the convolutions from Fourier analysis: we'll show that there are transforms that simplify their computation.

First, let's give an example of a problem that can be expressed in terms of such convolution. We consider the k-coloring problem in which we ask if a graph G has a proper coloring of at most k colours. This is equivalent to finding a partition of V into at most k independet sets.
The first step towards a computational reformulation is obtained by introducing indicator $s(X) = \begin{cases} 1 : X \ independent \\ 0 : \ else \end{cases}$ so that the fact that the number of proper colorings with at most k colours is counted by $\sum_{\substack{X_1,...,X_k \subseteq V \\ X_i \cap X_j = \emptyset : i \neq j \\ \cup_{i\in[k]} X_i = V}} \prod_{i\in[k]} s(X_i)$. Indeed, $\prod_{i\in[k]} s(X_i) = 1$ precisely when the X_i form a proper colorings with at most k colours (recall that we allow the X_i to be empty, which corresponds to color i not being used). This approach may more feel more motivated if one has a probability theory (or even random graph) background.
The next step is to recognize that this sum can be expressed as $\underbrace{s * s * ... * s}_{k-1 \ operations}(V)$, where $*$ is the **subset convolution** defined by $(f * g)(X) = \sum_{\substack{A \cup B = X \\ A \cap B = \emptyset}} f(A)g(B)$. Indeed, we can show this by induction.
For $k = 2$, we have $(s * s)(V) = \sum_{\substack{X_1 \cup X_2 = V \\ X_1 \cap X_2 = \emptyset}} \prod_{i\in[2]} s(X_i)$ by definition. For the step, for $f = s$ and $g = s * s * ... * s$, we have

$$(f*g)(V) = \sum_{\substack{A \cup B = V \\ A \cap B = \emptyset}} s(A) \left(\sum_{\substack{X_1,...,X_k \subseteq B \\ X_i \cap X_j = \emptyset : i \neq j \\ \cup_{i\in[k]} X_i = B}} \prod_{i\in[k]} s(X_i) \right) = \sum_{\substack{A, X_1,...,X_k \subseteq V \\ X_i \cap X_j = \emptyset : i \neq j \\ A \cup X_i = \emptyset \\ A \cup_{i\in[k]} X_i = V}} s(A) \prod_{i\in[k]} s(X_i)$$

and renaming $A = X_{k+1}$ does the job.
Hence, the goal is to compute $\underbrace{s * s * ... * s}_{k-1 \ operations}(V)$ efficiently, as brute evaluation requires computing a sum of $k^{|V|}$ terms.

We now discuss transforms of functions $f : 2^V \to \mathbb{R}$ that will simplify the convolutions similar to the Fourier transform. They can be thought of as generalizations of the inclusion exclusion principle, when phrased as a transform, similar to binomial- or Stirling-inversion from combinatorics.

Definitions:

For $f, g : 2^V \to \mathbb{R}$, we define the **subset convolution** by $(f * g)(X) = \sum_{\substack{A \cup B = X \\ A \cap B = \emptyset}} f(A)g(B)$.

We define the **cover product** by $(f *_c g)(X) = \sum_{A \cup B = X} f(A)g(B)$.

We define the **zeta-transform** as $\zeta : \begin{cases} (2^V \to \mathbb{R}) \to (2^V \to \mathbb{R}) \\ f \mapsto \zeta(f)(X) = \sum_{Y \subseteq X} f(Y) \end{cases}$.

We define the **Möbius-transform** as $\mu : \begin{cases} (2^V \to \mathbb{R}) \to (2^V \to \mathbb{R}) \\ f \mapsto \mu(f)(X) = \sum_{Y \subseteq X} (-1)^{|X|-|Y|} f(Y) \end{cases}$.

We define the **odd-negation-transform** as $\sigma : \begin{cases} (2^V \to \mathbb{R}) \to (2^V \to \mathbb{R}) \\ f \mapsto \sigma(f)(X) = (-1)^{|X|} f(X) \end{cases}$.

The link to inclusion-exclusion can be seen by letting $f(X) = -|\cap_{i \in X} A_i|$, we have reformulated inclusion-exclusion as $|\cup_{i \in X} A_i| = (\zeta\sigma f)([n]) + |U|$, with convention $\cap_{i \in \emptyset} = U$. The alternative expression of inclusion-exclusion can be expressed by letting $g(X) = |\overline{\cup_{i \in X} A_i}| = |\cap_{i \in X} \overline{A_i}|$, so that $|\cap_{i \in [n]} A_i| = (\sigma\mu g)([n])$. By writing $g([n]) = |U| - |\cup_{i \in [n]} A_i| = -(\zeta\sigma f)([n])$ and recalling $f(X) = -|\cap_{i \in X} A_i|$, we see that $\begin{cases} g([n]) = -(\zeta\sigma f)([n]) \\ f([n]) = -(\sigma\mu g)([n]) \end{cases}$. Linearity taking care of the negative sign, we have $(\zeta\sigma\sigma\mu)f = f$ and $(\sigma\mu\zeta\sigma)g = g$. This is actually a particular case of the following:

Transform inversion:

We have $\zeta = \sigma\mu\sigma$, $\mu = \sigma\zeta\sigma$, $\sigma\sigma = id$, $\zeta\mu = id$ and $\mu\zeta = id$.

Proof: We show the last two identities. We have $(\mu\zeta f)(X) = (\sigma\zeta\sigma\zeta f)(X) = (-1)^{|X|} \sum_{Y \subseteq X} (-1)^{|Y|} \sum_{Z \subseteq Y} f(Z)$ by definitions, which we reformulate as $(-1)^{|X|} \sum_{Z : Z \subseteq X} f(Z)(-1)^{|Z|} \sum_{Y : Z \subseteq Y \subseteq X} (-1)^{|Y \backslash Z|}$. The last sum is of form $\sum_{A : A \subseteq B} (-1)^{|A|}$ for $A = Y \backslash Z$ and $B = X$, which turns out to be 0, as we'll soon show, when $B \neq \emptyset$.

So we rewrite $(\mu\zeta f)(X) = (-1)^{|X|} \left(f(X)(-1)^{|X|} 1 + \sum_{Z : Z \subsetneq X} f(Z)(-1)^{|Z|} \sum_{Y : Z \subseteq Y \subseteq X} (-1)^{|Y \backslash Z|} \right)$ so that $B = X \backslash Z$ is non-empty, since $Z \subsetneq X$, leading to $(\mu\zeta f)(X) = f(X)$, as desired.

To see that $\sum_{A : A \subseteq B} (-1)^{|A|} = 0$, take a $b \in B \neq \emptyset$, and partition the subsets of B on containing b or not. There must be the same amount in both, since there is a bijection between them, given by $A \mapsto A \cup b$ and $A \mapsto A \backslash b$, that pairs up subsets A_1 and A_2 such that $A_1 \Delta A_2 = \{b\}$. Since the pairs differ only by one element, they have different parities, and cancel their corresponding terms in $\sum_{A : A \subsetneq B} (-1)^{|A|}$.

To see that $\zeta\mu = id$, we write $\zeta\mu = \sigma\mu\sigma\sigma\zeta\sigma = \sigma\mu\zeta\sigma = \sigma\sigma = id$, using the previous identities.

Transform:

We have $\zeta(f *_c g) = (\zeta f)(\zeta g)$, and therefor $f *_c g = \mu((\zeta f)(\zeta g))$.

Proof: $\zeta(f *_c g)(X) = \sum_{Y \subseteq X} \sum_{A \cup B \subseteq Y} f(A)g(B) = \sum_{A \cup B \subseteq X} f(A)g(B) = \sum_{A,B \subseteq X} f(A)g(B) = \left(\sum_{A \subseteq X} f(A)\right)\left(\sum_{B \subseteq X} g(B)\right) = (\zeta f)(X)(\zeta g)(X)$.

Convolution and cover:

By letting $f_k(X) = \begin{cases} f(X) : |X| = k \\ 0 : else \end{cases}$, we can write $(f * g)(X) = \sum_{i=0}^{|X|} (f_i *_c g_{|X|-i})(X)$.

Proof: $(f * g)(X) = \sum_{\substack{A \cup B = X \\ A \cap B = \emptyset}} f(A)g(B) = \sum_{\substack{A \cup B = X \\ |A| + |B| = |X|}} f(A)g(B)$ since $A \cup B = X$ and $|A| + |B| = |X|$ imply $A \cap B = \emptyset$, as $|A \cap B| + |X| = |A \cap B| + |A \cup B| = |A| + |B| = |X|$ so that $|A \cap B| = 0$. Next, we partition on sizes with $(f * g)(X) = \sum_{i=0}^{|X|} \sum_{\substack{A \cup B = X \\ |A| = i, |B| = |X| - i}} f(A)g(B)$, and relax to $(f * g)(X) = \sum_{i=0}^{|X|} \sum_{A \cup B = X} f_i(A) g_{|X|-i}(B)$, which is $\sum_{i=0}^{|X|} (f_i *_c g_{|X|-i})(X)$, as when $|A| = i, |B| = |X| - i$ doesn't hold, the term is 0.

We now have our pipeline for computing the subset convolution using transforms. We compute the $V|$ values $(f_i *_c g_{|X|-i})(X)$ for all $i \in [|X|]$ with identity $f *_c g = \mu((\zeta f)(\zeta g))$, so that 3 transforms have to be performed each time. We then conclude with $(f * g)(X) = \sum_{i=0}^{|X|} (f_i *_c g_{|X|-i})(X)$. The whole point of this is of course to gain time over the actual computation of the convolution. The latter requires enumerating all bipartitions of X. This can be done by choosing some $x \in X \neq \emptyset$, and letting A range over the subsets of $X \backslash x$, letting $B = X \backslash A$. Indeed, in any biaprtition of X, x is in exactly one of A or B, so that this case is enumerated with our method. Hence, the convolution is a sum of $2^{|X|-1}$ terms.
The subtlety is that for our coloring problem, we need to compute iterated convolutions. There, for computing the next convolution, we need to have the values of the previous convolution for all $X \subseteq V$. The brute computation of all convolution values would thus require $\sum_{X \subseteq V} 2^{|X|-1} = \frac{1}{2} \sum_{k=0}^{|V|} \binom{|V|}{k} 2^k = \frac{3^{|V|}}{2}$ terms.
We'll now show that we can do better with our transform pipeline.

Fast convolution:

We can compute all $2^{|V|}$ values $(f * g)(X)$ over $X \subseteq V$ in total time $O(2^{|V|}|V|^3)$.

Proof: The idea is similar to fast polynomial multiplication with the FFT. We will compute the values $(f_i *_c g_j)(X)$ ranging over $X \subseteq V$ with transforms and dynamic programming, in time $2^{|V|}|V|$, for fixed $i, j \in [|V|]$, so that the total process requires time $2^{|V|}|V|^3$ by ranging over i, j. We can then conclude with $2^{|V|}$ computations of the form $(f * g)(X) = \sum_{i=0}^{|X|}(f_i *_c g_{|X|-i})(X)$ with $O(|V|)$ terms, so that the runtime is still dominated by $2^{|V|}|V|^3$.
To compute the values $(f_i *_c g_j)(X)$, we'll use $(f_i *_c g_j)(X) = (\sigma\zeta\sigma)((\zeta f_i)(X)(\zeta g_j)(X))$. More presicely, we will compute the values $(\zeta f_i)(X)$, $(\zeta g_j)(X)$ and then $\zeta\left((-1)^{|.|}(\zeta f_i)(.)(\zeta g_j)(.)\right)(X)$, each time for all $X \subseteq V$ separately, in total time $3 \times 2^{|V|}|V|$, as each computation of all zeta-transform values will require time $2^{|V|}|V|$.

The idea is that since we take sums over subsets in the zeta-transfom, and different sets can have subsets in common, we don't have to recompute parts we have already implicitly computed if we organize computation well.

For the dynamic programming solution to computing the values, we will number the elelement in V from 1 to $n = |V|$, and associate to each set $X \subseteq V$ a string $x_1, ..., x_n$ where $x_i \in \{0, 1\}$ indicates if the ith element of V is in X. We will recurse from the top for computing the sums $\sum_{Y \subseteq X} f(Y)$. The last trems of the sums will be $f(X)$, which we can compute for all $X \subseteq V$. We then disjoin cases on the last element x_n. If X contains it, then it's subsets can be disjoint on whether they contain it too. In our notation, if $x_n = 1$, then the subsets of x correspond to strings of form $*, ..., *, 1$ or $*, ..., *, 0$, for $* \in \{0, 1\}$. In particular $\sum_{Y \subseteq X} f(Y)$ will contain both $f(x_1, ..., x_{n-1}, 1)$ and $f(x_1, ..., x_{n-1}, 0)$. If $x_n = 0$, then the sum only contains $f(x_1, ..., x_{n-1}, 0)$. We can then start our dynamc programming table. Picture a table with rows ndex by substes of V, represented by 0-1-strings. We will sucessively get more and more terms of the sums column by column. On the first column, we enter the values $f(X)$. We then add or pass values according the the way we discribed it for x_n, disjoining on the values of the next elements x_i. In tabular form:

$f(a_1, ..., 1, 0)$	$S_1(a_1, ..., 1, 0) = f(a_1, ..., 1, 0)$	$S_2(a_1, ..., 1, 0) = S_1(a_1, ..., 0, 0) + S_1(a_1, ..., 1, 0)$
$f(a_1, ..., 1, 1)$	$S_1(a_1, ..., 1, 1) = f(a_1, ..., 1, 0) + f(a_1, ..., 1, 1)$	$S_2(a_1, ..., 1, 1) = S_1(a_1, ..., 0, 1) + S_1(a_1, ..., 1, 1)$
$f(b_1, ..., 0, 0)$	$S_1(b_1, ..., 0, 0) = f(b_1, ..., 0, 0)$	$S_2(b_1, ..., 0, 0) = S_1(b_1, ..., 0, 0)$
$f(b_1, ..., 0, 1)$	$S_1(a_1, ..., 0, 1) = f(b_1, ..., 0, 0) + f(b_1, ..., 0, 1)$	$S_2(b_1, ..., 0, 1) = S_1(b_1, ..., 0, 1)$

This is a bloody mess.... FIX.

Coloring with convolutions:

We can check if graph G can be coloured with at most k colours in time $O(k2^{|V|}|V|^3)$.

COMPLETE: combi with treewidth.

4.6 Solutions

4.7 Solutions

5 Matroids

Accompanying Lawlers book

5.1 Definitions, examples, properties

We'll present matroids as an abstract formalism that allows to relate optimization problems with a common language. Historically however, matroids originated from linear algebra.

Many combinatorial optimization problems can be expressed as $\max\left(\sum_{i\in S} w_i : S \in \mathcal{F}\right)$ where $\mathcal{F}$ is a family of subsets of a finite set E, so $\mathcal{F} \subseteq 2^E$, that represents sets with a certain combinatorial among a larger set E. For example, in view of the MST problem, E could be the edge-set of a graph G, and $\mathcal{F}$ the family of edges inducing an acyclic graph. A particular class of such families are the following, which our example is a part of:

Independence system:

An **independence system** is (E, I) where E is a finite set and $I \subseteq 2^E$ is a family of sets that is subset-stable, in the sense that for all $A \in I$ and $B \subseteq I$, we have $B \in I$. We always have $\emptyset \in I$ by convention.

A **basis** of I is a $B \in I$ that is inclusion maximal, so that there is no $A \supsetneq B$ in I.

In our example, if the graph is connected, the bases of I are the spanning trees of the graph.
We can study the problems of finding maximum weight independent sets and of finding minimum weight bases. The MST problem is of the second kind, for the independece set of our example.
As we know, the MST is solved by a greedy algorithm, which in independece set terms would consist of sorting the elements of E by increasing weight, and constructing a base (tree) B by starting with $\emptyset$, adding elements (edges) of lowest weight not yet added to it, if the addition maintains independece (acyclicity).
To see that the outputted B is a base, assume that there is an $A \supsetneq B$ that is also in I, so that there is an $a \in A \backslash B$ and so that $(B \cup a) \in I$ since $(B \cup a) \subseteq A \in I$. If we consider the iteration in which a was considered, at which the output-candidate was $B' \subseteq B$, then we didn't include a because $(B' \cup a) \notin I$. This is a contradiction to subset-stability, as $(B' \cup a) \subseteq (B \cup a) \in I$. So the greedy algorithm does produce a base of the independence system. We'll see in the next section that it in fact provides a minimum weight base.

Now that we've expressed a greedy algorithm for independence systems, a question we can ask is: what are the independence systems for which the greedy algorithm produces an optimal output ? We can actually give a transparent property defining these independence systems.

Matroids:

A **matroid** is an independence system for which the greedy algorithm solves the optimization problem. Equivalently, it's an independence system (E, M) such that for all $A, B \in M$ with $|A| > |B|$, there is an $a \in A \backslash B$ such that $(B \cup a) \in M$ (the augmentation property).

We'll show the equivalence of both definitions in the next section.
Note that showing the augmentation property can be just as hard as directly proving the correctness of

the greedy algorithm. We now give a few examples of matroids:

The graphic/cycle/forest matroid:
It's our running example of the acyclic edge-sets of a graph. Take edge-sets $A, B \in M$ with $|A| > |B|$. We'll show that A must have an $a \in A \backslash B$ that connects two components of the forest B by contradiction. If it didn't then all edges $a \in A$ have both their endpoints in the same component of B. For such a component C, the edges of A with endpoints in C form an acyclic set, so that there can be at most $|C|-1$ of them, whereas B has exactly $|C|-1$ edges in its connected component C. Summing over components, we get $|A| \leqslant |B|$, a contradition to $|A| > |B|$. Hence an $a \in A \backslash B$ exists that connects two components of B: since $(B \cup a)$ is acyclic, the augmentation property is proved.

The partition and the uniform matroids:
We consider a disjoint partition $E = \cup_{i \in [p]} E_i$ and numbers $k_i \in \mathbb{N}$ for $i \in [p]$. The **partition matroid** is formed by $M = \{F \subseteq E : |F \cap E_i| \leqslant k_i, \forall i \in [p]\}$. An example is when E has elements of form (i, j) where i indexes a person and j an activity, and we which to match people to activities, where an activity can host multiple (possibly all) people, such that a function of the attribution choices is maximised. Then $E_i = \{(k, j) : k = i\}$ and $k_i = 1$, as a person can join at most one activity. A particular case is that of the **uniform matroid**, which is a partition matroid for which $p = 1$. They are independence systems, as $|F \cap E_i|$ can only decrease under subsets. To see the augemntation property, we take sets $A, B \in M$ with $|A| > |B|$ and show that there has to be some $i \in [p]$ such that $|A \cap E_i| > |B \cap E_i|$: indeed, as we have partition $E = \cup_{i \in [p]} E_i$, assuming the opposite, which is $|A \cap E_i| \leqslant |B \cap E_i|$ for all $i \in [p]$, leads to $|A| \leqslant |B|$ after summation, is a contraditon to $|A| > |B|$. Hence, there must be an $a \in (A \cap E_i) \backslash (B \cap E_i) \subseteq A \backslash B$, otherwise $(A \cap E_i) \subseteq (B \cap E_i)$, contradicting $|A \cap E_i| > |B \cap E_i|$. Then $(B \cup a) \in M$, for B in unchanged on the $j \neq i$, as $a \in E_i$ and the E_j are disjoint so that $|(B \cup a) \cap E_j| \leqslant |B \cap E_j| \leqslant k_j$ for $j \neq i$, and $|(B \cup a) \cap E_i| = |B \cap E_i| + 1 \leqslant |A \cap E_i| \leqslant k_i$ (sizes are integers).

The transversal and matching matroid:
We consider a population E set of g groups $X_i \subseteq E$ in the population, where individuals can belong to multiple groups. We seek a set of distinct individuals representing a group they belong to, which should be maximal and cost minimum, for individual costs. Here the system is $M = \{\{r_1, ..r_k\} : k \leqslant g; r_i \in X_{g_i}; g_i \neq g_j, \forall i \neq j\}$. It's an independence system since by taking a subset of representatives, those representatives still belong to group, each different for different representatives.As we'll soon see, it's a matroid, called the **transversal matroid**.
It's a particular case of the **matching matroid**, which is based on a bipartite graph $(U \cup V, E)$ and who's elements are vertices, unlike in the previous examples. It's (U, M) where M is the set of subsets $W \subseteq U$ that are covered by some matching E_M of the graph. To see that this is an idependece system, note that a subset of vertices covered by a matching is still covered by that matching. The relation to transverals is that we can represent the transversals by matchings in a graph. We set U to be the population, V to be the groups, and add an edge between vertices if the individual belongs to the group: then, a set covered by a matching corresponds to a set of representatives.
We'll now show the augmentation property. We consider sets $A, B \subseteq U$ covered by matchings E_A and E_B (that we assume minimal for covering these sets), so that $|A| > |B|$. To see what can be added from A, we investigate the symmetric difference $E_A \Delta E_B$. As we remember from the bipartite matching section, $E_A \Delta E_B$ forms disjoint paths and cycles, with vertices on them alternating bipartition sets, and edges alternating matching sets (alternating paths and cycles). The symetric difference can't be empty, since $|A| = |E_A|$ and $|B| = |E_B|$ and $E_A \Delta E_B = \emptyset$ implies $A = B$ which would then contradict $|A| > |B|$, and there can't be only cyles, as these cycles have to be even by bipartiteness, and they're alternating, so that $|E_A| = |E_B|$, contradicting $|A| > |B|$ again. There is therefore an alternating path in $E_A \Delta E_B$. I fact there has to be an odd one, for otherwise $|E_A| = |E_B|$ again, which leads to the same contradiction as before.

Further, there has to be an odd path with more edges from E_A, for otherwise $|E_A| < |E_B|$, contradicting $|A| > |B|$ again. This is the augmenting path P we're looking for to augment E_B. Indeed, $P \Delta E_B$ is a matching, and it covers the vertices of B, as well as an $a \in A \backslash B$ which is an endpoint of the path P, so that $B \cup a$ is independent.

Matroid intersection
Not all independence systems are matroids. This is clear from the fact that the greedy algorithm doesn't provide optimal solutions for all problems. For example, the edge sets of bipartite matchings are an independence system (subsets of matchings are matchings), but they aren't matroids, as the following example of failure of the augmentation property shows: We consider the complete bipartite $K_{2,2}$ with vertices u_1, u_2 and v_1, v_2. The matching $\{\{u_1, v_2\}, \{u_2, v_1\}\}$ is bigger then the single edge one $\{u_1, v_1\}$, yet no edge from the bigger one can be used to augment the smaller one.

However, matroids can still be used to discribe this problem ! Indeed, for a bipartite graph $(U \cup V, E)$, we can consider the partition matroids given by $\cup_{v \in U} \delta(v)$ and $k_v = 1$, and $\cup_{v \in V} \delta(v)$ and $k_v = 1$. A matching is independet in both matroids, and any edge set in both matroids is a matching. It turns out that many other combinatorial optimization problems can be considered as problems on intersections of matroids. Here are a few examples:

Branchings:
In a digaph (V, A), we can look for sets of edges that induce a partial order on vertices. This order is given by $u > v$ if there is a directed path from u to v. If (V, A) is a communication network, then we're interesetd in choosing a node where to pass a message from, so that the message will be propagated as much as possible. We'll request that each node recieve the message at most once, so that the arc set we're looking fo is in the in-degree matroid with $k_v = 1$. We also want to prevent cycles, so that we do get a partial order on vertices, which can be expressed as the edge sets being part of the forest matroid, for discarded orientation. This structure, the intersection of this in-degree matroid and the forest matroid is the set of **branchings**. We can look for a maximum size branching, so as to maximise the number of times the message was passed.

Tree partition:
We can get more complicated problems as matroid interection problems with some clever trickts and gadgets. For example, we can ask if a given graph contains t edge-disjoint spanning trees. The case of $t = 2$ will be improtant for Shannon's switching game, in the game theory section. To reduce this to matroid intersection, we'll build a graph G_s consisiting of t disjoint copies of the original G, where the edges e_i are the copies of edge e in copy i. The forest matroid will consist of forests in each copy of G. Our goal is for these forests to be "disjoint" in the sense that only one of the copies e_i is used per original e, so that "projecting" these forests on the original graph yields edge disjoint forests. If we interect with the partition matroid with partition $\cup_{e \in E} \{e_1, .., e_t\}$ and $k = 1$, we get the desired property.
If w can find a set in the intersection of size $t(|V| - 1)$, then it's a union of t disjoint trees, as there can be at most $|V| - 1$ edges per copy. In particular, this is a maximum size intersection set, so that the problem reduces to finding the largest set in the matroid intersection.

Hamiltonian path:
We can express some problems as th intersection of more then two matroids. This is the case of the directed Hamiltonian path problem for digraphs. We can consider the intersection of the in-degree matroid with $k = 1$, with that of the out-degree matroid with $k = 1$. These arc sets consist of disjoint directed paths and cycles (follow unique neighbours until repeats or there are none). If we intersect with the forest matroid for discarded orientations, we prohibit the cycles.
To see that finding the maximum size set in the intersection of these 3 matroids is equivalent to finding a

directed Hamiltoninan path, we note that an intersection set of size $|V|-1$ must be a Hamiltonian path, and that all other intersection sets have lesser size. The latter is due to the size of a path with vertex set U is $|U|-1$, so that by summing over the paths in the intersection set and bounding by the total number of vertices, we get the result.

Thus, matroid intersection can express both polynomial-time-solvable problems, and NP-complete ones. Note that the number of matroids intersected changed between these types of problems.

We'll now introduce a few measures of sets wrt. an Independence system (E, I) that will come in useful later, when studying the greedy algorithm.

Ranks:

For a independence system (E, I), we define the **rank** $r : 2^E \to \mathbb{N}$ as a function providing the size of the largest independent set in a subset of E, $r(X) = \max(|Y| : Y \subseteq X, Y \in I)$.

We can consider the inclusion maximal independent subsets of an arbitrary set $X \in 2^E$, which we call the **bases** of X, and define the **lower rank** to be the size the the smallest such base, $\rho(X) = \min(|Y| : Y \subseteq X, Y \in I, Y \cup x \notin I, \forall x \in X \backslash Y)$. A base of a matroid is a base of E.

Finally, we can define the **rank quotient**, which measures the largest distance between bases, $q(E, I) = \min\limits_{\substack{X \in 2^E \\ r(X) \geqslant 1}} \left(\frac{\rho(X)}{r(X)} \right)$.

Indeed, the set Y attaining size $r(X)$ is a base of X, for otherwise we could augment it by som $x \in X \backslash Y$ so that $Y \cup x \in I$ is a larger independent set contained in X. This implies that $\rho(X) \leqslant r(X)$ for all X and in particular $q(E, I) \leqslant 1$.

Bases in matroids:

For a matroid (E, M), all bases of a set X have the same size, so that $\rho = r$ and $q(E, M) = 1$.
Conversely, if all bases of a set X have the same size, aka. $\rho = r$ or $q(E, I) = 1$, then the independece system satisfies the augmentation property and is therefore a matroid.

Proof: If two bases A, B of X had different size, say $|A| > |B|$, then by the augementation property the is an $a \in A \backslash B \subseteq X \backslash B$ such that $B \cup a$ is independent, and since it's also in X, this contradicts B being a base. Conversely, if bases have same size, and we take any two independent sets A, B such that $|A| > |B|$, then we can look at the bases of $X = A \cup B$. Since A is independent, we can possibly augment it by elements of X until we have a base of X of size $\geqslant |A|$, so a base of size $> |B|$, preventing B from being a base of X, as they're supposed to have same sizes. This means that B can be augmented by some $x \in X \backslash B = A \backslash B$, so that $B \cup x$ is independent, which is th result of the augmentation property.

COMPLETE: lower bound to quotient with circuits, rank submodularity.

5.2 The greedy algorithms

5.3 Matroid intersection

5.4 Matroid partition

5.5 Matroid union

5.6 Solutions

6 Hypergraphs

6.1 Basics on hypergraphs

We'll start with some basic hypergraph theory, since it seldom appears in discrete math classes, even advanced ones. Here's a bunch of definitions and examples:

Hypergraphs (undirected):

An **undirected hypergraph** $H = (X, D)$ consists of a finite set X of vertices, and a set $D \subseteq 2^X$ of subset of vertices $d \subseteq X$, called hyperedges.

The **edge-neighbourhood** of a vertex $x \in X$ is the set of hyperedges it's contained in, $\delta(x) = \{d \in D : x \in d\}$, and the **vertex-neighbourhood** of a vertex $x \in X$ is the set of vertices it shares a hyperedge with, $N(x) = \cup_{d \in D} d$.

The **degree** of a vertex $x \in X$ is the number of hyperedges it's contained in, $\deg(x) = |\delta(x)|$, and the **degree** of an hyperedge $d \in D$ is its size.

Matroids are hypergraphs.

Hypergraphs (directed):

An **directed hypergraph** $H = (X, D)$ consists of a finite set X of vertices, and a set $D \subseteq \left(2^X \times 2^X\right)$ of directed hyperedges. For an hyperedge $d = (t(d), h(d)) \in D$, the set $t(d), h(d) \subseteq X$ are called the **tail** and **head** of the edge, respectively.

6.2 Shortest hyperpaths

6.3 Flows on hypergraphs

Italian paper

6.4 Hypergraph cuts

Paper

6.5 Hypergraph coloring

Chapter 10 of Voloshin

6.6 Packing and covering in hypergraphs

Bondy Murty. Schrijver's Combinatorial optimization in the worst case.

6.7 Solutions

7 Computational Logic

7.1 Normal forms

Recall that logical statements can be combined with connectors. For example, given two Boolean variables p and q, we can state $p \Rightarrow q$, which is true unless q is true and p is false. More generally, we'll be interested in a Boolean function f of d variables x_i which take values among "true" and "false", so that $f : \{true, false\}^d \to \{true, false\}$.

An interesting fact is that any such function can be expressed using only "not", "and" and "or":

Normal forms:

We call a **disjunctive clause** an expression of form $\bigvee_{i=1}^{m} l_i(x_i)$, and a conjunctive one an expression of form $\bigwedge_{i=1}^{m} l_i(x_i)$, where $l_i(x_i)$ is called a **literal**, for which $l_i(x_i) = x_i$ or $l_i(x_i) = \neg x_i$.

We call a **conjunctive normal form (CNF)** an expression of form $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} l_{i,j}(x_{\pi_i(j)})$ and a **disjunctive normal form (DNF)** an expression of form $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} l_{i,j}(x_{\pi_i(j)})$, for some maps $\pi_i : [m_i] \to [d]$.

Any Boolean function can be expressed as a DNF and CNF.

Proof: The key observation is that conjunctive clauses act like indicators in a sense: $\bigwedge_{i=1}^{m} l_i(x_i)$ is true only for one configuration of the $x_{[m]}$ (possibly multiple is $m < d$). So to construct a DNF representing f, we can consider the set $f^{-1}(true)$, for each vector x of which we construct the conjunctive clause $\bigwedge_{i=1}^{d} l_i(x_i)$ where $l_i(x_i) = x_i$ if $x_i = true$ and $l_i(x_i) = \neg x_i$ if $x_i = false$. This clause is true only for x. Now, the dijuction of these clauses $\bigvee_{i=1}^{|f^{-1}(true)|} \bigwedge_{j=1}^{d} l_{i,j}(x_i)$ will be true for all $x \in f^{-1}(true)$, since its corresponding clause is true at that value. For all $x \notin f^{-1}(true) \Leftrightarrow x \in f^{-1}(flase)$, none of the clauses are true, so that the disjunction is false. So the expressions take the same values, and we can write $f(x) = \bigvee_{i=1}^{|f^{-1}(true)|} \bigwedge_{j=1}^{d} l_{i,j}(x_i)$. For CNFs, we can get a DNF of $\neg f$, and use $\neg\neg f = f$ together with Morgans laws to get a CNF of f.

In the worst case scenario, $|f^{-1}(true)| = 2^d$, so that the algorithm wil produce 2^d clauses in a DNF. Yet, we can do better for that as the DNF $x_1 \vee \neg x_1$ also represents f in that case. An interesting question is finding the smallest normal forms representing a Boolean function.

Note also the duality of DNFs and CNFs: if f has a DNF or CNF with index ranges n and (m_i), then $\neg f$ has a CNF or DNF with index ranges n and (m_i) repectively,and vice-versa, by Morgans laws.

7.2 SAT

Given a CNF, the task of finding an assignement of the Boolean variables that satisfies the CNF is known as **SAT**. Variants include **k-SAT** in which the clauses may contain at most k literals, and **max-SAT** in which we seek an assignment satisfying as many clauses as possible.
The analogous task for DNF's is easy: if a (conjunctive) clause contains x_i and $\neg x_i$, then the instance is unsatisfiable, and otherwise, we pick any clause, which we can satisfy by setting literals of form x_i to true, thos of form $\neg x_i$, and other non-apprearing literals to any value, so that the clause, and hence the disjunction, is satisfied.

For DNFs however, the task is NP-complete. We will show this in two steps, reducing SAT to 3-SAT and showing 3-SAT to be NP-complete. First, we'll solve SAT with 3-SAT as a black-box.
COMPLETE (Worell 4.4)

SAT is NP-complete

Proof: One often shows that 3-SAT can solve circuit-SAT. Instead, we'll solve independent-set with SAT, and SAT with independent-set.
SAT to Independence-set: We reduce the SAT instance to a 3-SAT one as we described previously. We'll actually deal with a case where all clauses have 3 literals. To transform an instance to that form, we'll introduce fake variables. For clauses C of size 2, we replace them by the two clauses $(C \vee f)$ and $(C \vee \neg f)$, so that C must be satisfied, and for clauses C of size 1, we replace them by the four clauses of form $(C \vee (\neg) f \vee \neg) f')$, to get the same effect.
Then, we solve the 3-SAT instance with independence set. We build triangles with vertices $v_{i,c}, v_{j,c} v_{k,c}$ for clause c, if it contains variables x_i, x_j, x_k. If the same variable is contained in two different clauses with and without negation, we add an edge between the corresponding vertices. In this context, an idependent set will use vertices that can't correspond to the same variable but with and without negation. It can also use just one vertex per triangle: this way, the number of satisfied clauses is exactly the number of vertices of the independence set.
Independent-set to SAT : for an independence-set instance, we use variables w_v to inidacte if v is in the independent set ($|V|$ variables). We then have clauses $(\neg w_u \vee \neg w_v)$ for edges $\{u, v\} \in E$ (so $|E|$ clauses) to encode the fact that two adjacent vertices can't be both in the independent set. To count, we will sieve through the graph as follows: we number the vertices in $[|V|]$ and introduce variables $w_{i,s}$ to denote if the number of vertices of the independent set in $[i]$ is s, where s ranges in $[0, i]$ ($|V|^2$ variables). We replace $w_{1,0}$ by $\neg w_{v_1}$ and $w_{1,1}$ by w_{v_1}. Next. we add clauses $(w_{i,s} \wedge w_{v_{i+1}}) \Rightarrow w_{i+1,s+1} \equiv (\neg w_{i,s} \vee \neg w_{v_{i+1}} \vee w_{i+1,s+1})$ ($|V|^2$ clauses), to ensure that when $[i]$ contains s vertices of the independent set, and v_{i+1} is in the independent set, then $[i+1]$ contains $s+1$ of them. To avoid expressing the converse, we'll use $(w_{i+1,s+1} \wedge w_{v_{i+1}}) \Rightarrow w_{i,s} \equiv (\neg w_{i+1,s+1} \vee \neg w_{v_{i+1}} \vee w_{i,s})$. To deal with the other case, we introduce the clauses corresponding to $(w_{i,s} \wedge \neg w_{v_{i+1}}) \Rightarrow w_{i+1,s}$ and $(w_{i+1,s} \wedge \neg w_{v_{i+1}}) \Rightarrow w_{i,s}$. In total, this will be $4|V|^2$ clauses. These calsues actually imply that for each i, exactly one of the $w_{i,s}$ is true. This is true for $i = 1$ and with in an inductive set, we disjoin on whether $w_{v_{i+1}}$ or $\neg w_{v_{i+1}}$ is true, in which cases the induction hypothesis and some $(w_{i,s} \wedge w_{v_{i+1}}) \Rightarrow w_{i+1,s+1}$ will provide the truth of a $w_{i+1,q}$ in the first case, while a $(w_{i,s} \wedge \neg w_{v_{i+1}}) \Rightarrow w_{i+1,s}$ will provide the truth of a $w_{i+1,q}$ in the second case. Uniqueness is handled similarly, where assuming the opposite, then disjoining on whether $w_{v_{i+1}}$ or $\neg w_{v_{i+1}}$ is true and using the $(w_{i+1,s+1} \wedge w_{v_{i+1}}) \Rightarrow w_{i,s}$ or $(w_{i+1,s} \wedge \neg w_{v_{i+1}}) \Rightarrow w_{i,s}$ respectively will contradict uniqueness guaranteed by the induction hyposthesis. Satisfying the instance we built will correspond to an independent set, where the $w_{|V|,s}$ variables will indicate the size of the set. We can then add clause $(w_{|V|,1} \vee ... \vee w_{|V|,k})$ to test if there is an independent set of size at most k.

2-SAT:
As is in general the case, some subclasses of the problem can be solved in polynomail time:

2-SAT:

2-SAT can be solved in polynomial time.

Proof: In a preprocessing step, we can look at all clauses of size one first. If we find x_i and $\neg x_i$ among them, we can conlcude with unsatisfiability. Next, we set their values so as to make the 1-clause true. We then replace this truth value in other clauses featuring the variable, using $x \vee true \equiv true$ so that this clause can be ignored, and $x \vee false \equiv x$, so that we get a new 1-clause. We reiterate this preprocessing stage until there are no more 1-clauses.
The crutial observation is now that we can replace clauses using $a \Rightarrow b \equiv \neg a \vee b$. We therefore seek an assignement that is coherent with this network of implications. We'll take that expression litterally and build a digraph who's vertices are the $x_1, ..., x_d$ and $\neg x_1, ..., \neg x_d$, adding an edge from vertex a to b if $a \Rightarrow b$ is a clause. We'll actually add the contrapositives too, so that we add an edge from vertex $\neg b$ to $\neg a$ if $a \Rightarrow b$ is a clause, which will come in handy later. If we follow a chain of implications, aka. a path in the graph, we may get $x_i \Rightarrow \neg x_i$, which isn't a contradiction. However, if $x_i \Leftrightarrow \neg x_i$, aka. we find a directed cylce in our graph that contains both vertices x_i and $\neg x_i$, then the clause is unsatisfiable, as $x_i \Leftrightarrow \neg x_i$ is false under any assignment. We can check this by using Floyd-Warshall and checking if in the distance matrix, there are no two index-symmetric non-zero entries.

FIX THIS ABSOLUTE MESS OF A CORRECTNESS PROOF.
If we don't have such cycles, can we build a satisfying assignment ?
The answer is yes, and there is a clever way of doing it. We iteratively perform the following. We consider a pair of unassigned variables x_i and $\neg x_i$ (so at most d loops). If there is a dipath of x_i to $\neg x_i$, then we set $x_i = flase$, if there is one from $\neg x_i$ to x_i, we set $x_i = true$, and if non of the casese apply, we set $x_i = true$ or $flase$, as this won't matter, as we'll soon see. The idea is that $false \Rightarrow anything$, so that setting $false$ at the "top" of an implication chain is the safest thing we can do. Next, we do the following: for each edge with at a $true$ at its tail and an unassigned variables at its head, we set it's head to $true$. This loop will finish as the number of unassinged variables decreases.
We'll prove correctness by showing the loop invariant of the outer loop that any vertex in a path from a vertex that was assigned true is true. This is true at the begining where all vertices are unassigned. To see the step, we'll first show the inner loop invariant that there are no paths leading from true vertices to false vertices. Indeed, this holds for the vertex pair x_i and $\neg x_i$, as when there is a dipath of x_i to $\neg x_i$, then there can't be one from $\neg x_i$ to x_i, which would be a path from true to false. There also can't be any path from a true vertex to x_i: if there was, the by the outer-loop invariant, x_i would have been true, so in particular assigned already. So setting $x_i = false$ doesn't violate the inner-loop invariant. The case of a path from $\neg x_i$ to x_i is similar. If no paths exist.....
The inner-loop invariant holds at each step of the inner loop, since by assining a variable to be true if it had an in-neighbour that was true can't produce a $true \Rightarrow flase$, unless the vertex had an out-neighbour that was false, in which case there was a path from a true node to a false one, contradicting the outer-loop invariant. Since at the end of the loop, we' can't have implcations of form $true \Rightarrow unassined$ (otherwise the loop would have done one more lap) nor $true \Rightarrow flase$, due to the inner-loop invariant, all paths emanating from true nodes most lead to true nodes. Thus, the outer-invariant is maintained.
Once the algorithm terminates, the last outer-loop invariant guarantees that we can't encounter $true \Rightarrow flase$, so that all clauses must be satisfied.

Horn formulas:
We now present another class of CNFs on which SAT can be solved in polynomial time, in fact in time linear in the number of clauses:

Horn formulas:

A **Horn clause** is a disjunctive clause with at most 1 non-negated variable in its literals, and a **Horn formula** is a conjunction of such Horn clauses. Solving SAT on such instances can be done in time (polynomial and) linear in the number of clauses.

A typical Horn clause has form $(\neg x_1 \vee ... \vee \neg x_n \vee x_{n+1})$, which is equivalent to $(x_1 \wedge ... \wedge x_n) \Rightarrow x_{n+1}$, in the same train of thought we used in 2-SAT.
However, general Horn clauses can be cathegorized into three categories: those we just described, and the "degenerate" cases of no negation, $(x_i) \equiv true \Rightarrow x_i$, and those of all negation, $(\neg x_1 \vee ... \vee \neg x_n) \equiv (x_1 \wedge ... \wedge x_n) \Rightarrow false$. With these reformulations, we can now start the proof.

Proof: The key remark is that if $x_1 \wedge ... \wedge x_n$ is true for some assignment, then assignement that differ only in the false values of that assignement will also be true. In a more sophisticted language, we can define a partial order on $\{true, false\}^d$ by identifying it with $\{0,1\}^d$ and using $x \leqslant y$ (componentwise). Then if $(x_{\phi(1)} \wedge ... \wedge x_{\phi(n)})$ is true and $x \leqslant y$, $(y_{\phi(1)} \wedge ... \wedge y_{\phi(n)})$ is true as well, for any injection $\phi : [n] \to [d]$.
Our algorithm will consist of starting with assingment $x^{(0)} = (false, ..., false)$ and iteratively taking an unsatisfied clause, and satisfying by increasing the assignement, if this is possible. So if a clause of form $\left(x^{(k)}_{\phi(1)} \wedge ... \wedge x^{(k)}_{\phi(n)}\right) \Rightarrow x^{(k)}_{\phi(n+1)}$ is unsatisfied, we can simply set $x^{(k+1)}_{\phi(n+1)} = true$, so that its satisfied in the next assignement, and will stay satisfied, as assignements only increase, so that all future $x^{(k+q)}_{\phi(n+1)} = true$, and since $anything \Rightarrow true$ is true. Note that setting $x^{(k+1)}_{\phi(n+1)} = true$ increased the assignement, so that the loop invariant of increasing assignements is maintained. If a clause $true \Rightarrow x^{(k)}_i$ is unsatisfied we do the same, and get the same result.
Only the case of an unsatified $(x_{\phi(1)} \wedge ... \wedge x_{\phi(n)}) \Rightarrow false$ is problematic. It turns out that a certain loop invariant of our algorithm allows us to conclude that in this case, the formula is actually unsatisfiable. This invariant is that if the formula is satisfied by an assignment y, then on all iterations, $x^{(k)} \leqslant y$. This will allow us to conclude, as for y to satisfy $(y_{\phi(1)} \wedge ... \wedge y_{\phi(n)}) \Rightarrow false$, one of the $y_{\phi(i)}$, say $y_{\phi(j)}$, must be false, so that if $x^{(k)}$ fails to satisfy $\left(x^{(k)}_{\phi(1)} \wedge ... \wedge x^{(k)}_{\phi(n)}\right) \Rightarrow false$, all $x^{(k)}_{\phi(i)}$ must be true, in particular $x^{(k)}_{\phi(j)}$ must be, which contradicts $x^{(k)} \leqslant y$ at that coordinate. Hence if $x^{(k)}$ fails to satisfy $\left(x^{(k)}_{\phi(1)} \wedge ... \wedge x^{(k)}_{\phi(n)}\right) \Rightarrow false$, no satisfying assignment y can exist, and we can conclude with unsatisfiability.
To prove the loop invariant, note that it's true at the initial step, as $x^{(0)} = 0 \leqslant y$. For the step, assume that $x^{(k)} \leqslant y$. In that case, we use our initial remark, to see that when $\left(x^{(k)}_{\phi(1)} \wedge ... \wedge x^{(k)}_{\phi(n)}\right)$ is true, so is $(y_{\phi(1)} \wedge ... \wedge y_{\phi(n)})$. Now, if we are in the case of an unsatisfied $\left(x^{(k)}_{\phi(1)} \wedge ... \wedge x^{(k)}_{\phi(n)}\right) \Rightarrow x^{(k)}_{\phi(n+1)}$, which is only the case when $\left(x^{(k)}_{\phi(1)} \wedge ... \wedge x^{(k)}_{\phi(n)}\right)$ is true and $x^{(k)}_{\phi(n+1)}$ is false, then we know that $(y_{\phi(1)} \wedge ... \wedge y_{\phi(n)})$ is true, and since it satisfies the clause by its definition, it must be the case that $y^{(k)}_{\phi(n+1)}$ is also true. So setting $x^{(k+1)}_{\phi(n+1)} = true$ will maintain $x^{(k+1)} \leqslant y$ in this case. Next, if $true \Rightarrow x_i$ is unsatisfied, but $true \Rightarrow y_i$ is, we must have $x_i = false$ and $y_i = true$, so that setting x_i to true in the next iteration maintains the invariant.
So if a satisfying assignment exists, $x^{(k)} \leqslant y$ holds for all iterations until we encounter the case of an unsatisfied $(x_{\phi(1)} \wedge ... \wedge x_{\phi(n)}) \Rightarrow false$. But such an encounter contradicts the existence of y.

This algorithm has an intersting property. Since the satisfying assignement y was arbitrary, we have that the output satisfies $x \leqslant y$ for all such assignments, provided one exists, in which case the algorithm terminates with an output. So the output is the smallest satisfying assignement for our order.

Random walks for SAT:
Worell Lecture 4, Extremal Combinatorics book chapter 23.1

7.3 Binary decision diagrams

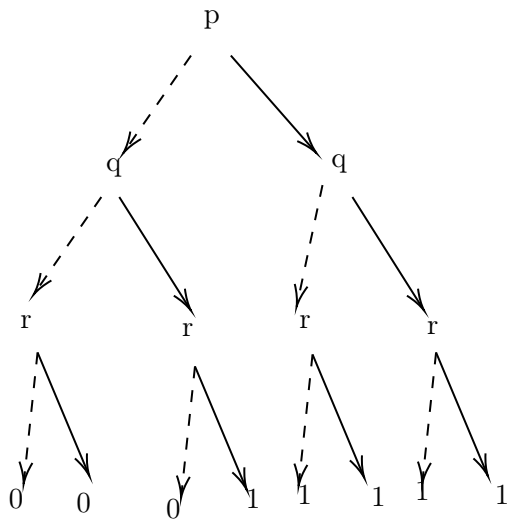
An interesting question is that of efficient storing truth tables or fast computation of a Boolean function. For example, in $p \vee (q \wedge r)$, is there a data-structure representing this Boolean function that would allow us to conclude that $p = true$ implies the value $true$?

An approach to this starts at representing a Boolean function by a binary decision tree, which we sequentially simplify. The following definition will be a lot clearer from the coming examples and constructions.

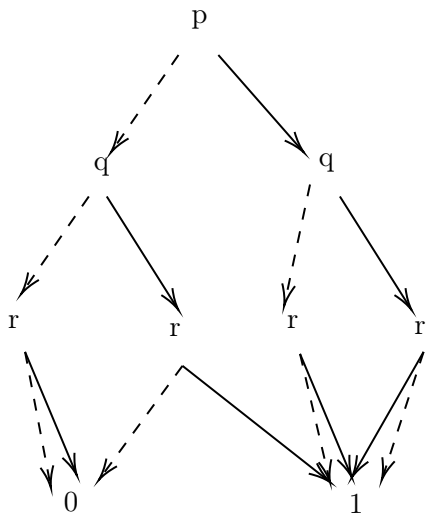
Binary decision diagrams:

A **binary decision diagram (BDD)** for a Boolean function with repect to an ordering of the Boolean variables is an acyclic digraph, where nodes are labeled with a variable and all non-sink nodes have exactly 2 out-edges, one labeled "true" and the other "false". All its sinks are labeld "true" or "false". The order on nodes induced by the acyclic digraph is compatible with the order of the labeling, in the sense that node u can be a parent of v if the label of u is smaller then that of v. The BBD represents the Boolean function in the following sense: if P is a dipath from a source to a sink (so a maximal path), which contains nodes with the labels in a set W of variables, then all assignements of the Boolean function, in which the variables of W have as truth values the labels of the edges they're the tail of in P, yield the same value, which is the value of the label of the sink end-point on P.

All Boolean functions have BBDs: we can use the "decision tree representation" of a Boolean function. We build a binary directed tree with layers labeled with the same variable, in the order of the variable order, and a last layer of sinks, who's label corresponds to the value of the function under the assignement corresponding to the path from root to leaf. For example, this is what we get for $p \vee (q \wedge r)$: All Boolean functions have BBDs: we can use the "decision tree representation" of a Boolean function. We build a binary directed tree with layers labeled with the same variable, in the order of the variable order, and a last layer of sinks, who's label corresponds to the value of the function under the assignement corresponding to the path from root to leaf. For example, this is what we get for $p \vee (q \wedge r)$:

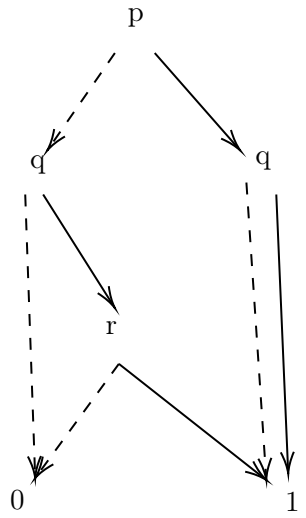


The question is if we can find a BDD with relatively few nodes.
There are ways in which BBDs can be "reduced", in the sense that we get a BBD representing the same Boolean function, but with less nodes. The first type of reduction is to group all sinks according to their label: we get at most two sinks with label 0 and 1, who's in-edges are have as tails the nodes that previously had an edge leading to a sink with that label. In our example, this lead to:

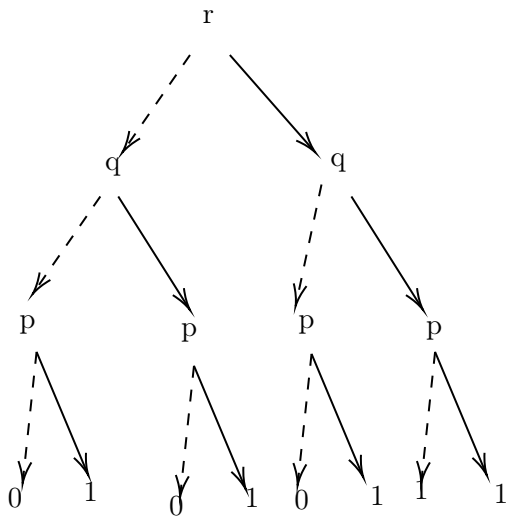


We still have a BBD representing the function, as the constarints on the order, the out-degree and the function representation where preserved.

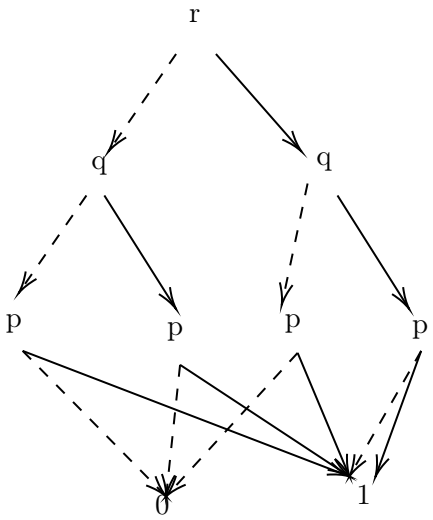
Now, notice in our example the appearance of double-edges: if we've followed the path from p to that r, then whatever edge we take next, we arrive a the same sink. This means that the value of the function, conditioned on those values of p and q is independent of that of r. We can therefore delete that node, letting the in-edges lead to the unique out-neighbour, in what's our second type of reductions:



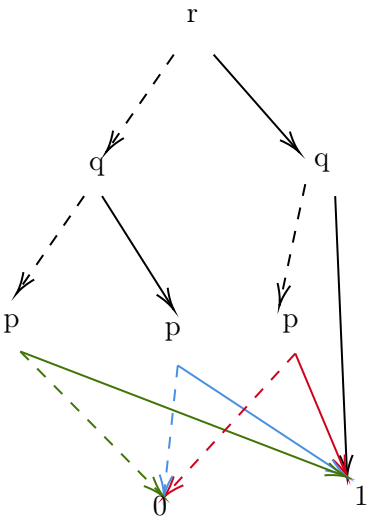
In our example, we can apply this type of reduction again, so a to arrive at a diagram of 5 nodes only. An intereseting quesion is whether the order of the variables mattered. Take the follwing order:



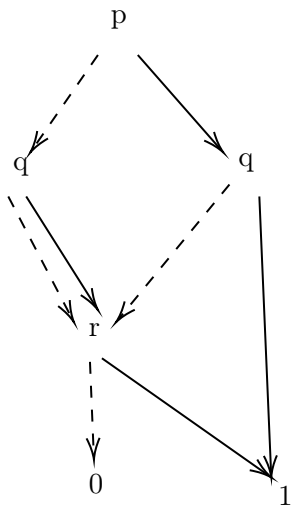
Then the first reduction yields:



And the second (ignore the colors for now):

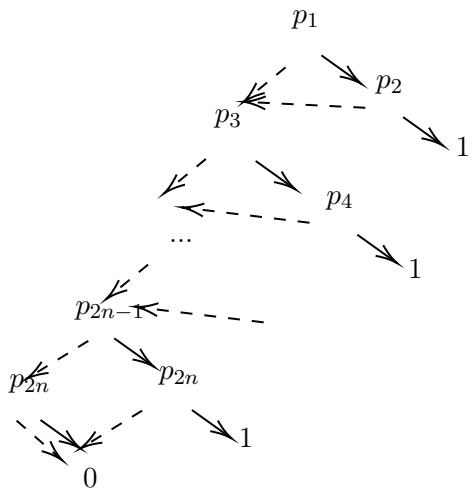


We can still simplify thing, though with a little more work. Note that the subgraphs indicated by the color are the essentially the same. We can group them in a third form of reduction to get:



In our example, we may perform one last reduction to get a representation of 5 nodes again.

However, in general, the number of nodes of different BDDs representing the same function can vary enormously depending on the imposed order. For example, consider the function $(p_1 \wedge p_2) \vee ... \vee (p_{2n-1} \wedge p_{2n})$. It can be represented by the $3n$ node diagram, in the index order:



We can make this a $2n + 2$ node diagram by fusing the nodes indexed 1.

If we impose order $1 < 3 < ... < 2n - 1 < 2 < 4 < ... < 2n$, then any BDD wrt. that order requires at least 2^n nodes. This is because knowing the values of the n odd variables doesn't allow us to conclude on the value of the function, as information on the even ones is necessary. Indeed, in a BBD must contain a binary tree of n layers, where each layer is labeled by an odd variable.
This is partly because the BBD can't skip odd variables, so that since the out-degree is always 2, we must have an n layer binary tree. If the BBD skiped an odd variable $2k - 1$, it wouldn't represent the function, as in the assignment $p_i = 0$ for $i \neq 2k - 1$ $p_{2k-1} = 1$ and $p_{2k} = 1$, the function has value 1, while in the assignment $p_i = 0$ for $i \neq 2k - 1$ $p_{2k-1} = 0$ and $p_{2k} = 1$, the function has value 0. So if variable $2k - 1$ wouldn't appear, we'd get a contradiction as the path corresponding to $p_i = 0$ for $i \neq 2k - 1$ and $p_{2k} = 1$ can't lead to two values, yet the function does for different assignments (different p_{2k-1}).
We can show this by induction. In the case $n = 1$, we can enumerate all candidates to BBDs and note that only least number of non-sink nodes needed is $2 = 2^1$. In the steps, disjoin on whether a hypothetical BBD has nodes labeled 1 or not. If it does, then it must be a source, so that deleting it forms a BBD for the subproblem on the rest of the variables with order $3 < ... < 2n - 1 < 2 < ...$and since we know that

the diagram can't skip odd variables, we can use the induction hypothesis to deduce that the layers on odd variables $3 < ... < 2n-1$ form a binary tree of 2^{n-1} nodes. FIX: find a source with an actual proof first.
If it doesn't, then the BBD cound't distinguish between assignements $1, 1, 0, ..., 0$ and $0, 1, 0, ..., 0$, which yield different function values, so this case can't occur.

We will now elaborate the reduction of "fusing" similar parts of the diagram. We conside parts to be similar if there is a graph isomorphism that preserves labels between them. It's important that the subgraphs we're talking about contain all the out-edges of their nodes. Otherwise, we'd allow isomorphisms of same-labeled nodes labeled by variables. If we delete a copy of these isomorphic graphs, and link all in-edges of the deleted copy to their image in the other copy, then we preserve acyclicity (if a cycle appeared, we'd alreday have a cycle with the edges of the pre-image), the out-degrees of 2, and the fact that maximal dipaths correspond to assignments and their value (the image of the path under the identification-morphism is still maximal, and the truth labels on the edges and the sink are the same).
To test for such isomorphisms can be done as follows. We consider the graphs that are the induced by the children of a node, and compare them layer by layer to graphs induced by the children of nodes that have the same label as that node.

Finally, we discuss the question whether we can perform reductions indefinitely. The answer is no since all reductions decrease the number of nodes in the BDD. If a BBD can't be reduced further with the reductions we mentioned, we call it reduced.

Building BDDs from connectors:
Our construction of a small BDD started from a massive binary tree that we successively reduced. This slow construction partly defetes the purpose of BDDs. The main puprose of BDDs is to have BDDs for a class of often used Boolean functions, from which more complicated ones are built, and on which we which to compute values of assignments or solve SAT (which is equivalent to the function having a BDD containing a sink labeled 1).
Assume we have a family of BDDs (d_i) for formulas (F_i), and we which to construct a BBD of a Boolean function obtained by a finite sequence of connectors/operators $\neg, \wedge, \vee, \Rightarrow, ...$ Is there an efficient way to proceed ?

We will see what to do one connector/operator at a time. For $\neg$, we can simply swap labels of the sinks of the BBD to their logical opposite to get a BBD of the negated function. It's a little more complicated for connectors.
We'll try to respect order, assuming the BBDs at dispositions follow the same order. We therefore look at the sources of both BBDs in what will be a recursive step of our algorithm. We'll call our recursive algorithm $merge(\circ, d_1, d_2)$, where $\circ$ denotes the connector, and the d_i are the BBDs of the functions to be connected with $\circ$.
If the sources are both labeled with a variable, and the same variable, then we build the new BBD by creating a source with that label, which will be the tail of a $false$-edge leading to the source of $merge(\circ, d_1', d_2')$, where d_i' is the graph of successors of the child along the $false$-edge of the root in d_i, as well as the tail of a $true$-edge leading to the source of $merge(\circ, d_1', d_2')$, where the $d_{i'}$ is the graph of successors of the child along the $true$-edge of the root in d_i. Assuming $merge$ does it's job on BDDs with less nodes, we can show that in this case, we get a BBD of the connection of functions. Acyclicity and out-degree are relatively easy to check, and to justify that this really represents of connection of the functions, take any assignment and follow the first disjunction on the first variable, which is represented by the root. It will lead us to the corresponding $merge(\circ, d_1', d_2')$, in which the d_i' are the BBDs of the successors of the appropriate child. By induction,

COMPLETE: seek different sources then Worell and Huth n Ryan, because they don't really prove stuff...

7.4 DPPL

We present a variant of the DPLL algorithm (Davis-Putnam-Logemann-Loveland) for solving SAT on CNFs. It relies on the notion of resolvents that we now present:

We'll be working with a representation of CNFs as a set-system of literals. To each clause, we associate the set of its literals, so for example $(\neg x \vee y \vee z) \approx \{\neg x, y, z\}$.
The aim is to deduce clauses from other clauses, for example in the goal of reducing the number of clauses representing the CNF, by merging them in a clever way. One such way is when two clauses C_1 and C_2 contain a literal $l \in C_1$ and its opposite $\bar{l} \in C_2$. To satisfy them both, we disjoin on the assignment of the variable of literal l: if l is true, then to satisfy C_2, we must satisfy $C_2 \backslash \bar{l}$ since $\bar{l}$ is false, and if $\bar{l}$ is true, then to satisfy C_1, we must satisfy $C_1 \backslash l$ since l is false.
We can group this as follows: consider the **resolvent of C_1 and C_2 wrt.** l, which is $R = (C_1 \backslash l) \cup (C_2 \backslash \bar{l})$. The resolvent is satisfyable precisely when C_1 and C_2 are simultaneously. When C_1 and C_2 are simultaneously satisfiable, one of them has a true literal that isn't l or $\bar{l}$ (as in one, that literal must be false, but since we assume the disjunction to be true, it must have a true literal): this literal will make the clause R true. Conversely, if R is true, it must have a true literal in $C_1 \backslash l$ or $C_2 \backslash \bar{l}$ (or both). If, for example, its in $C_1 \backslash l$, then if $C_2 \backslash \bar{l}$ isn't already satisfied, we can set $\bar{l} = true$ to satisfy C_2, while keeping C_1 satisfied.

7.5 Solutions

8 Computational/Numerical Algebra

8.1 Numerical linear algebra

Iterative solution methods, eigenvalue computation, gaussian method for quadratic forms, least squares and singular value decompositions

Gaussian algorithm for quadratic form reduction:
We consider a quadartic form $q(x) = x^t Ax = \sum_{i\in[d]} a_{ii}x_i^2 + 2\sum_{i<j\in[d]} a_{ij}x_ix_j$, which we hope to bring in form $q(x) = (Mx)^t B(Mx) = \sum_{i\in[d]} b_i \left(\sum_{j\in[d]} m_{ij}x_j\right)^2$ for a diagonal B and a change of variable given by the invertible M. The signs of the b_i will then inform us if the quadratic form is in-/definite, positive/negative. The idea Is to use algebraic relations to simplify expressions.

We will develop a recursive algorithm that recurses on the number of variables d, so that we prove its properties by induction on d. For the base case $d = 1$, quadratic forms have the desired form, where $b_1 = a_{11}$ and $M = 1$.

At a general iteration, we disjoin cases on whether we find an $a_{ii} \neq 0$, wlog a_{11}, or not. When $a_{11} \neq 0$, we can write $q(x) = a_{11}\left(x_1^2 + \frac{2}{a_{11}}x_1\left(\sum_{2\leqslant j\leqslant d} a_{1j}x_j\right)\right) + q'_r(x_2, ..., x_d)$ where $q_r(x_2, ..., x_d) = \sum_{i=2}^{d} a_{ii}x_i^2 + 2\sum_{2\leqslant i<j\leqslant d} a_{ij}x_ix_j$. Then, we can use $(a+b)^2 - b^2 = a^2 + 2ab$ to rewrite $a_{11}\left(x_1^2 + \frac{2}{a_{11}}x_1\left(\sum_{2\leqslant j\leqslant d} a_{1j}x_j\right)\right) = a_{11}\left(x_1 + \frac{1}{a_{11}}\left(\sum_{2\leqslant j\leqslant d} a_{1j}x_j\right)\right)^2 - \left(\sum_{2\leqslant j\leqslant d} \frac{a_{1j}}{a_{11}}x_j\right)^2$, so that for $b_1 = a_{11}$ and $m_{1j} = \frac{a_{1j}}{a_{11}}$, and $q_r(x_2, ..., x_d) = -\left(\sum_{2\leqslant j\leqslant d} \frac{a_{1j}}{a_{11}}x_j\right)^2 + q'_r(x_2, ..., x_d)$, we have $q(x) = b_1\left(\sum_{j\in[d]} m_{ij}x_j\right)^2 + q_r(x_2, ..., x_d)$. We then recurse on q_r, which is a quadratic form of less variables. To see that M is invertible, note that the $M = \begin{pmatrix} 1 & * \\ 0 & M_r \end{pmatrix}$ where M_r is the matrix from the recursion on q_r, which is invertible by induction.

To handle the case that all $a_{ii} = 0$, we either have a trivial quadratic form (set $b = 0$ and $M = I$), or some $a_{ij} \neq 0$, wlog a_{12}. In that case, we express $q(x) = 2\sum_{i<j\in[d]} a_{ij}x_ix_j$ as $q(x) = 2a_{12}x_1x_2 + 2x_1\left(\sum_{3\leqslant j\leqslant d} a_{1j}x_j\right) + 2x_2\left(\sum_{3\leqslant j\leqslant d} a_{2j}x_j\right) + q'_r(x_3, ..., x_d)$ where $q'_r(x_3, ..., x_d) = 2\sum_{i<j\in[d]} a_{ij}x_ix_j$. We then factor (with correcting term) the first part into $\frac{2}{a_{12}}\left(a_{12}x_1 + \sum_{3\leqslant j\leqslant d} a_{2j}x_j\right)\left(a_{12}x_2 + \sum_{3\leqslant j\leqslant d} a_{1j}x_j\right) - \left(\left(\sum_{3\leqslant j\leqslant d} a_{2j}x_j\right)\left(\sum_{3\leqslant j\leqslant d} a_{1j}x_j\right)\right)$, and collect the last term in $q_r(x_3, ..., x_d) = -\left(\left(\sum_{3\leqslant j\leqslant d} a_{2j}x_j\right)\left(\sum_{3\leqslant j\leqslant d} a_{1j}x_j\right)\right) - 2\sum_{i<j\in[d]} a_{ij}x_ix_j$, so that

$$q(x) = \frac{2}{a_{12}} \left(a_{12}x_1 + \sum_{3 \leqslant j \leqslant d} a_{2j}x_j \right) \left(a_{12}x_2 + \sum_{3 \leqslant j \leqslant d} a_{1j}x_j \right) + q_r(x_3, ..., x_d).$$

Next, we use $ab = \frac{1}{4}\left((a+b)^2 - (a-b)^2\right)$ to express the first part as

$$\frac{1}{2a_{12}} \left(\left(a_{12}x_1 + \sum_{3 \leqslant j \leqslant d} a_{2j}x_j + a_{12}x_2 + \sum_{3 \leqslant j \leqslant d} a_{1j}x_j \right)^2 + \left(a_{12}x_1 + \sum_{3 \leqslant j \leqslant d} a_{2j}x_j - a_{12}x_2 - \sum_{3 \leqslant j \leqslant d} a_{1j}x_j \right)^2 \right).$$

We get b_1 and b_2 from this, as well as M_{1*} and M_{2*} by doing the right associations. To see that M is invertible, note that $M = \begin{pmatrix} a_{12} & a_{12} & * \\ a_{12} & -a_{12} & * \\ 0 & 0 & M_r \end{pmatrix}$.

The power method for eigenvalue computation:

We consider a diagonalisable matrix A in dimension n tha isn't the 0-matrix. We consider a basis of eigenvectors (b_i) of A, so that any vector v can be written as $v = \sum_{i=1}^{n} v_i b_i$. We then can notice the following: $Av = \sum_{i=1}^{n} v_i \lambda_i b_i = \lambda \sum_{i=1}^{k} v_i \frac{\lambda_i}{\lambda} b_i + \lambda \sum_{i=k+1}^{n} v_i \frac{\lambda_i}{\lambda} b_i$, where we've ordered eigenvalues such that $|\lambda_1| = ... = |\lambda_k| > |\lambda_{k+1}| \geqslant ... \geqslant |\lambda_n|$ and where $\lambda = \lambda_1$. More generally, we'll have $A^p v = \lambda^p \left(\sum_{i=1}^{k} v_i \left(\frac{\lambda_i}{\lambda} \right)^p b_i + \sum_{i=k+1}^{n} v_i \left(\frac{\lambda_i}{\lambda} \right)^p b_i \right)$. The fact that $\left| \frac{\lambda_i}{\lambda} \right| < 1$ for $i > k$ hints at properties of a limit in $p \to \infty$. If we focus on a coordinate of the vectors, say the first, then we have $(A^p v)_1 = \lambda^p \left(\sum_{i=1}^{k} v_i \left(\frac{\lambda_i}{\lambda} \right)^p b_{i,1} + \sum_{i=k+1}^{n} v_i \left(\frac{\lambda_i}{\lambda} \right)^p b_{i,1} \right)$.

In the hopes of getting λ, we can study $\frac{(A^{p+1}v)_1}{(A^p v)_1} = \lambda \frac{\sum_{i=1}^{k} v_i \left(\frac{\lambda_i}{\lambda} \right)^{p+1} b_{i,1} + \sum_{i=k+1}^{n} v_i \left(\frac{\lambda_i}{\lambda} \right)^{p+1} b_{i,1}}{\sum_{i=1}^{k} v_i \left(\frac{\lambda_i}{\lambda} \right)^{p} b_{i,1} + \sum_{i=k+1}^{n} v_i \left(\frac{\lambda_i}{\lambda} \right)^{p} b_{i,1}}$.

Here we must make additional assumptions. One assumption is that $\lambda_1 = ... = \lambda_k$, which is in particular the case when $k = 1$. In that case, we have $\frac{(A^{p+1}v)_1}{(A^p v)_1} = \lambda \frac{\sum_{i=1}^{k} v_i b_{i,1} + \sum_{i=k+1}^{n} v_i \left(\frac{\lambda_i}{\lambda} \right)^{p+1} b_{i,1}}{\sum_{i=1}^{k} v_i b_{i,1} + \sum_{i=k+1}^{n} v_i \left(\frac{\lambda_i}{\lambda} \right)^{p} b_{i,1}} \xrightarrow[\infty]{p} \lambda$.

8.2 Interpolation

8.3 Symbolic integration

8.4 Number theoretical algorithms

Cryptography, FFT ?

8.5 Gröbner theory

In this section, we deal with the problem of solving systems of equations on multivariate polynomials. We consider polynomials $f \in \mathbb{R}[(x_i)_{i\in[n]}]$ or $f \in \mathbb{C}[(x_i)_{i\in[n]}]$, which have form $f = \sum_{a\in S} c_a x^a$, where we use multi-index notation, so that $S \subseteq \mathbb{N}^n$ is a finite set and $x^a = \prod_{i=1}^{n} x_i^{a_i}$, as well as $c_a \in \mathbb{R}$ or $c_a \in \mathbb{C}$.

The solution space: affine varieties

The solutions to a system $f_i(x_1, ..., x_n) = 0$ for $i \in [m]$, if there are any, in the particular field, aren't a set of finite points anymore, as it was the case for univariate polynomials (non-zero ones). For example for $f(x, y) = x - y \in \mathbb{R}[x, y]$, the solutions of $f(x, y) = 0$ form the line $y = x$. We can still get point-solutions, as as in $\begin{cases} x - y = 0 \\ -x - y = 0 \end{cases}$ who's only solution is $(0, 0)$. There may also be no solutions, such as in $\begin{cases} x^2 - y = 0 \\ x - 1 - y = 0 \end{cases}$, which is the empty intersection of a parabola and a line, since $x^2 > x - 1$ as $x^2 - x + 1 = \left(x - \frac{1}{2}\right)^2 + \frac{3}{4} > 0$.

Another example are the solutions of $\begin{cases} xz = 0 \\ yz = 0 \end{cases}$, as polynomials of $\mathbb{R}[x, y, z]$, which are the union of the x-y-plane $z = 0$ and the z-line $x = y = 0$, which isn't really a surface or a curve. Further intuition can be gained by the following example, where solution curves can sometimes be disconnected, as in the case of $\left((x - 2)^2 + y^2 - 1\right)\left((x + 2)^2 + y^2 - 1\right) = 0$, where solutions form two circles of radius one centered at $(\pm 2, 0)$ respectively.

We'll refer as the solutions of $f_i(x_1, ..., x_n) = 0$ for $i \in [m]$ as the **affine variety** $V(f_1, ..., f_m)$.

Ex.G1: Can any set be an affine variety ?

Solution representation: parametrization

So what do we mean by "solving" such a system ? As in the case of linear systems, we can attempt to parametrise the variety. For example, the solutions of $x - y = 0$ can be parametrised with $t \mapsto (t, t)$, and for a more complicated example, those of $x^2 + y^2 - 1 = 0$ can be parametrized with $t \mapsto \left(\frac{1 - t^2}{1 + t^2}, \frac{2t}{1 + t^2}\right)$ (with some nice geometry), as $\left(1 - t^2\right)^2 + 4t^2 = 1 + 2t^2 + t^4 = \left(1 + t^2\right)^2$, so that any point of the parametrisation is on the circle, and conversely, by incerting $t = \frac{y}{x + 1}$ for $(x, y) \neq (-1, 0)$, we have $\left(\frac{1 - t^2}{1 + t^2}, \frac{2t}{1 + t^2}\right) = \left(\frac{(x + 1)^2 - y^2}{(x + 1)^2 + y^2}, \frac{2(x + 1)y}{(x + 1)^2 + y^2}\right) = \left(\frac{x^2 + 2x + \left(1 - y^2\right)}{2x + 2}, \frac{(2x + 2)y}{2x + 2}\right) = (x, y)$ by using $1 - y^2 = x^2$ so that the first coordinate simplifies to $\frac{x(2x + 2)}{2x + 2}$. The parametrization doesn't reach point $(-1, 0)$, as $\frac{2t}{1 + t^2} = 0$ implies $\frac{1 - t^2}{1 + t^2} = 1$.

In general, we'll look for a set or rational functions $r_k \in \mathbb{R}((t_i)_{i\in[p]})$ for $k \in [n]$ such that for all $(t_1, ..., t_p) \in \mathbb{R}^p$ that aren't poles of an r_k, we have $(r_1(t_1, ..., t_p), ..., r_n(t_1, ..., t_p)) \in V(f_1, ..., f_m)$, meaning the this point is a solution. Of course, immediate questions are how to find such parametrizations, how to tell if there even are any, and tell if they cover all of the variety, or how much if not all of it.

The plan, version 1.

We briefly sketch the way we'll find solutions to polynomial systems.

Consider the example $\begin{cases} xy^2 - y + x = 0 \\ xy - x^2y^2 - 1 = 0 \end{cases}$: if we multiply the first row by x and add it to the second in order to cancel out the term of high power, we get $\begin{cases} x + x^3y = 0 \\ x^2 - 1 = 0 \end{cases}$, so that the second row constrains solutions to the union of lines $x = 1$ and $x = -1$. On the first line, the equations $xy^2 - y + x = 0$ and $xy - x^2y^2 - 1 = 0$ yield both $y^2 - y + 1 = 0$, so that the points $\left(1, \frac{1 \pm i\sqrt{3}}{2}\right)$ form solutions in $\mathbb{C}^2$ and there are no real solutions with $x = 1$ in this case, and on the second line $x = -1$ we have equation $y^2 + y + 1 = 0$, so that we also get additional solutions $(-1, \frac{-1 \pm i\sqrt{3}}{2})$ in $\mathbb{C}^2$ and there are no real solutions.
In this example, we took a "polynomial-linear" combination that provided a equation of lower degree, $x^2 - 1 = 0$, that we could analyse more easily. This is similar to Gaussian elemination where we sought combinations that cancel variables. We then checked if the solutions to the new equations, or a part of them, were in fact solutions to the system.
We will devellop the division algorithm, which will allow us to obtain polynomials of lower degree. We'll define a notion of degree and of oder on it, that will relate to the idea of reducing the difficulty in our system of equations. Gröbner bases are what will make the division algorithm work. Once we know how to simplify the system of equations, we'll discuss how to recover the actual solutions from the simplifications.

Ideals
In an attempt to generalise Gaussian elimination for linear systems, we'll be interested in taking "polynomial-linear combinations" of the lines of system $f_i(x_1, ..., x_n) = 0$ for $i \in [m]$, which are expressions of from $\sum_{i=1}^{m} h_i f_i$, where the h_i are any polynomials in $\mathbb{R}[(x_i)_{i\in[n]}]$. The set of these expressions is called the **ideal generated** by the f_i, and is denoted by $\langle f_1, ..., f_m \rangle$.

They relate to the equations in the sense that for $x \in V(f_1, ..., f_m)$ and $f \in \langle f_1, ..., f_m \rangle$, we have $f(x) = 0$. The converse doesn't hold, in the sens that solutions of $f(x) = 0$ may not be in $V(f_1, ..., f_m)$, as the basic example of $f = x - y \in \langle x, y \rangle$ shows, since $f(1, 1) = 0$, yet $V(x, y) = \{(0, 0)\}$. However, we can make a certain change of basis. For example, in $\begin{cases} x^2 - y = 0 \\ -x^2 - y = 0 \end{cases}$, we have $x^2, y \in \langle x^2 - y, -x^2 - y \rangle$ (sum and difference divided by 2), and the solutions of $\begin{cases} x^2 = 0 \\ y = 0 \end{cases}$ are the same as those of the original system. In general, if we have two **bases/generating sets** such that $\langle f_1, ..., f_m \rangle = \langle g_1, ..., g_q \rangle$, then $V(f_1, ..., f_m) = V(g_1, ..., g_q)$ (write one systems as linear combination of the other). In our example, $\langle x^2 - y, -x^2 - y \rangle = \langle x^2, y \rangle$ as we can generators as scalar-linear combinations of the others.
Ex.G2: For a given set V, we can define the **ideal of** V to be $I(V) = \{f : f(x) = 0, \forall x \in V\}$. Is it true that $\langle f_1, ..., f_m \rangle = I(V(f_1, ..., f_m))$?

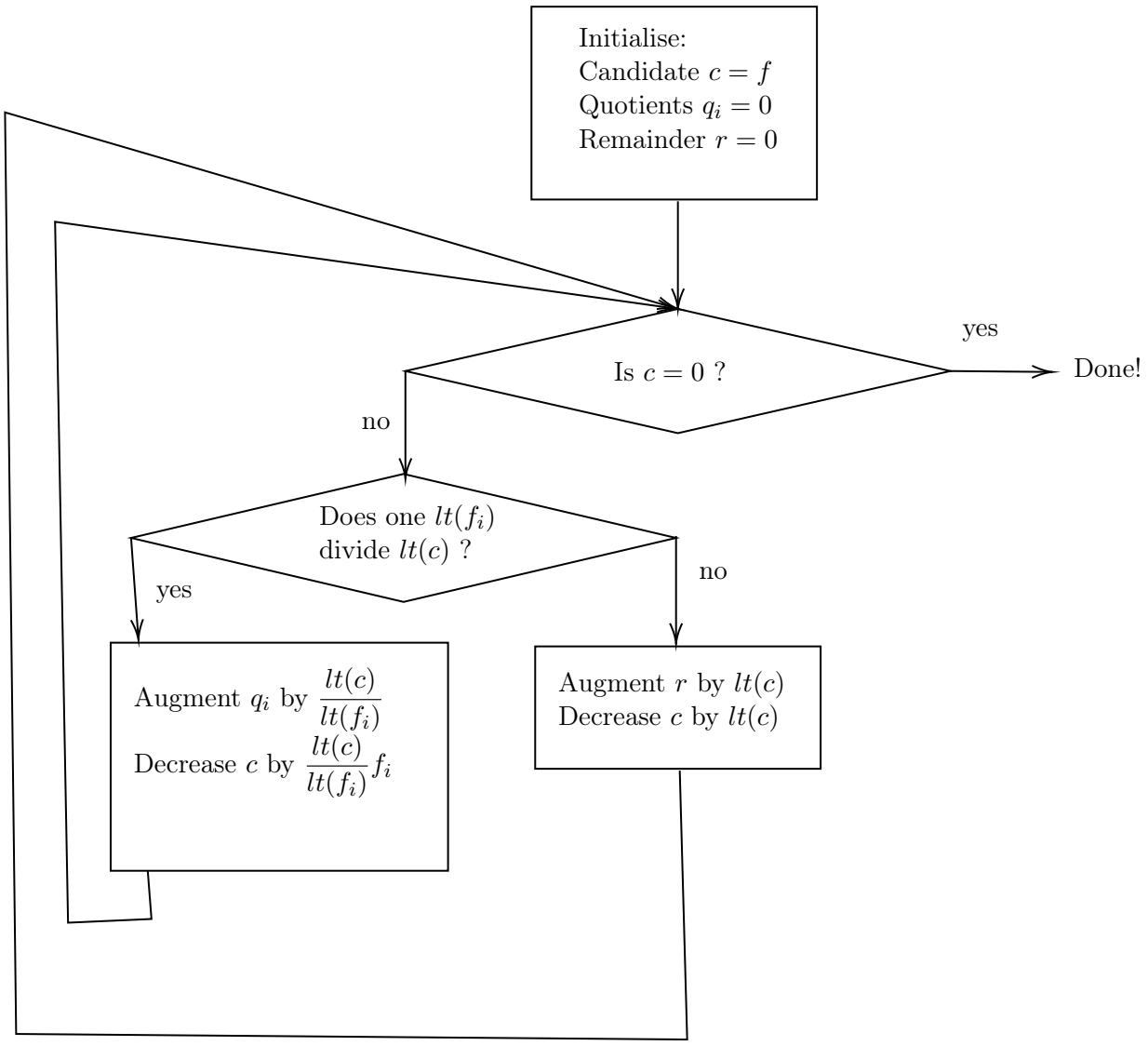
Ordering
Our goal is to reduce the number of variables in the system by performing operations that lower the degree. So the notion of degree and the order on it should reflect this idea. In multi-index notation, monomial are represented by points of $\mathbb{N}^n$. The **lexicographic ordering** of $a > b \Leftrightarrow \exists i \in [n]$ so that $a_j = b_j$ for $j < i$ and $a_i > b_i$ seems to be well suited, since for example monomials without the variable x_1 will be of lower degree then the monomials that do have x_1, so that elimination of x_1 is favoured by degree reduction.
Ex.G3: Show that this is an order on $\mathbb{N}^n$. Show that a finite subset of $\mathbb{N}^n$ has a maximum in this order. Depending on the indexing of variables, multiple lexicographic orderings are possible. In general, on can work with the abstract notion of a monomial ordering, but we'll work with the lexicographic ordering, and introduce its properties when they are needed.

The **degree** $\deg(f)$ of a polynomial f will be the multi-index that is greatest among the indices of the monomials making up the polynomial. The **leading monomial** $lm(f)$ of a polynomial f will be the monomial attaining the maximum defining the degree. The **leading coefficient** $lc(f)$ of a polynomial f will be the coefficient of $lm(f)$ in f. The **leading term** $lt(f)$ of a polynomial f will be $lc(f)lm(f)$.

The division algorithm

We'll try to generalise the generalization of euclidean division to univariate polynomials. We'll also extend it to the possibility of having multiple divisors. For an f and divisors f_i not all zero, we seek quotients q_i and remainder r such that $f = r + \sum_{i=1}^{m} q_i f_i$. Instead of seeking particular properties of the remainder, we'll see what we can get from the generalisation of division in the univariate case. In that case, we successively reduced the dividend by mutiples of the divisor by eliminating leading terms. In the multivariate and multi-divisor case, we can check if an $lt(f_i)$ divides $lt(f)$. If it does, we can write $f = \left(f - \frac{lt(f)}{lt(f_i)} f_i\right) + \frac{lt(f)}{lt(f_i)} f_i$, where $\left(f - \frac{lt(f)}{lt(f_i)} f_i\right)$ will be the candidate for the next iteration and which has lower degree then f, as we'll soon show. The part $\frac{lt(f)}{lt(f_i)}$ could be made to contribute to q_i. In case that none of the $lt(f_i)$ divide $lt(f)$, we'll make our lives easy and write $f = (f - lt(f)) + lt(f)$, where $(f - lt(f))$ is the candidate of the next iteration, and $lt(f)$ will be attributed to the remainder. We perform these steps until the candidate has become 0 (which it does, as we'll soon show), so as to get an expression of form $f = r + \sum_{i=1}^{m} q_i f_i$.

Summarising in a flow-chart:



Note that after each loop, we have $f = c + r + \sum_{i=1}^{m} q_i f_i$, an invariant maintained by the loops. So if the algorithm terminates, we'll indeed have $f = r + \sum_{i=1}^{m} q_i f_i$. The property of the remainder is that it's either zero, or none of its constituing monomials are divisible by any of the $lt(f_i)$, which is also a loop invariant.

We'll now show that the algorithm terminates.
We'll show that the degree of candiate c drops in each iteration, and that it can drop only finitely many times. Indeed, in iterations where the remainder is augmented, c is decreased by its leading term, so that the degree decreases. In iterations where the quotient is augmented, we have to show that the monomials of $\frac{lt(f)}{lt(f_i)} f_i$ have smaller multi-index then the leading term, which is $lt(f)$, so that the degree of c decreases in those iterations too. The fact that $lt(f_i) | lt(f)$ translates to $\exists k, sx^a = kx^b tx^c$ where $lt(f) = sx^a$ and $lt(f_i) = tx^c$, so that $a = b + c$. If d is the multi-index of any other monomial of f_i then the leading one, then $c > d$, so that $a = c + b > d + b$. Thus, $\frac{lt(f)}{lt(f_i)} f_i$ has same leading term as f and its other monomials have lower multi-index, so that $f - \frac{lt(f)}{lt(f_i)} f_i$ has lower degree then f.

Now, we'll show that a sequence of decreasing multi-indices must eventually reach $(0, ...0)$. Once that degree is reached, c is a constant that will be accounted in the remainder in a last iteration, if it isn't zero already. The property we show is known as the well-ordering of the lexicographic order, and a formal proof requires induction. In the univariate case $n = 1$, this is just the well-ordering of $\mathbb{N}$. For the induction step, we note that the elements in the sequence for which a_1 has the same value must be finite. This is due to the induction assumption, since the projection on the last coordinates in $\mathbb{N}^{n-1}$ forms a lexicographically decreasing sequence, which is finite by induction. So a_1 can take the same value only finitely many times in the sequence, after which it must decrease in value, so that eventually, a_1 must reach value 0.

The plan, version 2.
Recall the big idea of solving polynomial systems: we take polynomial-linear combinations of the polynomials defining the system, hoping that higher index monomials cancel out, preferably resulting in a polynomial who's indices are so low in the lexicographic context, that the number of variables has decreased, preferably to a single variable.
The problem is that there is little insight in getting these cancellations for arbitrary $f_1, ..., f_m$. What if instead we had a base $g_1, ..., g_q$ such that $\langle f_1, ..., f_m \rangle = I = \langle g_1, ..., g_q \rangle$ (so we have same variety), which had the property that for any $f \in I$, $lt(f)$ would be divisible by some $lt(g_i)$? We would then have insigth into how low of a degree the $f \in I$ could have, as their leading term must be divisible by one of the g_i, so that $\deg(f) \geqslant \deg(g_i)$. If one of the g_i is a univariate polynomial, then we would know by $g_i \in \langle f_1, ..., f_m \rangle = \langle g_1, ..., g_q \rangle$ that we can get it as a polynomial-linear combination, so that solutions would have to have their corresponding variable be among the roots of g_i. If on the other hand none of the g_i have univariate leading term, we'd know that it's pointless to search for a univariate polynomial-linear combination of the $f_1, ..., f_m$, since its leading term would have to be divisible by a multivariate leading term of some g_i, which is impossible.
Note that such a basis $g_1, ..., g_q$ would also have the property that the division of an $f \in I$ by the $g_1, ..., g_q$ would always have remainder 0. Indeed, throughout the algorithm, $c \in I$ as a polynomial-linear combination, so that the case of its leading term not being divisible by that of a g_i, which would augment the remainder, never occurs.
These bases will be Gröbner bases. We'll now charcterise the property defining Gröbner bases, give some of their properties and of course give an algorithm for finding them, showing that they do indeed exist.

COMPLETE: skip Dickson and Hilbert, and go directly to Buchberger ???

8.6 Polynomial identity testing and applications

Consider the following problem. Alice and Bruno a part of a secret society. Members of the secret society know a password, which allows members to certify that they're part of the group. However, if Alice and Bruno want to find out if the other is a member of the secret society, they can't simply communicate the password to each other, as they'd run the risk of the other lying about being in the society, and getting their hand on the password by asking for it. How can they certify that they're members to each other, without divuging the password ?

Suppose the password is a sequence of numbers $p_1, ...p_m$. Alice and Bruno could start by fixing an ordering of the sequence into multi-index notation, so that the sequence is for example index in the form $p_{i,j,k}$. They then form the 3-variate polynomial $\sum_{i,j,k} p_{i,j,k} x_1^i x_2^j x_3^k$. If Alice and Bruno have the same password, the polynomials A and B formed this way should be equal. The key point is that they can test for equality by evaluating the polynomials at different points. If the polynomials are equal, then they should evaluate too the same. If one among Alice and Bruno is a liar, then for many evaluations, it seems unlikely that the liar will always get the correct value, which will identify them as a liar. Also, if sufficiently few test of this sort are performed, the liar can't deduce the password from the test, as there won't be enough equations $y = \sum_{i,j,k} p_{i,j,k} x_1^i x_2^j x_3^k$ to determine the $p_{i,j,k}$. The question is therefore, how likely is it for a non-zero $f = A - B$ to get a root of it when choosing some point at random ?

Schwarz-Zippel lemma:

We consider an n-variate non-zero polynomial f over a field $\mathbb{F}$, of multidegree-sum bounded by d. If we pick a point uniformily in the grid S^n for some set $S \subseteq \mathbb{F}$ of finite size $|S| > d$, then the probability of getting a root of f is at most $\frac{d}{|S|}$. In particular, doing this for $|S| = 2d$ (for an infinite field) for k idependent times, and getting only roots, occurs with pobability at most 2^{-k}.

The condition on d is that for all monomials $\prod_{i \in [n]} x_i^{m_i}$ of f, we have $m_1 + ... + m_n \leqslant d$. In the context of the password, it means that slicing the password yields better results. Indeed, if the password has size 27, we would have $d \geqslant 27$ in the univariate case, but by slicing it into 3 coordinates ranging in [3], we'd have $d \geqslant 3 + 3 + 3 = 9$ for a 3-variate polynamial. However, the space on which we randomize would go from $|S| \geqslant 27$ to $|S^3| \geqslant 729$.

Proof: We'll show that the number of roots of f in S^n is at most $d|S|^{n-1}$, so that under uniform probability, the probability of getting one is at most $\frac{1}{|S|^{n-1}} d|S|^{n-1} = \frac{d}{|S|}$.
The proof of this is by induction on the number of variables n. For $n = 1$, d is at least the degree of f, and f has at most $\deg(f)$ roots (as it's non-zero), so that there are at most $d|S|^0$ roots.
For the step, we write f as $f = \sum_{i=0}^{m} f_i x_{n+1}^i$ where the f_i are polynomials in the $x_1, ..., x_n$, and where $f_m \neq 0$, in the polynomial sense (if no $f_i \neq 0$ existed, we'd have $f = 0$, opposite to our assumption). If we fix the $(x_1, ..., x_n)$ to some value $(a_1, ..., a_n) \in S^{n-1}$, $\sum_{i=0}^{m} f_i x_{n+1}^i$ is univariate and has at most m roots, unless it's zero. The latter can only occur if all $f_i(a_1, ..., a_n) = 0$, ad in particular, $(a_1, ..., a_n)$ is a root of f_m. We can estimate how often this happens by our induction hypothesis. Indeed, if the multidegree

of f_m is $m_1, .., m_n$, then by assumption on d, we had $m_i + ... + m_n + m \leqslant d$, so that by the induction hypothesis applied to f_m on S^n with $d-m$, we have at most $(d-m)|S|^{n-1}$ possible $(a_1, ..., a_n)$ for which all $f_i(a_1, ..., a_n) = 0$. In those cases, for any of the $|S|$ choices of value for x_{n+1}, we get a root $(a_1, ..., a_n, x_{n+1})$ of f.

When this doesn't happen, which may happen at most $|S|^n$ (all) times, $\sum_{i=0}^{m} f_i x_{n+1}^i$ has at most m roots. So the total number of roots in S^{n+1} is at most $m|S|^n + (d-m)|S|^{n-1}|S| = d|S|^n$, which concludes the step.

COMPLETE: polynomial method for perfect matchings.

8.7 Solutions

Ex.G1: No! for example, consider $cone((1,1))$. Any $P \in \mathbb{R}[x,y]$ that vanishes on it satisfies $P(t,t) = 0$ for all $t \geqslant 0$. Since $P(t,t)$ is a polynomial in one variable with infinitely many zeros, it must be the zero polynomial, so that $P(t,t) = 0$ for all $t \in \mathbb{R}$. So if a variety contains $cone(1,1)$, it must contain $span(1,1)$, so that $cone(1,1)$ by itself is not a variety.

Ex.G2: We do have $\langle f_1, ..., f_m \rangle \subseteq I(V(f_1, ..., f_m))$. Indeed, if $f \in \langle f_1, ..., f_m \rangle$, then $f(x) = 0$ for $x \in V(f_1, ..., f_m)$, since f is a combinationof the f_i, so that $f \in I(V(f_1, ..., f_m))$. However, the converse may not hold! A technical problem is for example displayed in $\langle x^2 \rangle$ for which $I\left(V\left(x^2\right)\right) = \{f : f(0) = 0\}$, so that $x \in I\left(V\left(x^2\right)\right)$, but $x \notin \langle x^2 \rangle$, since multiplying by non-zero polynomials only raises the degree.

Ex.G3: Both $a > b$ and $a < b$ can't hold together, as we'd reach a contradiction at the minimum index. To see transitivity, by letting i be the index in $a > b$ and j that in $b > c$, we have equality of coordinates until $k = \min(i, j)$, where $a_k \geqslant b_k$ and $b_k \geqslant c_k$ with at least one strict, so that $a_k > c_k$ and hence $a > c$. For the existence of a maximum, assume the contrary, which yields a sequence of points greater an greater, so that by finiteness, we'll evetually get contradition $a > a$ for some a in the set, by transitivity.

9 Basics on polyhedra

In this chapter, we develop the relevant notions on polyhedra used for linear and integer linear programming. To motivate these notions, you may skip this chapter and come back to it when the relevant notions and results turn up in the coming chapters.

9.1 Polyhedra

Convexity:
Let $y_1, ..., y_n$ be a finite set of points. A **convex combination** of y_i's is given by:

$$\sum_{1 \leq i \leq n} \alpha_i y_i \quad \forall\, \alpha_i \in \mathbb{R}_{\geq 0},\ \sum_{i=1}^{n} \alpha_i = 1$$

The **convex hull** of a set S is $conv(S)$, the convex combinations of finite subsets of S.

Cones:
Let $y_1, ..., y_n$ be a finite set of points. A **conic combination** of y_i's is given by:

$$\sum_{1 \leq i \leq n} \alpha_i y_i \quad \forall\, \alpha_i \in \mathbb{R}_{\geq 0}$$

The **conic hull** of a set S is $cone(S)$, the conic combinations of finite subsets of S.

Polytopes:

An **H-polyhedron** is $P = \cap_{i=1}^{n} H_i^+$ where

$$\text{Halfspace: } H^+ = (v \neq \overrightarrow{0} \in \mathbb{R}^d,\ \ \alpha \in \mathbb{R}) = \{x \in \mathbb{R}^d : \langle v, x \rangle \leq \alpha\}$$
$$\text{Hyperplane: } H = (v, \alpha) = \{x \in \mathbb{R}^d : \langle v, x \rangle = \alpha\}$$

so that P has form $P = \left\{x \in \mathbb{R}^d : Ax \leqslant b\right\}$ for some $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^d$.
(If P is bounded $P = \cap_{i=1}^{n} H_i^+$ is a poly**tope**)

An **E-polyhedron** is a $P = \text{conv}(X) + cone(Y),\ \ |X|, |Y| < \infty,\ \ X, Y \subseteq \mathbb{R}^d$
(For polytopes, Y is empty)

They are actually the same:

Main Theorem of Polyhedra:

Let $P \subseteq \mathbb{R}^d$. P is a H-polytope $\Leftrightarrow$ P is an E-polytope.

The proof is long and not of immediate interest to us. All that we'll mention about it is that in its constructive versions, the algorithms aren't polynomial time and aren't efficient in practice.

Examples: A **d-cube** is both the convex hull of all vectors with 0-1-coordinates, $conv\left(\sum_S e_i : S \subset [d]\right)$, and the intersection $\bigcap_{i=1}^{d}(0 \leqslant x_i \leqslant 1)$. To see this, notice that the points $\sum_S e_i : S \subset [d]$ are in the intersection, so that by its convexity, $conv\left(\sum_S e_i : S \subset [d]\right) \subset \bigcap_{i=1}^{d}(0 \leqslant x_i \leqslant 1)$; the back-inclusion is harder: we use induction on the dimension, so we write $(x_1, ..., x_d) = x_d(x_1, ..., x_{d-1}, 1) + (1 - x_d)(x_1, ..., x_{d-1}, 0)$, a convex combination of points which are in $(d-1)$-cubes, and use the induction hypothesis.
A **d-simplex** is the convex hull of any $d+1$ affinely independent points. The standard d-simplex is $conv(0, e_i : i \in [d]) = \left(\sum_{i=1}^{d} x_i = 1\right) \cap \bigcap_{i=0}^{d}(0 \leqslant x_i)$.
There is an analog version of these descriptions for cones:

(polyhedral) Cones:

An **H-cone** is a $C = \cap_{i=1}^{n} H_i^+$ where

$$\text{Halfspace: } H^+ = (v \neq \overrightarrow{0} \in \mathbb{R}^d, \ = \{x \in \mathbb{R}^d : \langle v, x\rangle \leq 0\}$$
$$\text{(vectorial) Hyperplane: } H = (v, \alpha) = \{x \in \mathbb{R}^d : \langle v, x\rangle = 0\}$$

A **R-cone** is a $C = \text{cone}(X), \ |X| < \infty, \ X \subseteq \mathbb{R}^d$

They are actually the same:

Main Theorem of Cones:

Let $C \subseteq \mathbb{R}^d$. C is a R-cone $\Leftrightarrow$ P is an H-cone.

One can relate cones and polyhedra via a technique called **homogenization**. We'll show how the main theorem for cones implies that for polyhedra, using this technique.
Given a V-polyhedron $conv(V) + cone(R)$, it's points have form $x = V\lambda + R\mu$ where $\lambda, \mu \geqslant 0$ and $1^t\lambda = 1$. We can get a cone from this by lifting dimension: we have $\begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} V \\ 1^t \end{pmatrix}\lambda + \begin{pmatrix} R \\ 0^t \end{pmatrix}\mu$ where $\lambda, \mu \geqslant 0$ for points x in the V-polyhedron, which is now equivalent to $\begin{pmatrix} x \\ 1 \end{pmatrix}$ being in the V-cone generated by $\begin{pmatrix} V \\ 1^t \end{pmatrix}$ and $\begin{pmatrix} R \\ 0^t \end{pmatrix}$, which we call the **V-homogenisation**. The main theorem for cones then provides an H-description $Ay \leqslant 0$ for this cone, so that x is in the V-polyhedron precisely when $A\begin{pmatrix} x \\ 1 \end{pmatrix} \leqslant 0$, or equivalently $A_{*[d]}x \leqslant -A_{*(d+1)}$, which is an H-description for polyhedra.
Conversely, given an H-polyhedron $Ax \leqslant b$, we can conisder it as an intersection of the cone $Ax \leqslant bz$ with $z = 1$. For reasons that will soon become clear we consider the **H-homogenisation** of the H-polyhedron

given by the H-cone $\begin{pmatrix} A & -b \\ 0^t & -1 \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} \leqslant 0$ (we require $z \geqslant 0$) so that x is in the H-polyhedron precisely when $\begin{pmatrix} x \\ 1 \end{pmatrix}$ is in the H-cone $\begin{pmatrix} A & -b \\ 0^t & -1 \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} \leqslant 0$. Again with the main theorem for cones, the latter H-cone has V-description $cone(E)$, so that x is in the H-polyhedron precisely when $\begin{pmatrix} x \\ 1 \end{pmatrix} = E\lambda$ for some $\lambda \geqslant 0$. In order to have a convex combination appear, we rescale E to $\begin{pmatrix} V & R \\ 1^t & 0^t \end{pmatrix}$ by rescaling the comlumns of E for which $e_{(d+1),k} > 0$ by $\dfrac{1}{e_{(d+1),k}}$. This is why we required $z \geqslant 0$: it ensures that the case $e_{(d+1),k} < 0$ doesn't appear. If we then resacle λ to λ' by $\lambda'_k = \dfrac{1}{e_{(d+1),k}} \lambda_k$, then we still have $\lambda' \geqslant 0$ and $\begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} V & R \\ 1^t & 0^t \end{pmatrix} \lambda'$. The latter shows x is a sum of a convec combination of V and a conic one of R. Therefore, this is a V-description of the H-polyhedron.

We conclude with the Farkas Lemmas, which we prove in the computational geometry chapter.

Farkas:

Cones: For a V-cone $B\mathbb{R}^d_+$ and a vector b, either $b \in B\mathbb{R}^d_+$ or he can separate b from $B\mathbb{R}^d_+$ by a vectorial hyperplanes, so that $h^t b < 0$ and $h^t B\mathbb{R}^d_+ \geqslant 0$.

Polyhedra: An H-polyhedron $Ax \leqslant b$ is either non-empty, or we can find a contradictory combination of its defining equations in the form of $\lambda^t A = 0$ and $\lambda^t b < 0$ for some $\lambda \geqslant 0$.

Standard: A polyhedron in standard form $Ax = b$, $x \geqslant 0$ is either non-empty, or we can find a contradictory combination of its defining equations in the form of $\lambda^t A \geqslant 0$ and $\lambda^t b < 0$ for some λ.

9.2 Faces

The intuitive notion of a face of a polyhedron can be made formal through the following definition: a face is a part of the polyhedron for which the polyhedron is on one particular side of it (we don't cut the polyhedron in half along it).

Faces of polyhedra:

A **face** of a polyhedron P is a subset $F \subseteq P$ such that there is a hyperplane H for which $P \cap H = F$ and $P \subseteq H^+$ (the polytope is in one of the halfspaces and meet the hyperplane in the face).
The **dimension of the face** is the dimension of the affine hull of the face.
Faces of dimension 0, 1 and $d-1$ are called **vertices**, **edges/rays** (depending on if they're bounded) and **facets** respectively.

Remark: The hyperplanes defining the H-polytope may not define faces (for example, if a parallel hyperplane is included in a first ones halfspace): those are redundant. Hyperplanes defining a face might not be unique: for example, The edge $(0,0) \times [0,1]$ of the unit cube is defined by the planes given by $2x + y \geqslant 0$ and $2y + x \geqslant 0$.

Ex.F1: Prove that the normal vectors of the hyperplanes defining a face form a cone.

Characterisation of faces (H-version):

If a halfspace $c^t x \leqslant d$ is valid for a polyhedron $Ax \leqslant b$, then there is a $\lambda \geqslant 0$ such that $c = A^t\lambda$ and $d \geqslant b^t\lambda$. In addition if a halfspace $c^t x \leqslant d$ is face defining for a polyhedron $Ax \leqslant b$, then there is a $\lambda \geqslant 0$ such that $c = A^t\lambda$ and $d = b^t\lambda$ (c is in the cone of the normals to the hyperplanes defining the polyhedron). More precisely, we have two descriptions of the face: $\left\{ x : \begin{cases} Ax \leqslant b \\ c^t x = d \end{cases} \right\} = \left\{ x : \begin{cases} Ax \leqslant b \\ A_{supp(\lambda)*} x = b_{supp(\lambda)} \end{cases} \right\}$.
Conversely, for all $I \subseteq [m]$, the sets $\left\{ x : \begin{cases} Ax \leqslant b \\ A_{I*} x = b_I \end{cases} \right\}$ form faces (possibly empty or all of the polyhedron). In particular, there are at most 2^m faces.

Proof: Note the similarity of the expressions of the first statement to those of the Farkas lemmas. We prove the first statement by contraposition. So we start from the non-existance of a $\lambda \geqslant 0$ such that $c = A^t\lambda$ and $d > b^t\lambda$. In order to use Farkas, we homogenize this statement into the non-/existence of a $(\lambda, s) \geqslant 0$ such that $\begin{pmatrix} A^t & -c \end{pmatrix} \begin{pmatrix} \lambda \\ s \end{pmatrix} = 0$ and $\begin{pmatrix} b^t, -d \end{pmatrix} \begin{pmatrix} \lambda \\ s \end{pmatrix} < 0$. The two are equivalent since $s > 0$, otherwise $0 = A^t\lambda$ and $0 > b^t\lambda$ would make the polyhedron empty (the system infeasible), so that we can divide by s and get a new $\lambda \geqslant 0$ for which $c = A^t\lambda$ and $d > b^t\lambda$. Now, with polyhedron-Farkas, we get the existence of a solution to $\begin{pmatrix} A \\ -c^t \end{pmatrix} x \leqslant \begin{pmatrix} b \\ -d \end{pmatrix}$, or equivalently $Ax \leqslant b$ and $c^t x \geqslant d$, so that x is a point of the polyhedron of on the other side of the halfspace, preventing $c^t x \leqslant d$ from being valid but not face-defining.
For the face-defining case, we show that there is a $\lambda \geqslant 0$ such that $c = A^t\lambda$ and $d \geqslant b^t\lambda$ and exclude the case $d > b^t\lambda$. Indeed, if $d > b^t\lambda$, then for a point x on the face, so $c^t x = d$, we have $c^t x = \lambda^t A x \leqslant b^t\lambda < d$, an immediate contradiction. To show the first part, we use contradiction and to link the statement to Farkas, we introduce slack: we start from the assumption that $c^t x \leqslant d$ is face-defining and that there is a

$\lambda, s \geqslant 0$ such that $c = A^t\lambda$ and $d = b^t\lambda + s$. Rewriting this as $\begin{pmatrix} A^t & 0 \\ b^t & 1 \end{pmatrix}\begin{pmatrix} \lambda \\ s \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix}$, with $\lambda, s \geqslant 0$, we can use standard-Farkas to get h such that $h^t\begin{pmatrix} A^t & 0 \\ b^t & 1 \end{pmatrix} \geqslant 0$ and $h^t\begin{pmatrix} c \\ d \end{pmatrix} < 0$, or equivalently $Ah_{[d]} \geqslant -h_{d+1}b$, $h_{d+1} \geqslant 0$ (crutially, due to slack), and $c^t h_{[d]} < -h_{d+1}d$. Dividing by $-h_{d+1} \leqslant 0$ if it's non-zero then provides h' such that $Ah' \leqslant b$ and $c^t h' > d$, so that the plane is not even valid. If $-h_{d+1} = 0$, we get $A(-h_{[d]}) \leqslant 0$ and $c^t(-h_{[d]}) > 0$, and since the polyhedron is non-empty and since $c^t x \leqslant d$ is face defining, for any point y on that face, $A(y - h_{[d]}) \leqslant b$ and $c^t(y - h_{[d]}) > d$, an immediate contradiction.

Now, for the points on the face defining plane we have $\begin{cases} Ax \leqslant b \\ c^t x = d \end{cases}$ which rewrites to $\begin{cases} Ax \leqslant b \\ \lambda^t Ax = \lambda^t b \end{cases}$, we must have $A_{supp(\lambda)*}x = b_{supp(\lambda)}$. To see this, assume that there is an $i \in supp(\lambda)$ such that $A_{supp(\lambda)*}x \neq b_{supp(\lambda)}$: since x is in the polyhedron, we have $A_{supp(\lambda)*}x < b_{supp(\lambda)}$, but then summing with $\lambda \geqslant 0$ and $\lambda_i > 0$ we have $\lambda^t Ax < \lambda^t b$, contradicting the choice of x. Conversely, if $A_{supp(\lambda)*}x = b_{supp(\lambda)}$, then $\lambda^t Ax = \lambda^t b$ as the terms not indexed by $supp(\lambda)$ vanish.

We now show that for all $I \subseteq [m]$, the sets $\left\{ x : \begin{cases} Ax \leqslant b \\ A_{I*}x = b_I \end{cases} \right\}$ form faces. We'll build a supporting hyperplane with $c^t = \sum_{i \in I} A_{i*}$ and $d = \sum_{i \in I} b_i$: indeed, $c^t x = \sum_{i \in I} A_{i*}x \leqslant \sum_{i \in I} b_i = d$ for all points of the polyhedron $Ax \leqslant b$, and for those points for which $A_{I*}x = b_I$, there is actually equality.

Characterisation of faces (V-version):

For a polyhedron $P = cone(V) + cone(R)$ and a face F with defining halfspace $c^t x \leqslant d$ of it, we can find index subsets I and J such that $F = cone(V_{*I}) + cone(R_{*J})$, where $V_{*I} = \left\{ V_{*i} : c^t V_{*i} = d \right\}$ and $R_{*J} = \left\{ R_{*j} : c^t R_{*j} = 0 \right\}$. We also note that $c^t R \leqslant 0$.

Proof: Indeed, $cone(V_{*I}) + cone(R_{*J}) \subseteq F$ as they are in P and satisfy $c^t x = d$ (convex combination and definitions). Next, note that we must have $c^t R \leqslant 0$. Otherwise, if $c^t R_{*j} > 0$, then for large enough s, we'll have $c^t(V_{*1} + sR_{*j}) > d$ contradicting validity of the hyperplane. If we decompose elements of F according to $cone(V) + cone(R)$, if one of the terms (with positive scalar) satisfied $c^t V_{*i} < d$ or $c^t R_{*j} < 0$, then the sum would be $< d$, contradicting the fact that its one the face defined by $c^t x = d$, so actually $F \subseteq cone(V_{*I}) + cone(R_{*J})$.

Note that the converse doesn't hold: not every $cone(V_{*I}) + cone(R_{*J})$ is a face of P. For example consider $conv((1,0),(0,0),(0,1)) + cone(1,1)$, where $(0,0) + cone(1,1)$ isn't a face.

Vertices:

Vertices can't be the convex combination of other points of the polyhedron.
A polytope is the convex hull of its vertices.
The set of hyperplanes of the H-polyhedron of dimension d that a vertex is in has size at least d.
There are finitely many vertices.

Proof: We take a hyperplane defining vertex v, given by $n \cdot x = b$. Then $n \cdot x < b$ for all vertices in the polytope but for v, for which $n \cdot v = b$. If we could write the vertex as a convex combination of other points

of the polytope, $v = \sum s_i p_i$, taking the dot-porduct with n, bounding with $s_i(n \cdot p_i) < s_i b$ and recalling that $\sum s_i = 1$, we arrive at a contradiction to $n \cdot v = b$. So v isn't a combination of other points.
By the main theorem of polytopes, the vertices are in $conv(S)$, for a point set S. Seeing as they can't be combinations of other points, the vertices must be included in S. The points in S are of two types: either they are combinations of other points of S or they aren't. By proving that those in the second case are vertices, we can show that $conv(S) = conv(V)$ where V is the set of vertices.
So if $s \notin conv(S \backslash s)$, the separation theorem gives us a hyperplane passing through s and such that $conv(S \backslash s)$ is in one of the open halfspaces of it. This hyperplane defines s as a vertex.

To get the next claim, we'll prove that the set of hyperplanes of the H-polytope of dimension d that a vertex is in are such that their normal vectors span $\mathbb{R}^d$: therefore, there must be at least d such vectors, meaning d such hyperplanes.
If this wasn't the case, then for normal hyperplanes $n_i \cdot x \leqslant b_i$ such that for the vertex v, $n_i \cdot v = b_i$, the system given by the $n_i \cdot x = 0$ would have a solution space of dimension 1 at least. So if we take a non-zero solution y to it, then we notice that $v \pm t.y$ verifies $n_i \cdot x = b_i$ and that for the other hyperplanes defining the polytope it verifies $N_i \cdot x \leqslant B_i$ for $t \in \left[\max_{N_i \cdot y < 0} \left(\frac{B_i - N_i \cdot v}{N_i \cdot y} \right), \min_{N_i \cdot y > 0} \left(\frac{B_i - N_i \cdot v}{N_i \cdot y} \right) \right]$ (for $v + t.y$, and similarly for $v - t.y$). This means that we can choose a $t > 0$ such that the $v \pm t.y$ are in the polytope. But v is then the midpoint (convex combination) of the segment with endpoints $v \pm t.y$, which contradicts the fact that it's a vertex.
The converse is true in a certain sense. If the n_i span $\mathbb{R}^d$, the the system $n_i \cdot x = b_i$ has a unique solution. If this solution v is in the polytope (= it verifies the other $N_i \cdot x \leqslant B_i$) then it's a vertex by our previous arguments, since it can't be the convex combination of other points of the polytope: if $v = \sum s_i x_i$, then $n_i \cdot x \leqslant b_i$, $n_i \cdot v = b_i$ and the nature of scalars require that $n_i \cdot x_i = b_i$, contradicting uniqueness of the solution.

Finally, this last condition allows us to prove that there are finitely many vertices: there are only finitely many ways of choosing d hyperplanes of the H-polytope so that their normal vectors span the space, each such way defining a vertex. □

Remarks: A polyhedron may not have any vertices. For example, the polyhedron given by $y \geqslant x$ and $y \geqslant -x$ in $\mathbb{R}^3$ has no vertices, as the H-description has only 2 hyperplanes, while 3 are necessary. The next definition will investigate this phenomenon.
Also, a vertex of a d-polytope may be in more then d hyperplanes. For example, the top of an Egyptian pyramid, which is a 3-polytope, is in 4 hyperplanes.

A point that isn't the convex combination of any other points of the polyhedron is called an **extreme/al point**. One can show with separation theorems that extreme points of a polytohedron are vertices.

Lineality space and recession cone:

The **lineality space** of a cone C is the set of directions of lines contained in the cone $L(C) = \{x : lin(x) \subseteq C\}$. For an H-cone $C = \{x : Ax \leqslant 0\}$, they are $L(C) = \{x : Ax = 0\}$. The lineality space is a vector space and a face of the cone. It's included in every other face of the cone, so that all faces have larger dimension then it.
The **lineality space** of a non-empty polyhedron P in H-description $Ax \leq b$ is $L(P) = \{x : Ax = 0\}$. Every (non-empty) face of P (including itself) contains a translate of the lineality space of P.

Polyhedra with lineality space $\{0\}$ are called **pointed**.

One can also define the **recession cone** $rec(P)$ of a polyhedron P that represents the directions such that a ray in this direction is contained in P: $rec(P) = \{y : \exists x \in P, x + t.y \in P, \forall t \geqslant 0\}$. In particular, on can see that $L(P) = rec(P) \cap (-rec(P))$. In H-description, $rec(P) = x : Ax \leqslant 0$
For V-description $P = conv(V) + cone(R)$, we have $rec(P) = cone(R)$.

Indeed, if $A(b + t.x) \leqslant 0$, it must be that $Ax = 0$, otherwise some values of t lead to a contradiction. $Ax = 0$ is a vector space that can have arbitrary dimension, depending on A.
To see that it's a face, we consider the cone of the normal vectors of A given by $(A_{i*})^t$: for $a = \sum (A_{i*})^t$, $a \cdot x \leqslant 0$ on the cone and $a \cdot x = 0$ precisely on the lineality space, it requires that $Ax = 0$.
Finally, to see that $L(C)$ is in any other face of C, consider the face F of C defined by hyperplane $n \cdot x \leqslant b$. Then since C is a cone and since $n \cdot x \leqslant b$ must hold on all of C, $n \cdot x = 0$ or $b = 0$, for otherwise $n \cdot (t.x) \leqslant b$ leads to a contradiction of signs by letting $t \in [0, \infty)$ vary. But since the hyperplane intersects the cone, there is some $x \in C$ such that $n \cdot x = b$, so in any case, $b = 0$.
In our final step, we prove that $n \in cone\left((A_{i*})^t\right)$ so that $Ax = 0$ implies $n \cdot x = 0$, meaning that $x \in F$ and so $L(C) \subseteq F$. If this wasn't the case, we can use the Faraks lemma (which is based on the separation theorem) to get the existence of an x such that $x^t A^t \leqslant 0$ and $x \cdot n > 0$. The first inequation rephrases to $Ax \leqslant 0$, so that $x \in C$: since $x \cdot n > 0$, we contradict $n \cdot x \leqslant b$.

For the lineality space of a polyhedron, not that if x is some point of P and $y \in L(P)$, then $x + y \in P$ (add the systems). If a face is defined by the valid inequality $c^t x = d$ so that $c^t x \leqslant d$ for the polyhedron, then take any $y \in L(P)$ and assume for contradiction that $c^t y \neq 0$: then since $x \pm y$ are in the polyhedra, one of them must verify $c^t(x \pm y) > d$, contradicting the validity of the hyperplane. So $c^t y = 0$ and hence $c^t(x + y) = d$, showing that its on the face.

To see that $rec(P) = \{y : Ay \leqslant 0\}$, note that $\{y : Ay \leqslant 0\} \subseteq rec(P)$ since for any $x \in P$ and $\lambda \geqslant 0$, we have $A(x + \lambda y) \leqslant b + \lambda.0$. Now assume for contradiction that there is a $y \in rec(P)$ such that there is a line i for which $A_{i*}y > 0$, then for large enough λ for a given $x \in P$, $A_{i*}(x + \lambda y) > b$, contradicting the definition of $y \in rec(P)$.
To show $rec(P) = cone(R)$, note that $rec(P) \supseteq cone(R)$, since $t.cone(R) \subseteq cone(R)$ for $t \geqslant 0$. For $rec(P) \subseteq cone(R)$, we consider $y \in rec(P)$ and assume for contradiction that $y \notin cone(R)$. Then by Farkas, we can separate y from $cone(R)$ by a hyperplane $h^t y < 0$ and $h^t R \geqslant 0$. By noting that for all $x \in P$, $h^t x \geqslant \min_i \left(h^t V_{*i}\right)$ ($h^t R \geqslant 0$ and the minimum of $h^t V \lambda$ where λ is in the standard simplex) we get a contradiction to $x + t.y \in P$ for all $t \geqslant 0$ by considering $t = \dfrac{|\min_i \left(h^t V_{*i}\right)| + |h^t x|}{|h^t y|} \geqslant 0$, as $h^t(x + t.y) = h^t x - \left(|\min_i \left(h^t V_{*i}\right)| + |h^t x|\right) < \min_i \left(h^t V_{*i}\right)$.

Proposition:

If P is non-empty and pointed, then all its faces contain a vertex (of themselves and therefore of P too).

Proof:
We'll use the fact that vertices and extreme points are the same. We'll first show that extreme points of face F are extreme points of the polyhedron P. If we write the convex combination $x = \sum s_k y_k$ for

$y_k \in P$ and $s_k > 0$, and face F is given by $a \cdot x \leqslant b$, then if we had $a \cdot y_k < b$ for some index, then bounding $a \cdot x = \sum s_k (a \cdot y_k)$, we'd get $a \cdot x < b$, so that x couldn't have been on the face. So $a \cdot y_k = b$, meaning $y_k \in F$, so that for an extreme point of the face $x = \sum s_k y_k$, we have $y_k = x$, making x an extreme point of the polyhedron.
We can then prove the proposition by induction on the dimension. This is because faces are lower dimensional polyhedra, so the if we prove that a non-empty pointed polyhedron has a non-empty face, then since this faces lineality space is a subspace of the polyhedron's one, the face is also pointed and induction hypothesis provides an extreme point on it, which will also be an extreme point of the polyhedron.
If P is non-empty and pointed, the take one of it's points x and some direction y: as it's pointed the line in direction y passing through x can't be contained in all of the polyhedron, meaning that at some point z on it, z must verify the system of the H-description of the polyhedron with at least one tight inequality. Since that tight inequality defines a face, z is on that face and the face is non-empty.
The base case of the induction is the 0-dimensional one which is true by definitions.

We now focus on a particular type of polyhedra given by H-description $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$, with $A \in \mathbb{R}^{m \times n}$ of full row rank $m < n$, as this type of polyhedron is important for solving LPs.
This is indeed a polyhedron as we can describe it with $\begin{pmatrix} A \\ -A \\ -I \end{pmatrix} x \leqslant \begin{pmatrix} b \\ -b \\ 0 \end{pmatrix}$ where we've used the trick $x = y \Leftrightarrow \begin{cases} x \geqslant y \\ x \leqslant y \end{cases}$. Its lineality space is therefore $\ker \left(\begin{pmatrix} A \\ -A \\ -I \end{pmatrix} \right) = \ker(I) = \{0\}$, so that such polyhedra are always pointed. In particular all their faces have vertices, which we now characterise:

Proposition:
The vertices of polyhedra of form $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$ are the solutions to $\begin{cases} A_{*B} x_B = b \\ x_{[n]\setminus B} = 0, x \geqslant 0 \end{cases}$ for some subset $B \subset [n]$ of size $|B| = m$ (called a basis) so that the columns $(A_{*j})_{j \in B}$ are linearly independent, so that $x_B = A_{*B}^{-1} b$, $x_{[n]\setminus B} = 0$ and $x \geqslant 0$.

Proof: As we've seen in the previous general characterisation of vertices, the vertices of $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$ must be in at least n hyperplanes who's normal vectors span all of the space. Since any point of the polyhedron is already in the m hyperplanes $A_{i*} x = b_i$, there are at least $n - m$ coordinates so that $x_i \geqslant 0$ is tight, for a vertex. Since by the previous general characterisation of vertices the normal vectors should span the space, and A has full row rank m, we can complete the rows of A into a base of the space with unit vectors. If the coordinates of these unit vectors are in a set $S \subseteq [n] \setminus \{i : x_i > 0\}$ of size $|S| = n - m$, then the vertex x verifies $\begin{pmatrix} A \\ e_{i \in S}^t \\ \vdots \end{pmatrix} x = \begin{pmatrix} b \\ 0 \end{pmatrix}$, where $\begin{pmatrix} A \\ e_{i \in S}^t \\ \vdots \end{pmatrix}$ is a square matrix of full row rank. This matrix therefore also has full column rank, meaning that it's columns are linearly independent. We now set B to be $[n] \setminus S$. For B, the latter means that the columns $\begin{pmatrix} A_{*j} \\ 0 \end{pmatrix}$ for $j \in B$ are linearly independent, so that actually the A_{*j} are linearly independent. Since we chose B so that $|B| = n - |S| = m$, we've finished the first part of the proof. Note that there may have been multiple choices for S (and therefore for B) since

the vertex could have been in more then $n-m$ hyperplanes of form $x_i = 0$.
For the converse, we consider a solution to $\begin{cases} A_{*B}x_B = b \\ x_{[n]\setminus B} = 0, x \geqslant 0 \end{cases}$ for some subset $B \subset [n]$ of size $|B| = m$ (called a basis) so that the columns $(A_{*j})_{j\in B}$ are linearly independent. This is an extreme point as for any convex combination $x = \sum s_k y_k$, the condition $y_k \geqslant 0$ and $x_{[n]\setminus B} = 0$ forces $(y_k)_{[n]\setminus B} = 0$, so that $Ay_k = b$ becomes $A_{*B}(y_k)_B = b$, which has a unique solution as the $(A_{*j})_{j\in B}$ are linearly independent, so that $x = y_k$.

Minimal proper faces and characterisation of the recession cone:

Proof: Consider any $y \in rec(P)$, and we denote by $eq(y)$ the line for which $Ay \leqslant 0$ is tight. By the characterization of faces, $\begin{cases} Az \leqslant 0 \\ A_{eq(y)}z = 0 \end{cases}$ is a face of $rec(P)$, which is non-empty as it contains y. If $y \in L(P)$, we get the result. Otherwise, the face $\begin{cases} Az \leqslant 0 \\ A_{eq(y)}z = 0 \end{cases}$ contains a minimum proper face, and hence one of the y_k. Since $y_k \notin L(P)$ by assumption, there is a line $i \notin eq(y)$ such that $A_{i*}y_k < 0$. We now consider $y - \lambda y_k$ for $\lambda \geqslant 0$. For some pivoting $\lambda > 0$, we'll have $A(y-\lambda y_k) \leqslant 0$, $A_{eq(y)}(y-\lambda y_k) = 0$ and $A_{j*}(y-\lambda y_k) = 0$ for some $j \in [m]\setminus eq(y)$, because $\begin{cases} Ay_k \leqslant 0 \\ A_{eq(y)}y_k = 0 \end{cases}$ and $A(-\lambda y_k) \geqslant 0$. For this value, $y - \lambda y_k$, will be a point of the recession cone, but with $|[m]\setminus eq(y-\lambda y_k)| < |[m]\setminus eq(y)|$. If we reiterate this procedure on $y - \lambda y_k$, and so on, getting expressions of form $y - \sum \lambda_k y_k$ along the way, we'll eventually end up at $y - \sum \lambda_k y_k \in L(P)$, since eventually $|[m]\setminus eq\left(y - \sum \lambda_k y_k\right)| = 0$. But then $y = \sum \lambda_k y_k + L(P)$, which is the desired result.

Related to the previous is the notion of extreme rays:

Extreme Rays:

We consider a pointed cone $Ax \leqslant 0$, where this description is irredundant.
An **extreme ray** of this cone is a point that can't be written as a conic combination of other, non-colinear rays of the cone. We often consider rays up to positive scaling.
We can characterise rays as follows: r is an extreme ray $\Leftrightarrow$ $Ar \leqslant 0$ and for $eq(r) = \{i : A_{i*}r = 0\}$, the space $A_{eq(r)*}x = 0$ is 1-dimensional, aka. $rank(A_{eq(r)*}) = d-1$. In particular, there are at most $\binom{m}{d-1}$ extreme rays.
Also, the extreme rays form the unique minimal and minimum generating set of the cone in V-description.

To prove the properties of extreme rays, we need the following lemma:

Rays:

A face F of the cone of dimension $k \geq 2$ given by $\begin{cases} Ax \leqslant 0 \\ A_{I*}x = 0 \end{cases}$ for some $I \subseteq [m]$ contains (at least) k

linearly independent rays/points r_i such that for $k-1$ of them $I \subseteq eq(r_i)$ and $rank(A_{I*}) < rank(A_{eq(r_i)*})$. Also, any ray $r \neq 0$ on the face of dimension $\geqslant 2$ for which $I = eq(r)$ can be written as a conic combination of two rays of the face r_1, r_2 for which $eq(r) \subseteq eq(r_i)$ and $rank(A_{eq(r)*}) < rank(A_{eq(r_i)*})$.

Proof: Indeed, we take a point $r \neq 0$ on F, which we complete to a basis of $A_{I*}x = 0$ with $b_2, ..., b_k$. These basis vectors may not be on the face, but we can use them to get points on the face by considering $r_i = r + s_i b_i$ for some well-chosen s_i. Then $r, r_2, ..., r_k$ still will be linearly independent, but now on the face. It turns out that it will be more advantageous for the rest of the proof if we choose none of the b_i to be on F: this can be done by taking their opposite/antipodal if they are in F, as this point won't be in F, which is pointed as part of a pointed cone.
To get $Ar_i \leqslant 0 \Leftrightarrow s_i Ab_i \leqslant -Ar$, knowing that $A_{I*}b_i = 0$ and there is some j for which $A_{j*}b_i > 0$, we can set $s_i = \min\left(\frac{(-Ar)_j}{(Ab_i)_j} : (Ab_i)_j > 0\right) \geqslant 0$ as $0 \leqslant -Ar$ to get the job done: then $s_i Ab_i \leqslant -Ar$ will be valid on lines I as they're both zero, on lines for which $A_{j*}b_i \leqslant 0$ as $s_i \geqslant 0$ and $0 \leqslant -Ar$, and finally on the lines for which $A_{j*}b_i > 0$, by definition of s_i.
Note that for the latter points, at the line j where $\min\left(\frac{(-Ar)_j}{(Ab_i)_j} : (Ab_i)_j > 0\right)$ is attained, $A_{j*}r_i = 0$, so that $I \subseteq eq(r_i)$. Furthermore, A_{j*} can't be spanned by the other rows of I since $A_{I*}b_i = 0$ but $A_{j*}b_i > 0$. So for these r_i, $rank(A_{I*}) < rank(A_{eq(r_i)*})$.

For the second part of the lemma, we do a similar construction. The difference is that we slide r along a vector $v \neq 0$ of $A_{I*}x = 0$, such that nor v nor $-v$ are in the cone. Showing the existence of such a vector is surprisingly a bit difficult. We consider the hyperplane defining the origin as a face: $\left(\sum_{i\in[m]} A_{i*}\right) x = 0$, so that for all points in the cone $\left(\sum_{i\in[m]} A_{i*}\right) x \leqslant 0$, with equality only at the origin.
We pick any $v \neq 0$ on the intersection of that hyperplane with $A_{I*}x = 0$ (which can be done as $A_{I*}x = 0$ has dimension greater then 1): then $-v \neq 0$ is also on the intersection, and none of them can be in the face. Now, we consider $r_1 = r + s_1 v$ with $s_1 = \left(\min\left(\frac{(-Ar)_j}{(Av)_j} : (Av)_j > 0\right)\right)$ and $r_2 = r + s_2(-v)$ with $s_2 = \left(\min\left(\frac{(-Ar)_j}{(-Av)_j} : (-Av)_j > 0\right)\right)$, which by similar arguments to the first part of the proof have the desired properties of the lemma. To see that r is a conic (even convex) combination of them, note that $r = \frac{s_2}{s_1+s_2} r_1 + \frac{s_1}{s_1+s_2} r_2$, if $s_1 + s_2 > 0$. The case $s_i = 0$ occurs if $(Ar)_j = 0$ for some $j \notin I$, but by assumption $I = eq(r)$, so this can't happen.

We can now turn to the properties of extreme rays. If $A_{eq(r)*}x = 0$ is 1-dimensional, then for any conic combination $r = \sum \lambda_i x_i$ where the x_i are in the cone and $\lambda > 0$, since the x_i must be in $A_{eq(r)*}x = 0$ (otherwise $A_{j*}r < 0$ for some $j \in eq(r)$), the x_i have form $x_i = \mu_i r$, where $\mu \geqslant 0$ (compute and rewrite $A_{j*}x_i$ for any $j \notin eq(r)$). So r is a extreme ray. Conversely, we use the contrapositive: if the $A_{eq(r)*}x = 0$ dimension $\geqslant 2$, then so does the associated face by the previous lemma, and we can use the points r_1, r_2 of the previous lemma to write r as a conic combination of non-colinear points (if they where colinear, so would r and v be, but v is in $\left(\sum_{i\in[m]} A_{i*}\right) x = 0$ while r isn't, and both are non-zero), so that r isn't an extreme ray. In particular, the extreme rays are determined by $A_{I*}x = 0$ for $rank(A_{I*}) = d - 1$, so that we can check all $I \subset [m]$ of size $d-1$ for this property to find the extreme rays.
We can reiterate this decomposition on the terms from the lemma, as long as the terms are not extreme

rays. At each iteration k, the rank of $A_{eq\left(r^{(k)}\right)*}$ increases, so that it will reach d or $d-1$, the first case corresponding to the origin and the second to an extreme ray. Since the minimum rank of the matrices associated to the eq of the terms increases at each iteration, there are at most d iterations, so that the decomposition will eventually reach a sum with terms made of extreme rays only.
Finally, let's prove that the extreme rays form a minimum and minimal generating set of the cone. Any generating set has to contain the extreme rays, as they generate the extreme rays in particular, which can only be generated by themselves. Next, the extreme rays generate the cone, as any ray can be written as a conic sum of them, by the previous paragraph.

Adjacency of rays:

We consider two extreme rays r_1, r_2 to be **adjacent** if the minimal face of the cone containing both, given by $A_{(eq(r_1)\cap eq(r_2))*}x = 0$, contains no other extreme ray, or equivalently, has dimension two.

Proof: By the characterization of faces, any face has form $\begin{cases} Ax \leqslant 0 \\ A_{I*}x = 0 \end{cases}$. If it contains r_1 and r_2, then we must have $I \subseteq eq(r_i)$ for both and so $I \subseteq eq(r_1) \cap eq(r_2)$. Therefore $\left\{ x : \begin{cases} Ax \leqslant 0 \\ A_{(eq(r_1)\cap eq(r_2))*}x = 0 \end{cases} \right\} \subseteq \left\{ x : \begin{cases} Ax \leqslant 0 \\ A_{I*}x = 0 \end{cases} \right\}$, and since the first set is a face by the characterization of faces, it's the minimal face containing r_1 and r_2.
We show the equivalence of notions. If the face has dimension two, then r_1 and r_2 generate it, as they from an affinely independent triple with the origin (distinct extreme rays aren't colinear) and any point on the face must actually be a conic combination of the two, since $eq(r_1) \nsubseteq eq(r_2)$ and $eq(r_2) \nsubseteq eq(r_1)$(otherwise they belong to the same one dimensional space due to the characterisation of extreme rays), so that in $x = \lambda_1 r_1 + \lambda_2 r_2$, by picking $i \in eq(r_1) \backslash eq(r_2)$ and $j \in eq(r_2) \backslash eq(r_1)$, and computing $A_{i*}x \leqslant 0$ and $A_{ji*}x \leqslant 0$, we get $\lambda_1, \lambda_2 \geqslant 0$ respectively. Hence, any other potential extreme ray on the face would be a conic sme of these two, contradicting its nature.
Conversely, using the contrapositive, if the minimum face has dimension 3 or more, then by the previous lemma, it also contains 3 linearly independent rays on it. They decompose into conic sums of extreme rays, which also have to be on the face. We'll show that 3 or more extreme rays are necessary in the decompositions to satisfy linear independence. If the points were $x = x_1 r_1 + x_2 r_2$, $y = y_1 r_1 + y_2 r_2$ and $z = z_1 r_1 + z_2 r_2$, then by linear independence, x, y generate $span(r_1, r_2)$, so also z, which is supposed to be independent from them. So a third extreme ray is necessary on the face, hence the rays aren't adjacent.

9.3 Polarity/Duality

Definition:

The **dual** or **polar** of a cone C is defined by $C^* = \{y : y^t x \leqslant 0, \forall x \in C\}$. It's also a cone and represents all (vectorial) hyperplanes valid for the cone.

The fact that it's a cone is shown by an interplay of V and H descriptions that often appears in duality theory. If $y^t x \leqslant 0, \forall x \in C$, then in particular for the finitely many generators c_i of C, $y^t c_i \leqslant 0$. Conversely, if $y^t c_i \leqslant 0$ for all generators, since the sign is preserved under conic combinations, $y^t x \leqslant 0, \forall x \in C$. Then the $y^t c_i \leqslant 0$ are an H-description of a cone.

Why it's called dual:

$$C^{**} = C$$

Proof: Note that for $x \in C$, $x^t y \leqslant 0$ for all $y \in C^*$, by definition of C^*: this statement is precisely the definition of C^{**}, so that $C \subseteq C^{**}$. If there was a $z \in C^{**}$ not in C, Farkas provides us with $h^t z > 0$ and $h^t C \leqslant 0$. Then $h \in C^*$ so that $z^t h = h^t z > 0$ prevents $z \in C^{**}$, hence z must be in C.

Explicit duals:

If $C = \{B\lambda : \lambda \geqslant 0\}$, then $C^* = \{a : a^t B \leqslant 0\}$.
If $C = \{x : Ax \leqslant 0\}$, then $C^* = \{A^t \lambda : \lambda \geqslant 0\}$.

Proof: For the first one, note that if $a^t B \leqslant 0$ then $a^t(B\lambda) \leqslant 0$ for all $\lambda \geqslant 0$, so that $a \in C^*$ and conversely, if $a \in C^*$ then $a^t B \leqslant 0$ by applying the definition with $\lambda = e_i$. For the second one, we'll write $C = \{x : x^t A^t \leqslant 0\}$ to see that $C = D^*$ for $D = \{A^t \lambda : \lambda \geqslant 0\}$ and use the fact that $C^* = D^{**} = D$.

If we consider the dual cone of the homogenisation of a polyhedron P, then for all its vectors y we have $y^t(x, 1) \leqslant 0, \forall x \in P$. If we treat this dual cone like the homogenization of a polyhedron, then we can consider the intersection with $y_{d+1} = 1$ of this cone, which gives points y such that $y^t x \leqslant -1, \forall x \in P$. However, since the homogenization cones are in $x_{d+1} \geqslant 0$, and $(0, -1)$ satisfies $y^t(x, 1) \leqslant 0, \forall x \in P$, so that the cone of the y contains parts of $y_{d+1} \leqslant 0$, it makes more sense to flip things around and consider the intersection with $y_{d+1} = -1$.

Definition:

The **dual** of **polar** of a polyhedron P is defined by $P^* = \{y : y^t x \leqslant 1, \forall x \in C\}$.
It's also a polyhedron.

It can be thought of as the set of valid hyperplanes for P, in the case that $0 \in int(P)$, since $c^t x \leqslant d$ for all P and in particular for 0 we get $0 < d$, then $y = \frac{1}{d} c^t \in P^*$. We have a little difference:

Why it's called dual:

If $0 \in P$, then $P^{**} = P$

Proof: Again, for $x \in P$, $x^t y \leqslant 1$ for all $y \in P^*$, so that $P \subseteq P^{**}$. For the converse, we use H-homogenization, though a dual one: if P has homogenization H , the $x \in P \Leftrightarrow \begin{pmatrix} x \\ -1 \end{pmatrix} \in H$. For example if we have description $conv(V) + cone(R)$ for the polyhedron, the homogenization is $cone\left(\begin{pmatrix} V & R \\ -1^t & 0^t \end{pmatrix}\right)$. Assume for contradiction that there was a $z \in P^{**}$ not in P, then $\begin{pmatrix} z \\ -1 \end{pmatrix}$ isn't in the cone H and we can use strict Farkas so as to find $\left(h^t, h_{d+1}\right)$ such that $\left(h^t, h_{d+1}\right)\begin{pmatrix} z \\ -1 \end{pmatrix} > 0$ and $\left(h^t, h_{d+1}\right) H \leqslant 0$. Since $0 \in P \Leftrightarrow \begin{pmatrix} 0 \\ -1 \end{pmatrix} \in H$, the second inequality gives $h_{d+1} \geqslant 0$, and also that for all $x \in P$, we have $h^t x \leqslant h_{d+1}$. To proceed, we actually need $h_{d+1} > 0$. Rewriting the inequalities as $h^t z > h_{d+1}$ and $h^t(V, R) \leqslant |V| h_{d+1}$. If $h_{d+1} = 0$, we can consider $h' = \frac{1}{h^t z} h$ and $h'_{d+1} = \frac{1}{2} > 0$ to get $h'^t z = 1 > h'_{d+1}$ and $h'^t(V, R) = \frac{1}{h^t z} h^t(V, R) \leqslant 0 \leqslant h'_{d+1}$. So we have $h'^t x \leqslant h'_{d+1}$ for $x \in P$ and $h'^t z > h'_{d+1}$, with $h'_{d+1} > 0$. By considering $h'' = \frac{1}{h'_{d+1}} h'$, we see $h''^t x \leqslant 1$ for all $x \in P$ so that $h \in P^*$. Yet the first inequality, which rewrites to $h''^t z > 1$ then contradicts $z \in P^{**}$.

Dual property:

If $F \subseteq G$, then $G^* \subseteq F^*$.

Proof: If $F \subseteq G$, the for $y \in G^*$, we have $y^t x \leqslant 1$ for all $x \in G$, and in particular for all $x \in F$ so that $y \in F^*$.

We can interpret the dual as follows:

Duality of faces:

For a face $\emptyset \subsetneq F \subsetneq P$, and a polyhedron such that $0 \in int(P)$, consider the set of hyperplanes definig that face F, in the sense $F^\delta = \left\{ c : \begin{cases} c^t x \leqslant 1, \forall x \in P \\ c^t x = 1, \forall x \in F \end{cases} \right\}$. F^δ is a face of P^*.

Proof: Indeed, any $x \in F$ defines the face F^δ of P^* since $x^t c \leqslant 1, \forall c \in P^*$ since $x \in P$ and $x^t c = 1, \forall c \in F^\delta$.

9.4 Integer Polyhedra

Pfaffenhofer and IPintegerHull

9.5 Polyhedral decompositions

Factoring, to tell if an LP is decoupled. "PolyhedraFacto" in folder.
An example is when we can find a change of variabes P so that we can split the dimensions in two via $Px = \begin{pmatrix} y \\ z \end{pmatrix}$, and so tha for the polyhedron $Ax \leqslant b$, we have $AP^{-1}\begin{pmatrix} y \\ z \end{pmatrix} \leqslant b$. If we can find positive matrices $M, N \geqslant 0$ such that $MAP^{-1} = \begin{pmatrix} C & 0 \\ 0 & D \end{pmatrix}$ and $N\begin{pmatrix} C & 0 \\ 0 & D \end{pmatrix} = AP^{-1}$ and $Mb = \begin{pmatrix} c \\ d \end{pmatrix}$ and $N\begin{pmatrix} c \\ d \end{pmatrix}$, then the polyhedron $AP^{-1}\begin{pmatrix} y \\ z \end{pmatrix} \leqslant b$ can be represented as the product of $Cy \leqslant c$ and $Dz \leqslant d$. If we then split the linear cost function $f^t P^{-1}$, we see that an LP on $Ax \leqslant b$ can be decoupled into smaller ones.
Cartesian Factoring of Polyhedra in Linear Relation Analysis by Nicolas Halbwachs, David Merchat, Catherine Parent-Vigouroux
Judith Beestermoeller, Decomposing Polyhedra as Cartesian Products. Maserthesis Advisor: B. Gärtner, E. Welzl, 2021
Also paper by Henk on decomposing MIPs into Minkovski sums

9.6 Solutions

Ex.F1: All we have to show is that they are stable under conic combinations: for normal vectors n_i corresponding to halfspaces given by $n_i \cdot x \leqslant b_i$,, which P is contained in and for which the inequality is strict on a face F, the conic combination yields $\left(\sum s_i n_i\right) \cdot x = \sum s_i (n_i \cdot x) \leqslant \sum s_i b_i$ by positivity of the scalar. So P is in the halfspace defined by the combination and on F, the previous equality is a equality, so that the combination defines the face F. □

10 Linear Programming

10.1 Introduction and examples

Transportation: We which to send a product from a set of factories to a set of stores for a minimum transportation cost. If we index factories with $i \in [n]$ and stores with $j \in [m]$, we can let p_i denote the production capacity of factory i, s_j denote the storage capacity, and c_{ij} be the cost of transporting a unit of product from factory i to store j. We can then decribe the transportation with variables $x_{ij} \in \mathbb{R}$ to denote the quantity of product shipped from factory i to store j (assume the product can be subdivided indefinitly, like icecream, so that a real variable is suitable for the model). We then which to solve the constrained minimisation problem $\min \left(\sum_{ij} c_{ij} x_{ij} \right)$ st. $\begin{cases} x \geqslant 0 \\ \sum_j x_{ij} = p_i \\ \sum_i x_{ij} = s_j \end{cases}$

Regression: In statsitics, the problem of polynomial regression consists in finding an (multivariable) polynomial $P(x) = \sum_{\sum p_i \leqslant d} w_p \cdot x^p + w_0$ that estimates a real-valued parameter as a function of other parameters in $\mathbb{R}^n$, given a set of observed data $(x_i, y_i)_{i \leqslant m}$. The goal is to find P so that $P(x_i) \approx y_i$ for all data points.
We can get a good P by considering margins $M_i \geqslant 0$ on the difference $P(x_i) - y_i$ that we attempt to minimise:

$$\min \sum_{i=1}^{m} M_i \text{ s.t. } \begin{cases} \sum_{\sum p_i \leqslant d} w_p \cdot x_i^p + w_0 \leqslant y_i + M_i \\ \sum_{\sum p_i \leqslant d} w_p \cdot x_i^p + w_0 \geqslant y_i - M_i \\ M_i \geqslant 0, \ w \in \mathbb{R}^n, w_0 \in \mathbb{R} \end{cases}$$

Minimax, Shapely value, transform of PL-convex to LP, max radius ball (Chebycheff center).

10.2 Primal simplex algorithm

Standard LP:
As we've seen in the previous section, many problems are qualified as LPs. In order to solve them all with the same method, we'll reduce them to a standard form:

Definition:
An linear program (LP) in **standard form** is $\min(c\cdot v)$ st. $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$ for some $A \in \mathbb{R}^{m\times n}$ of full row rank $m < n$.

To get from inequality constraints to the standard form, one can introduce slack variables. In the transportation problem for example, these can be interprested as storage/waste of the product for factories: we turn condition $a\cdot x \leqslant b$ into $\begin{cases} a\cdot x + s = b \\ s \geqslant 0,\ s \in \mathbb{R} \end{cases}$. To handle $a\cdot x \geqslant b$, we write it as $(-a)\cdot x \leqslant (-b)$. To deal with a variable x that may take negative values, we split it into $x_+ - x_-$ substituting these expressions into the constraints, the additional with $x_+, x_- \geqslant 0$.
At this stage, the constraints have form $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$, but A may not have full row rank. We therefore delete the rows of $Ax = b$ that are linear combinations of others: if this leads to contradictions of type $0 = \delta$ where $\delta \neq 0$, then there are no feasible solutions to the problem. Thus, we may replace A by it's rows that form a base of the space spanned by its rows, so that its rank is $\leqslant n$. If the rank was n, then A is invertible and the problem has a unique solution, so that there's nothing to optimise.

For a solution x of the initial polyhedron with value $\sum_{i=1}^{n} c_i x_i$, we can construct a solution of the standard form problem by setting $x_{+,i} = \max(0, x_i)$, $x_{-,i} = -\min(0, x_i)$ and $s_i := b_i - A_{i*}x$, and by extending the cost through $c_{+,i} = c_i$, $c_{-,i} = -c_i$ and $c_s = 0$, this solution has the same objective value as x. Conversely, for a solution to the standard form of the problem, $x = x_+ - x_-$ becomes a feasible solution for the initial problem, with same objective value. The optimisation problems are therefore equivalent.

The big idea of the simplex algorithm:
Unless the problem is infeasible or unbounded (in the sense that we can find a sequence of feasible solutions who's value tends to $-\infty$), we know that by compactness, the optimal value c_{opt} is attained. The solutions will then form a face with defining halfspace $c\cdot x \leqslant c_{opt}$.
Since for most problems, we're satisfied from getting one optimal solution, not necessarily all of them, we can use the fact that the standard form polyhedron is pointed and that all its faces must therefore have an vertex to focus on finding an optimal vertex of the polyhedron. In fact, the optimal face may (and does with probability 1 for uniformly distributed costs) consist of just a vertex.
We've characterised the vertices of polyhedra of form $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$, which are the solutions to $\begin{cases} A_{*B}x_B = b \\ x_{[n]\setminus B} = 0, x \geqslant 0 \end{cases}$ for some subset $B \subset [n]$ of size $|B| = m$ (called a basis) so that the colums $(A_{*j})_{j\in B}$ are linearly independent, so that $x_B = A_{*B}^{-1}b$, $x_{[n]\setminus B} = 0$ and $x \geqslant 0$.
The brute force approach to solving LPs is that of testing for all $\binom{n}{m}$ choices of B whether this provides a vertex, and taking note of the minimum value $c\cdot x = c\cdot A_{*B}^{-1}b$ encountered so far, so that this will be the minimum value of the LP after having checked all possibilities. this approach has two flaws: first, it doesn't tell us if the LP is unbounded and second, it's inefficient.
An alternative (and successful) approach would be to navigate the vertices along edges or rays that lead to

cost decreases. This should shorten the vertex exploration time needed and allow us to detect rays along which the cost infinitely decreases.
This local search approach is motivated by the fact that the objective is linear and the solution space is convex. Indeed, if we arrive at a local minimum x for which $c\cdot(x+y) \leqslant c\cdot x$ for all $y \in B(0,\varepsilon)$ so that $x+y$ is in the polyhedron P, this will be a global minimum. Otherwise, if $z \neq x$ were a point of P of lower value, since the segment $[x,z]$ would be in the convex P, so would be the point $x + \frac{\varepsilon}{2\|z-x\|}(z-x)$ on it, which has value $c\cdot\left(x + \frac{\varepsilon}{2\|z-x\|}(z-x)\right) < c\cdot x$ as $c\cdot z < c\cdot x$, so that we contradict the assumption that x was a local minimum. So successive improvements until none are possible should lead us to a local and therefore global optimum.
Here, we find another advantage of focusing on vertices. It's usually hard to prove termination of local search algorithms. Here however, if we move from vertex to vertex, since there are only finitely many vertices, termination is easily guaranteed, as we can strictly improve in objective value only a finite number of times.

The simplex algorithm (without degeneracy):
We start the simplex algorithm with an initial vertex x_0 with basis B_0. To get such an initial vertex, we can start with a brute force approach as mentioned before, searching for feasibility through $\binom{[n]}{m}$. Another method for initialising the simplex algorithm will be discussed in the next section.

We search for an improved solution of form $x_0 + t.y$ for $t > 0$ (we decouple direction and magnitude). It should verify $\begin{cases} A(x_0+t.y) = b \\ x_0 + t.y \geqslant 0 \end{cases}$ and $c\cdot(x_0+t.y) \leqslant c\cdot x_0$. This is equivalent to $\begin{cases} Ay = 0 \\ t.y \geqslant -x_0 \end{cases}$ and $c\cdot y \leqslant 0$ (recall $t > 0$).
If the current solution x_0 isn't optimal, such an improvement must exist. Indeed, if there is a z with $Az = b$, $z \geqslant 0$ and $c\cdot z < c\cdot x_0$, one can take $y = z - x$ and $t = 1$.
There are many improvements possible at this stage, but finding a solution the equations characterising it we just developed is as hard as solving LPs. (If we had an efficient way of finding a point in a polyhedron or telling that it's empty, we could solve the LP approximately with binary search).
However, we're looking for a specific improvement: one that will lead us to another vertex. For $x_0 + t.y$ to be a vertex, it has to additionally have a basis B_1, so that $x_{0,i} + t.y_i = 0$ for all $i \in [n]\backslash B_1$ and the $(A_{*j})_{j\in B_1}$ are linearly independent.

The simplest improvement from vertex to vertex one can make is by moving along an edge of the polyhedron. An edge of the polyhedron is contained in at least $n-1$ hyperplanes spanning an $(n-1)$-dimensional space. For our standard form polyhedron, this means that there must be at least $n-m-1$ equations of form $x_i \geqslant 0$ that must be tight on the edge. Since the edge we seek contains x_0, the coordinates for which $x_i \geqslant 0$ is tight must be in $[n]\backslash B_0$. So we must pick a $j \in [n]\backslash B_0$ so that the edge-points verify $x_{[n]\backslash(B_0\cup j)} = 0$. To leave x_0 for another vertex along an edge, we can therefore increase $x_{0,j}$ and keep $x_{0,[n]\backslash(B_0\cup j)} = 0$. This leads us to choose y so that $y_{[n]\backslash(B_0\cup j)} = 0$ and $y_j = 1$. To stay in the polyhedron we have to adapt y_{B_0}: $Ay = 0$ will now have form $A_{*B_0}y_{B_0} + A_{*j} = 0$, and since B_0 is a basis, this uniquely defines $y_{B_0} = -A_{*B_0}^{-1}A_{*j}$.
The condition $t.y \geqslant -x_0$ is a bit more subtle. As $x_0 \geqslant 0$ and $y_{[n]\backslash(B_0\cup j)} = 0$, the coordinates in $[n]\backslash(B_0\cup j)$ add no constraints. At j, we have $t \geqslant -x_{0,j} = 0$ as j was chosen in $[n]\backslash B_0$, so this is no constraint either. At B_0 however, we have $t.y \geqslant -x_0 \Leftrightarrow t.A_{*B_0}^{-1}A_{*j} \leqslant x_{0,B_0}$.

Here multiple things may happen. If all coordinates of $A_{*B_0}^{-1}A_{*j}$ are negative, then no constraints are added,

as $t > 0$ and $x_0 \geqslant 0$. Note that in this case, we can let $t \to \infty$ and still be on the edge: this means that we are on a ray of the polyhedron. Otherwise, a constraint is added: $t \leqslant \min_{k \in B_0,\ \left(A_{*B_0}^{-1} A_{*j}\right)_k > 0} \left(\frac{x_{0,k}}{\left(A_{*B_0}^{-1} A_{*j}\right)_k} \right)$.

In this case, we expect $x_1 := x_0 + \min_{k \in B_0,\ \left(A_{*B_0}^{-1} A_{*j}\right)_k > 0} \left(\frac{x_{0,k}}{\left(A_{*B_0}^{-1} A_{*j}\right)_k} \right) . \left(-A_{*B_0}^{-1} A_{*j}\right)$ to be a vertex: the other endpoint of the edge. To prove that this is indeed a vertex, we use our characterisation, in the sense that we'll find a basis B_1 for it. We have $x_{1,[n]\backslash(B_0 \cup j)} = 0$ by construction, so $B_1 \subset (B_0 \cup j)$. Now if k is the index in which the minimum $\min_{k \in B_0,\ \left(A_{*B_0}^{-1} A_{*j}\right)_k > 0} \left(\frac{x_{0,k}}{\left(A_{*B_0}^{-1} A_{*j}\right)_k} \right)$ is attained, then at it $t . \left(A_{*B_0}^{-1} A_{*j}\right)_k = x_{0,k}$, or equivalently $x_{1,k} = 0$. This makes all indices attaining $\min_{k \in B_0,\ \left(A_{*B_0}^{-1} A_{*j}\right)_k > 0} \left(\frac{x_{0,k}}{\left(A_{*B_0}^{-1} A_{*j}\right)_k} \right)$ candidates for leaving B_0, so that we could set $B_1 = (B_0 \cup j)\backslash k$. Is such a $B_1 = (B_0 \cup j)\backslash k$ a basis of x_1 ?
The only basis requirement that isn't obtained by construction is that the $(A_{*i})_{i \in B_1}$ are linearly independent.
If we write $\sum_{B_1} s_i A_{*i} = 0$, we can get a simpler form of this expression by multiplying by $A_{*B_0}^{-1}$, as it then becomes $\sum_{B_0 \backslash j} s_i e_i = -s_j A_{*B_0}^{-1} A_{*j}$. By construction $y_j = 1$, so that $s_j = 0$, and then we directly get the rest $s_i = 0$. So the $(A_{*i})_{i \in B_1}$ are truly linearly independent, and therefore B_1 is a basis of the vertex x_1.

There's a pathological case we need to discuss: the case in which $\min_{k \in B_0,\ \left(A_{*B_0}^{-1} A_{*j}\right)_k > 0} \left(\frac{x_{0,k}}{\left(A_{*B_0}^{-1} A_{*j}\right)_k} \right)$. This doesn't affect any of the previous reasoning's, but it means that the basis B_0 and B_1 correspond to the same vertex. This is called degeneracy, and it occurs precisely if there's an index k with $k \in B_0$, $\left(A_{*B_0}^{-1} A_{*j}\right)_k > 0$ and $x_{0,k} = 0$, that is to say, that x_0 was in more then n hyperplanes, aka. more then $n - m$ inequalities of form $x_i \geqslant 0$ are tight.
We'll assume that this case is never encountered in the algorithm for now, and handle this specific phenomenon in a later section.

Now that we know how to move from vertex to vertex or detect rays, we'll discuss the change in objective. We were looking for $c \cdot y < 0$, which is now $c_j - c_{B_0} A_{*B_0}^{-1} A_{*j} < 0$. If we can find a $j \in [n]\backslash B_0$ for which this is the case, then we may proceed with two cases. Either, the magnitude of the increase t is finite so that $c \cdot x_1 = c \cdot (x_0 + t.y) < c \cdot x_0$ and we'll successfully decreased the cost by moving to a different vertex, or we can let $t \to \infty$ so that the LP is unbounded.
If however for all $j \in [n]\backslash B_0$, $c_j - c_{B_0} A_{*B_0}^{-1} A_{*j} \geqslant 0$, then we expect there to be no vertex of lower value. If there was a better solution z, then for $d = z - x_0$, we have $c \cdot d < 0$ which we can split into $c_{B_0} \cdot d_{B_0} + c_{[n]\backslash B_0} \cdot d_{[n]\backslash B_0} < 0$ for further analysis. Then we can rewrite the constraint $0 = Ad = A_{*B_0} d_{B_0} + A_{*[n]\backslash B_0} d_{[n]\backslash B_0}$ by multiplying by $A_{*B_0}^{-1}$, so that $d_{B_0} = -A_{*B_0}^{-1} A_{*[n]\backslash B_0} d_{[n]\backslash B_0}$. This yields $\left(c_{B_0} \left(-A_{*B_0}^{-1} A_{*[n]\backslash B_0}\right) + c_{[n]\backslash B_0}\right) \cdot d_{[n]\backslash B_0} < 0$ by substitution. The coordinates of the vector on the left side of the dot-product have form $j \in [n]\backslash B_0$, $c_j - c_{B_0} A_{*B_0}^{-1} A_{*j} \geqslant 0$ while those on the right have form $z_{[n]\backslash B_0} - x_{0,[n]\backslash B_0} = z_{[n]\backslash B_0} \geqslant 0$ as B_0 is a basis of x_0 and z is feasible. So the dot-product should be positive, a contradiction to the fact that z is a better solution.
So we can finally conclude that when no $j \in [n]\backslash B_0$ with $c_j - c_{B_0} A_{*B_0}^{-1} A_{*j} < 0$ exists, x_0 is an optimal vertex, and our algorithm terminates.

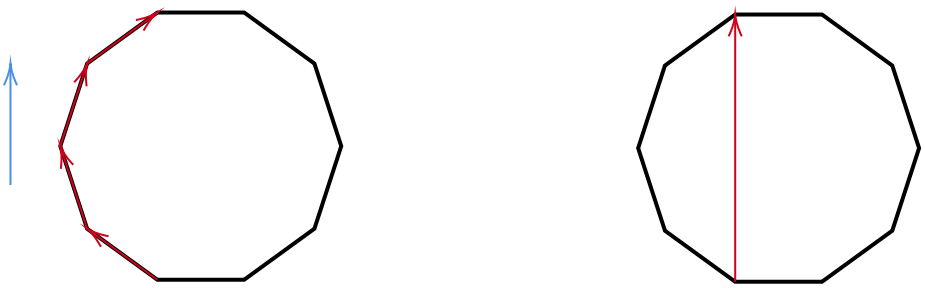
Initialising the simplex algorithm:
In some cases, we can obtain a vertex and a basis from inspection of the problem. For example, for LPs on polyhedra of the form $Ax \leqslant b$ with $b \geqslant 0$, the standard form is $\begin{cases} (A, -A, I) \begin{pmatrix} x_+ \\ x_- \\ s \end{pmatrix} = b \\ x_+, x_-, s \geqslant 0 \end{cases}$, where full row rank is obtained automatically from the slack variables. Then the slack variables form a basis and the corresponding vertex is $(0, 0, b)$, which is feasible as $b \geqslant 0$.

But in the general case, we must use a more general approach. It turns out that on can solve the feasibility problem of a standard form polyhedron (is it empty ?) with an LP. The idea is that of introducing error variables that we which to minimise: for $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$, we introduce the error variable y and investigate $\begin{cases} Ax + y = b \\ x \geqslant 0 \end{cases}$. For a feasible point x of the standard problem, setting $y = 0$ produces a feasible point to this auxiliary problem. Then, by requiring $y \geqslant 0$ and minimising $1 \cdot y$, we're solving an LP on a polyhedron in standard form: $\min(1 \cdot y)$ st. $\begin{cases} Ax + y = b \\ x, y \geqslant 0 \end{cases}$. By positivity of y, this minimum positive and it is precisely 0 if the $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$ is feasible.
At this stage, it may seem we#re going in circles: why try initiating an LP by solving another one. The reason is that $\begin{cases} Ax + y = b \\ x, y \geqslant 0 \end{cases}$ has an obvious vertex that we can always start the simplex method with. It's closely related to the previous trick: we first multiply rows of $Ax = b$ by -1 in order to have $b \geqslant 0$, if necessary. Then $x = 0, y = b$ is a vertex of $\begin{cases} Ax + y = b \\ x, y \geqslant 0 \end{cases}$ with basis the error variables.
Once we've solved $\min(1 \cdot y)$ st. $\begin{cases} Ax + y = b \\ x, y \geqslant 0 \end{cases}$ and found a solution of value 0 with the simplex algorithm (otherwise, the initial LP is infeasible), we have to ask if the x of this optimal solution is a vertex of the initial problem, as the simplex algorithm requires a vertex to start. The indices for which $x_i > 0$, the support S, (if there are any) must be in the basis. In order to get a basis with only indices of non-error variables, we have to add indices corresponding to $x_i = 0$ to S, forming B, so that $(A_{*j})_{j \in B}$ is a basis of $\mathbb{R}^m$. Since S was in the basis of the auxiliary problem, the $(A_{*j})_{j \in S}$ were linearly independent. As A has full rank m, we can complete the $(A_{*j})_{j \in S}$ into a basis using more columns of A. This can be done relatively efficiently, for example by successively adding indices to the base B, starting from S, by checking for linear independence by solving linear systems of form $A_{*B} z = 0$ at each iteration, so at most n times.

The simplex algorithm in tableau form:

A trick for potentially speeding up the simplex method (in special cases):
One reason why the simplex method can take a long time before terminating is that it may visit many vertices of the polyhedron. For example, on the left of the figure below, we sketch the steps of the algorithm,

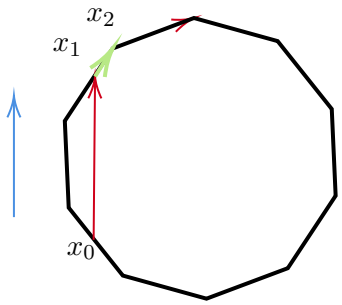
in red, on a 10-gon with height (blue direction) as objective.



What if, instead of following the boundary of the polyhedron, we first moved in the best increasing direction there is, which is that of the cost c (for a maxmization problem) ? In the figure above on the rigth, we see that only one such augmentation would have been necessary, so this seams like a reasonable idea.

We'll assume that we're at a point x_0 of the polyhedron $Ax \leqslant b$, which doesn't have to be a vertex. If no such point is available, we can find one by solving a auxiliary problem as in phase 1 of the simplex method, or determine infeasibility. We then find the largest t so that $x_0 + tc$ is still in the polyhedron, which is $\max\limits_{i:A_{i*}c>0} \left(\dfrac{b_i - A_{i*}x}{A_{i*}c} \right)$. If this maximum is infinite, in the sense that for all i we have $A_{i*}c \leqslant 0$, then we know the LP is unbounded, as the cost increases in $t\|c\|^2 > 0$ along the ray. Otherwise, the corresponding point x_1 is a point of the polyhedron, which now may not be a vertex, as was guaranteed by the simplex algorithm.

Now, what do we do from x_1 ? The way to recycle our idea by merging it to the simplex algorithm, is to find a vertex close to x_1 and continiue with the simplex algorithm. As the figure below shows, if we start from a different x_0, we observe that x_1 need not be either optimal, nor a vertex. However, if we find a method to find a close vertex, as we indicate in a green step to be clarified, we can continiue with the simplex method. This initial step still has the advantage that it can conceptually speed up the algorithm.



Now, it only makes sense to find a vertex if the polyhedron $Ax \leqslant b$ has any. We therefore assume that the polyhedron has 0-dimensional lineality space, which is a condition for our alogorithm to work. We then find a vertex near x_1 iteratively, as follows. At each step, an intermediate point x_k will be in a face given by $\begin{cases} Ax \leqslant b \\ A_{I_k*}x = b \end{cases}$ for sets of indices $I_1 \subset I_2 \subset ....$ To get the next point, starting from x_1, with I_1 being the indices of tight constraints, at each iteration, we take any non-zero direction v satisfying $A_{I_k*}v = 0$, which exist until I_k has as many elements as the dimension, in which case $\begin{cases} Ax \leqslant b \\ A_{I_k*}x = b \end{cases}$ describes a vertex already, and we've achieved our goal. We then seek the largest t such that $x_k + tv$ is still in the polyhedron. If $x_k + tv$ is in the polyhedron, for any t ($A_{i*}v \leqslant 0$ for all i), then we do the same for $-v$. This is where the assumption on the lineality space comes into play. Since the polyhedron is pointed by assumption, it can't contain an entire line, so that there must be a largest t such that $x_k + t(-v)$ is in the polyhedron.

For efficiency, if we detect a ray in direction v at this step, we can check if $c^t v > 0$, in which case we can stop immediately and return that the LP is unbounded. With this choice of v, we have no guarantee that the objective has improved at the next point. All we know is that the next point is in a face of lower dimension, as we set $I_{k+1} = I_k \cup \left\{ argmax_{i:A_{i*}v>0} \left(\frac{b_i - A_{i*}x}{A_{i*}v} \right) \right\}$. These iterations will terminate once a vertex is reached. This vertex will be close to x_1 in the sense that they are on a common facet.

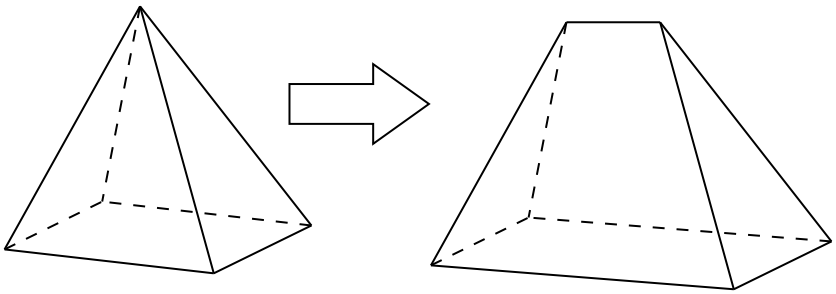
We'd like to point out that this first step fails to be useful in case the polyhedron isn't full dimensional, or at least that the objective c isn't parallel to the affine hull of the polyhedron. The first step would then yield $t = 0$.

As a concluding remark, we note that this trick doesn't improve the simplex method in the worst case scenario. Indeed, if x_0 is a vertex and c is parallel to an edge at x_0, then our algorithm coincides with a step of the simplex algorithm.
Also, we need to compare the speed of pivoting in the simplex method to that for solving systems of form $A_{I_k*}v = 0$ to be able to conclude with a performance improvement.

10.3 Anti-cycling rules

Degeneracy corresponds to the case where there are multiple basis representing a vertex, meaning that there are more then n hyperplanes the vertex is in, each basis corresponding to an n-tuple of these hyperplanes.

The first idea proposed to deal with this is to perturb the hyperplanes by a small amount. Indeed, for the case of an Egyptian pyramid, if we move one of hyperplanes touching at the top "outwards" of the pyramid, then the following happens:



We see that the vertices aren't degenerate anymore, as they all correspond to exactly 3 hyperplanes they're in. The hope is that for a small enough movement, we can fix possible infeasibility wrt. the initial polyhedron, while keeping an almost optimal solution.

We therefore perturb the initial standard form polytope $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$ to $\begin{cases} Ax = b + p \\ x \geqslant 0 \end{cases}$ (in the form of the initial tableau) for some small vector p to be chosen later. By the tableau form of the simplex algorithm, we see that pivoting will cause future tableaus ta carry the perturbation only in the first column (the b-column). For a vertex to be non degenerate, the b-column should never contain a 0 in its entries, in any of the tableaus. A clever way to do this is by letting $p = (\varepsilon, ..., \varepsilon^m)^t$ where we imagine ε being small, but treat expressions as polynomials in ε. The invariant we then want to keep from tableau to tableau is that the b-column doesn't contain the 0-polynomial. Since the initial tableau is $(b + p, A)$, this is true since $b_i + \varepsilon^i$ isn't the zero polynomial.
When we pivot, we have to find the polynomials we can consider to be "positive" and find the "smallest" among them. To be able to tell this, we let $\varepsilon \to 0$ and compare limits. The polynomial $f(\varepsilon)$ will be considered positive if for $i = \min(j : f_j \neq 0)$, $f_i > 0$ and for polynomials $f(\varepsilon)$ and $g(\varepsilon)$, we'll consider $f > g$ if for $i = \min(j : f_j \neq g_i)$, $f_i > g_i$. This will be the bridge to the regular regular simplex algorithm, as at each pivot, we can assume $\varepsilon > 0$ small enough so that the pivot is the same as for the simplex algorithm with actual numerical values.

The invariant we seek was true for the initial tableau, but is it maintained through pivoting ? This is where the clever choice of p comes in: we'll show that the b-column is made up of linearly independent polynomials in each tableau. In particular, none of them can then be the 0-polynomial. This is true for the initial tableau, as the $b_i + \varepsilon^i$ are linearly independent. Pivoting affects then b-column by adding multiples of polynomials. If the initial b-column was made up of polynomials $f_k(\varepsilon)$ and we pivot on row i, then the next row has polynomials of form $f_k(\varepsilon) + s_k f_i(\varepsilon)$ and $s_i f_i(\varepsilon)$ (where $s_i \neq 0$) so that linear independence is preserved (check that a linear combination that is 0 has 0 coefficients).
We can therefore apply the regular simplex method to this perturbed problem, and never obtain a degenerate basis. The simplex method will therefore never cycle and terminate. At termination, we let $\varepsilon = 0$ to recover an optimal tableau: the tableau will indeed be optimal, as the costs were never affected by the perturbation: only the optimal value was.

The lexicographic rule:
This first approch we gave has develloped into the notion of lexicographic rules. They are a formal way of avoiding to dicuss the whole polynomial in ε stuff. We now present an advanced such exicographic rule.

Definition:
We say that vector v is **lexicographically** greater or equal then w, written $v \geq_L w$ if the first index on which $v_i \neq w_i$, we have $v_i > w_i$. The name come from the fact that if we associate numbers 0 to 25 to the alphabet, a dictionary has words ordered in lexicographically increasing order. For a strict version, we get $v >_L W$ when $v_1 > w_1$.

The **lexicographic pivoting rule** consists choosing, for an entering index s, the leaving index i so that the row i is the lexicographic minimum among the normalised rows $\left(\frac{(b_i, A_{i*})}{a_{is}}\right)_{i\in[m]}$.

Remark: An important property the lexicographic order is that it can break ties: either $v >_L w$, or $v <_L w$ or $v = w$ (as if we aren't in the two firs cases, all entries must be equal). In particular, among a list of different vectors, there exist unique lexicographic extrema. This implies that the minimum in the lexicographic rule is unique, so that all other normalised rows are strictly lexicorgraphically greater. Indeed, no two rows in a tableau can be equal, as we assume the initial tableau to have full rank, a rank that pivots preserve and the presence of two equal rows would decrease.
This rule doesn't affect the fact that the simplex method goes through bases of increasing cost. It only brakes ties when pivoting.

Proposition:
Pivoting under the lexicographic rule maintains lexicographic positivity of rows.
It also causes costs (z, c) to strictly lexicographically increase, if the rows were lexicorgraphically poisitve prior to pivoting. This means that a basis (not necessarily its associated vertex) may appear at most once in the sequence of tableaus obtained by pivoting, as a basis determines the reduced costs.

Proof: Starting from a tableau for which rows $(b_i, A_{i*}) >_L 0$ for all lines i, we start pivoting by finding the unique lexicographic minimum of the $\frac{(b_i, A_{i*})}{a_{is}}$ for which $a_{is} > 0$, where s is the entering variable, or conclude that the LP is unbounded. If this minimum is achieved in row i, then for the other rows j, pivoting produces rows $(b_j, A_{j*}) - \frac{a_{js}}{a_{is}}(b_i, A_{i*})$ for the case when $a_{js} > 0$ and $j \neq i$. We can write this as $a_{js}\left(\frac{(b_j, A_{j*})}{a_{js}} - \frac{(b_i, A_{i*})}{a_{is}}\right)$ where we see the origin of the lexicographic rule: by having $\frac{(b_j, A_{j*})}{a_{js}} >_L \frac{(b_i, A_{i*})}{a_{is}}$ guaranteed (and $a_{js} > 0$), we have $(b_j, A_{j*}) - \frac{a_{js}}{a_{is}}(b_i, A_{i*}) >_L 0$.
For the case $a_{js} < 0$ and $j \neq i$, we can use of the fact that $-\frac{a_{js}}{a_{is}} > 0$ and both (b_j, A_{j*}) and (b_i, A_{i*}) where assumed to be lex-positive, so that $(b_j, A_{j*}) - \frac{a_{js}}{a_{is}}(b_i, A_{i*}) >_L 0$ here as well. Finally, line i becomes $\frac{(b_i, A_{i*})}{a_{is}} >_L 0$ as $a_{is} > 0$ and $(b_i, A_{i*}) >_L 0$. So lex-positivity is maintained.
The costs become $(z, c) - \frac{c_s}{a_{is}}(b_i, A_{i*})$ where $c_s < 0$ by choice of the entering variable, so that $-\frac{c_s}{a_{is}}(b_i, A_{i*}) >_L 0$ and therefore $(z, c) - \frac{c_s}{a_{is}}(b_i, A_{i*}) >_L (z, c)$, so that costs do lex-increase.

The problem at this stage is that the initial tableau of the simplex algorithm may not be lex-positive. In our definition of lexicography, we used the natural order of indices of variables (/dimension?) to determine the lexicographic order of vectors. However, we could have permuted that order of comparison, without affecting any of the development that followed. If we then remark that the basic variables can be ordered so that their columns for the identity matrix in any tableau, then we can take the initial tableau permute its columns (variables) so that the first part of the tableau forms this identity matrix, putting the b-column at the back . Then, the tableau will be lex-positive, as the first non-zero entries of the rows will be positive. Once this is done, the previous proposition ensures that the lexicographic rule will prevent cycling in the simplex algorithm, causing it to successfully terminate.

Bland's/Smallest-subscript rule:
The problem with degeneracy was the we couldn't increase t beyond 0. As we mentioned, if an iterate x_i was degenerate, then we can still define x_{i+1} as we did, who's basis is the of x_i, except for a leaving variable k and a with the addition of an entering variable j. We will have $x_{i+1} = x_i$, but the corresponding bases will be different: we've just chosen a different subsets of hyperplanes intersecting to the same vertex. The question is if by proceeding like this, we will eventually find a leaving variable and an entering one such that the vertex changes.
There are only finitely many bases that can represent the same vertex. So if we keep pivoting without changing vertex, we will have cycled through the bases at some iteration. This means that all variables that have left the base at some iteration will have entered it at a later iteration. This suggests that if we specify a rule on how to choose entering and leaving variables, this cycling may lead to a contradiction in combination to the rule, guaranteeing that the rule will cause a different vertex to appear at some point. One idea is to order the variables and see what comes from choosing leaving and entering variables wrt that order.

Bland's/Smallest-subscript rule:

If we choose entering and leaving variables with the smallest index, then cycles can't occur, so that since there are only finitely many bases (over all vertices), the simplex method will terminate if we use this rule.

Proof: We consider the variables that have left and entered the cycle, and call them fickle. Of particular interest is the fickle variable of largest index, say t. At the base B_e where t is selected as entering variable to get the next base, since by the pivot rule we choose the entering variable of lowest index, no other fickle variable qualified as entering variable. If we denote by $c_e(j) = c^t y_e = c_j^t - c_{B_e}^t A_{*B_e}^{-1} A_{*j}$ the reduced cost, where we check for $c_e(j) < 0$ to determine if j is a candidate for an entering variable, then the situation described translates to $c_e(t) < 0$ and $c_e(j) \geqslant 0$ for all other fickle variables j.
There must also be a base B_l at which t leaves the base (in the next iteration). With our pivot rule, this means that for all fickle variables couldn't have been allowed to leave the bases. This translates to $x_{i,j} > 0$ for all fickle variables j such that $\left(A_{*B_e}^{-1} A_{*s}\right)_j > 0$, where i is the corresponding iteration and s is the variable entering, as the increment is $t = 0$. However, all fickle variables stay at 0 in all bases, since they are non-basic in one base of the cycle, where they take value 0, and all bases of the cycle represent the same point. So it must be the case that for all fickle variables j other then t, we have $\left(A_{*B_e}^{-1} A_{*s}\right)_j \leqslant 0$.

For a unknown motivation, we can consider the increment y direction at basis B_l. It satisfies $Ay = 0$ as an increment. By noticing that for $c_e^t = \sum c_e(j) e_j^t$ and $c_l^t = \sum c_l(j) e_j^t$, we have $c_e^t - c_l^t =$

$\left(\left(c^t_{B_l}A^{-1}_{*B_l} - c^t_{B_e}A^{-1}_{*B_e}\right)_j\right)A$, giving finally the property $(c_e - c_l)^t y = 0$. We will now decompose the dot-product and use the previous properties, together with a closer study of y, to get a contradiction on this dot product being 0.

Recall that $y_s = 1$, $y_{B_l} = -A^{-1}_{*B_l}A_{*s}$ and zeor on the other coordinates. Since t was chosen to leave, $y_t = -\left(A^{-1}_{*B_l}A_{*s}\right)_t < 0$ as to leave the index has to verify $\left(A^{-1}_{*B_l}A_{*s}\right)_t > 0$. For the fickle $j < t$ in B_l, we have $y_j = -\left(A^{-1}_{*B_l}A_{*s}\right)_j \geqslant 0$ by our previous paragraph.

Since t leaves B_l and was in B_e, we have $c_l(t) < 0$ and $c_e(t) = c_t - c^t_{B_e}A^{-1}_{*B_e}A_{*t} = c_t - c^t_{B_e}e_t = 0$, so that $c_e(t) - c_l(t) < 0$. Since $c_l(s) < 0$ as s enters B_l, and s is fickle, so that $c_e(s) \geqslant 0$, we have $c_e(s) - c_l(s) > 0$. For the fickle j that are in B_l, $c_e(j) \geqslant 0$ and $c_l(j) = 0$, so that $c_e(j) - c_l(j) \geqslant 0$. Finally, for the basic j that aren't fickle, hence are in all bases, we have $c_l(j) = c_e(j) = 0$ and in particular $c_e(j) - c_l(j) = 0$.

	$y_t < 0$	$c_e(t) - c_l(t) < 0$	
	$y_s > 0$	$c_e(s) - c_l(s) > 0$	
Summing up:	$y_j \geqslant 0$	$c_e(j) - c_l(j) \geqslant 0$	j *fickle*, *basic*, $j \neq t$
	$y_j = 0$	?	j *not basic*, $j \neq s$
	?	$c_e(j) - c_l(j) = 0$	j *basic*, *not fickle*

This implies that $(c_e - c_l)^t y > 0$, as can be seen by spliting up the sum along coordinates and using the signs we've established. This is a contradiction to $(c_e - c_l)^t y = 0$. So the initial assumption that cycles occur when we use our pivot rule is false. Hence, our pivot rule prohibits cycling.

10.4 Duality and the dual simplex algorithm

There is a general duality theory for optimization problems that is based on Lagrange multipliers, that we'll develop in a later chapter. The duality theory for LPs can be motivated directly.
For an LP of form $\max c^t x$ st. $Ax \leqslant b$, one may have the following idea: if it's possible to write c^t as combination of the rows of A, in the sense $c^t = \sum_{i\in[m]} s_i A_{i*}$, and in addition we have done this with positive scalars $s \geqslant 0$, then for any feasible point x, we have $c^t x = \sum_{i\in[m]} s_i A_{i*} x \leqslant s^t b$, so that $s^t b$ becomes an upper bound on the maximum value of the LP, as the latter inequality is true for any point achieving the maximum.
We can then look for the best possible upper bound by solving $\min b^t s$ st. $A^t s = c$ and $s \geqslant 0$. This seems a strange thing to do at first as we still have to solve an LP and we're not sure if the minimum of this new LP has any information on the first LP, besides providing an upper bound on its optimal value. Still, we'll investigate the idea further.

A first observation is that if one of he LPs is feasible, then the other either infeasible or bounded. Also, if one is unbounded, then the other is infeasible.
For the first LP, if it's feasible, then it's either unbounded or it's bounded with maximum z_M. The the latter case, $c^t x = z_M$ defines the optimal face of the polyhedron, and we can use the characterisation of faces to get the existence of a $s \geqslant 0$ such that $A^t s = c$ and $b^t s = z_M$. This s is a feasible found of the second LP, and it's actually optimal, as all values of the second LP upper bound z_M. So in this case, the second LP is feasible and the two have same optimal value.

We can try to show that if the second LP is feasible and bounded, then the first is two and the optimal values coincide. In that case, either both are feasible (hence the other is bounded) and their optimal value coincide, or both are infeasible or one is and the other is unbounded.
Instead of showing the claim directly, we can use an observation we'll develop now.

How can we get such a second LP for a standard form one $\min c^t x$ st. $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$?

If we transform $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$ to $\begin{pmatrix} A \\ -A \\ -I \end{pmatrix} x \leqslant \begin{pmatrix} b \\ -b \\ 0 \end{pmatrix}$ and the objective to $\max -c^t x$, and then consider the second LP, then it has form $\min \begin{pmatrix} b^t, -b^t, 0 \end{pmatrix} s$ st. $\begin{pmatrix} A^t, -A^t, -I \end{pmatrix} s = -c$ and $s \geqslant 0$. As any number can be represented as $y = y_+ - y_-$ for $y_+, y_- \geqslant 0$, the second LP is equivalent to $\min b^t y$ st. $A^t y \geqslant -c$ (forget slack), which is equivalent to $\min b^t z$ st. $A^t z \leqslant c$ (change $z = -y$). We note that this last form is the same as that of the very first LP. Hence, if we consider the "second" LP of the second LP $\min b^t s$ st. $\begin{cases} A^t s = c \\ s \geqslant 0 \end{cases}$, we note that we'll have gone full circle, as we recover that $\max c^t x$ st. $Ax \leqslant b$.
This is why the second LP associated the the first in the manners we've described is called the **dual LP** to the first one, which we call the **primal** LP.

We remark that the name dual is due to the idempotence of the operation, and has nothing to do with the duality of polyhedra, where the name was given for the same reason. For example, the dual to optimising a linear function on a cube is not an optimisation on the cross-polytope, which is the dual of the cube: the dimensions of the optimisation problems don't match, yet a polyhedron and its dual have same dimension and are in the same space.

The relation of primal and dual LP goes beyond having equal optimal values:

Complementary slackness:

For a primal LP $\max c^t x$ st. $Ax \leqslant b$ and its dual $\min b^t s$ st. $A^t s = c$ and $s \geqslant 0$, and two feasible solutions x and s to each respectively, then both are optimal precisely when for all $j \in [m]$, either $s_j = 0$ when $(Ax-b)_j = A_{j*}x - b_j < 0$, or $(Ax-b)_j = A_{j*}x - b_j = 0$ when $s_j > 0$, or both are zero.

Proof: We can investigate the duality gap $c^t x - s^t b$ by introducing a term that both relate to, $c^t x - s^t b = c^t x - s^t Ax + s^t Ax - s^t b = \left(c^t - s^t A\right) x + s^t (Ax - b)$, and we see that for feasible s and x, this gap is precisely 0 (which is precisely what happens when the solutions are optimal) when for all $j \in [m]$, either $s_j = 0$ when $(Ax-b)_j = A_{j*}x - b_j < 0$, or $(Ax-b)_j = A_{j*}x - b_j = 0$ when $s_j > 0$, or both are zero.

Complementary slackness provides a was of finding a dual optimal solution, if a non-degenerate primal one is known. If x^* is a non-degenerate solution to $\max c^t x$ st. $Ax \leqslant b$, in the sense that the tight inequalities of $Ax^* \leqslant b$ are correspond precisely to a set linearly independent rows of A indexed by I, that form a base of the space of x^*, then $x^* = A_{I*}^{-1} b_I$. Complementary slackness then tells us that for the minimum dual solution s^*, we have $s_j^* = 0$ for $j \notin I$. Then dual feasibility simplifies to $c = A^t s^* \Leftrightarrow c = (A_{I*})^t s_I^*$, which has the unique solution $s_I^* = \left(A_{I*}^{-1}\right)^t c$. So we can get the dual solution by simply solving a linear system. This isn't the case for a degenerate optimum.

Conversely, if we have a non-degenerate optimal solution to $\min b^t s$ for $c = A^t s$ and $s \geqslant 0$, provided by the simplex method, so that there is I for which $s_j^* = 0$ for $j \notin I$ and $s_I^* = \left(A_{I*}^{-1}\right)^t c$ and $s_j^* > 0$ for $j \in I$ (non-degeneracy). Then complementary slackness tells us that $x^* = A_{I*}^{-1} b_I$ is precisely the optimal soution to the dual.

We can note the following. Given an index set I for which the rows of A index by I for a base of $\mathbb{R}^n$, then if $x = A_{I*}^{-1} b_I$ satisfies $A_{[m]\setminus I*} x \leqslant b_{[m]\setminus I}$, then this point is a feasible solution to the LP on $Ax \leqslant b$, and if $s_I = \left(A_{I*}^{-1}\right)^t c$ satisfies $s_I \geqslant 0$, then we can complete it to a feasible solution of the LP on $A^t s = c$, $s \geqslant 0$, by setting $s_j = 0$ for $j \notin I$. If we can do both, then $c^t x = s_I^t A_{I*} A_{I*}^{-1} b_I = s^t b$, so that weak duality ensures that they are actually optimal for their respective LP.

COMPLETE WITH THE TABLEAU STUFF

10.5 Sensitivity analysis and post-optimization

We start withe the sensitivity analysis.
TO COMPLETE: "ContinuityLinProg" in folder, "A charcterisation of stability for linear programming" by Robinson. And chapter by Bertsimas, parts on global dependece.

We now turn to post-optimization.
Variation of the right term:
We ask how to "fix" our solution if $\max c^t x$ st. $Ax \leqslant b$ is pertubed to $\max c^t x$ st. $Ax \leqslant b'$ or if $\min c^t x$ st. $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$ is perturbed to $\min c^t x$ st. $\begin{cases} Ax = b' \\ x \geqslant 0 \end{cases}$. We assume that the solutions of the unperturbed problems are non-degenerate.
In the first case, recall that the associated dual solution s^* to an optimal x^*, which can be found by solving a linear system, remains a basic feasible solution to the dual LP (with basis the indices of tght inequalities at the primal optimum), which has constraints $\begin{cases} A^t s = c \\ s \geqslant 0 \end{cases}$, but now has objective value $b'^t s$, which may not be optimal anymore. We can the perform the simplex method on the dual with s^* as initial point, sparing us the phase 1 of the simplex method of the perturbed primal.
In the second case, we do the same, with duality as well.

Variation of the cost:
If the costs of $\min c^t x$ st. $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$ is perturbed to $\min c'^t x$ st. $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$, then an optimal solution to the first problem remains feasible, but may not be optimal anymore. We can check this by considering the new reduced costs, $c'_j - c'_{B^*} A^{-1}_{*B^*} A_{*j}$, where B^* was a basis of an optimal solution to the unperturbed problem. We can then persue the simplex algorithm, with these new costs.

Adding a new variable:
ssume a new variable has to be considered, so that $\min c^t x$ st. $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$ becomes $\min c^t x + c_N x_N$ st. $\begin{cases} Ax + x_N a = b \\ x, x_N \geqslant 0 \end{cases}$. We have feasible solution $(x, x_N) = (x^*, 0)$, where x^* was an optimal solution to the previous problem, with same basis B^*. It may however not be optimal anymore: the reduced cost at N is now $c_N - c_{B^*} A^{-1}_{*B^*} a$. Note that $c_{B^*} A^{-1}_{*B^*}$ corresponds to the optimal dual solution, in the non-degenerate case, so that the reduced cost can be computed easily. We may have to keep runing the simplex algorithm if this cost is negative.

TO FINISH

10.6 Klee-Minty cube

"How good is the simplex algorithm" klee minty

10.7 Large-scale and decomposition methods

Delayed column generation method:
We consider the cutting stock problem, in which we're given steal bars of length L, and a set of m lengths l_i at which steal bars are usually sold. To cut the large rods into smaller ones, we use paterns $a_1, ..., a_m$ where a_i is the number of rods of length l_i we cut from one big rod. So we require that $\sum_{i\in[m]} a_i l_i \leqslant L$. We assume that we have demands $b_i \geqslant 0$ for bars of legth l_i that we have to meet by cutting the rod of length L. The goal is to minimise the number of big rods we need, to meet the demand. To model this, we can introduce variables $x_{(a_1,...,a_m)}$ which are the numbers of rods of length L to be cut according to valid pattern $a_1, ..., a_m$. We then try to solve the LP $\min 1^t x$ st. $\begin{cases}\sum_{(a_1,...,a_m)} a_i x_{(a_1,...,a_m)} = b_i \\ x \geqslant 0\end{cases}$ (this should be an IP, but rounding the LP relaxation upwards is good enough for industrial purposes). The problem here is that the variables and sums range over a set that must first be computed (we sum over the integer $a_1, ..., a_m$ for which $\sum_{i\in[m]} a_i l_i \leqslant L$) and that may have exponential size. We'll now describe a method for handling this.

We can find an initial basic feasible solution by setting $x_{e_j^t} = b_i$ for $j \in [m]$ and $x_{(a_1,...,a_m)} = 0$ otherwise, with basis $\left(e_j^t\right)_{j\in[m]}$ (remember the indices are vectors of integers). In the simplex algorithm, at a given basis B, we decide which way to go by looking at the residual costs, which have form $1^t_{(a_1,...,a_m)} - 1_B A_{*B}^{-1} A_{*(a_1,...,a_m)}$. If it's positive for all patterns, then the current solution is optimal, otherwise we can find a pattern for which this isn't the case and have it enter the basis. To determine this, we can solve $\min\left(1^t_{(a_1,...,a_m)} - 1^t_B A^{-1}_{*B} A_{*(a_1,...,a_m)}\right)$ over integer $a_1, ..., a_m$ for which $\sum_{i\in[m]} a_i l_i \leqslant L$: if the minimum is positive, the basis is optimal, otherwise the pattern minimising it is a candidate for entering the basis as an index. The wonderful twist in difficulty is that the latter problem is equivalent to maximising $1^t_B A^{-1}_{*B} A_{*(a_1,...,a_m)} = 1^t_B A^{-1}_{*B}(a_1, ..., a_m)^t$ over these constraints: this is the knapsack problem, and it can be solved more efficiently with dynamic programming methods, or even rapidly approximately with approximation algorithms. This means that we can move from basis to basis and check optimality fairly rapidly. This particular adaptation of the revisited simplex method allows us to solve a large problem without a large number of computations at each iteration.

Generally, we refer to a delayed column generation method as describing a revisited simplex method in which the check for optimality and can be done more efficiently then by checking all the reduced costs individually, usually with a subroutine such as knapsack in our case.

Delayed constraint generation or cutting plane method:
This in a sense the dual version of column generation. It applies to problems of form $\max c^t x$ st. $Ax \leqslant b$, where the number of contraints is exponentially large, but where we know that the problem is bounded. It works by starting on a bounded problem $\max c^t x$ st. $A_{I*} x \leqslant b_I$ for a small index set I. At each step, we solve this problem, finding an optimal vertex x^* (due to boundedness). The question is if this x^* is feasible for all constraints, which requires us to check $Ax^* \leqslant b$, which we can do by solving $\min_i (b_i - A_{i*}x^*)$ and checking if its positive. If it is, then x^* is feasible and since any feasible x for the main problem on $Ax \leqslant b$ is feasible on $A_{I*}x \leqslant b_I$, so in particular the optimum of the main problem x^{**}, then $c^t x^{**} \leqslant c^t x^*$ from the small problem, so that x^* is feasible optimal for the main problem. Oherwise, we get an index $i \notin I$ with $b_i - A_{i*}x^* < 0$. We then reiterate our procedure with the new index set $I \cup i$: however, we can recycle x^* by noticing that all we have to do is a post-optimization step, as only one new constraint $A_{i*}x \leqslant b_i$ has been added, to get the next optimum x^*, which is attained as since $\max c^t x$ st. $A_{I*}x \leqslant b_I$ is bounded, so must the problem for $I := I \cup i$ be. Note that we may discover infeasibility along the way. Note also that $b_i = A_{i*}x$ separates x^* from the solution space, hence the name cutting plane method.

The point is that, just like for reduced costs in column generation, the routine problem $\min_i (b_i - A_{i*}x^*)$ may be solved more efficiently then by computing all differences, depending on the structure of the problem.

Dantzig-Wolfe decomposition:
Dantzig-Wolfe decomposition uses the representation of polyhedra in a particular context.
Suppose we want to solve a problem of form $\min c^t \begin{pmatrix} x \\ y \end{pmatrix}$ st. $\begin{cases} Ax + By = a \\ Cx = b \\ Dy = d \\ x, y \geqslant 0 \end{cases}$. By letting P be the polyhedron defined by $\begin{cases} Cx = b \\ x \geqslant 0 \end{cases}$ and Q the one defined by $\begin{cases} Dy = d \\ y \geqslant 0 \end{cases}$, we can reformulate the solution space as $\begin{cases} Ax + By = a \\ x \in P \\ y \in Q \end{cases}$.

By writing the polyhedra in V-representation, we get $\begin{cases} AV_1\lambda_1 + AR_1\mu_1 + BV_2\lambda_2 + BR_2\mu_2 = a \\ \lambda_1, \mu_1 \geqslant 0, 1^t\lambda_1 = 1 \\ \lambda_2, \mu_2 \geqslant 0, 1^t\lambda_2 = 1 \end{cases}$. We'll never have to actually compute this representation, but we need to consider it for theoretical purposes.
If we want to solve the problem in V-representation, we can use the revisited simplex method. The problem here is that the residual costs to compute may be exponential in number (as many as vertices and extreme rays). For example, for a basis B, the residual cost of variable $\lambda_{1,j}$ is $c^tV_{1,j} - c_B^tM_B^{-1}\begin{pmatrix} AV_{1,j} \\ 1 \\ 0 \end{pmatrix}$, where M denotes the matrix representation of the LP in V-representation, and the last two lines of $\begin{pmatrix} AV_{1,j} \\ 1 \\ 0 \end{pmatrix}$ correspond to the contraints $1^t\lambda_1 = 1$ and $1^t\lambda_2 = 1$ respectively. Note that $c^tV_{1,j} - c_B^tM_B^{-1}\begin{pmatrix} AV_{1,j} \\ 1 \\ 0 \end{pmatrix} = \left(c^t - c_B^tM_B^{-1}\right)AV_{1,j} + r$ for some rest expression r depending only on B. For rays, the residual cost of variable $\mu_{1,j}$ is $c^tR_{1,j} - c_B^tM_B^{-1}\begin{pmatrix} AR_{1,j} \\ 0 \\ 0 \end{pmatrix} = \left(c^t - c_B^tM_B^{-1}\right)AR_{1,j}$.

We can now see how to use column generation techniques in this context. By solving $\min \left(\left(c^t - c_B^tM_B^{-1}\right)A\right)x$ of $x \in P$ with the simplex method, we either get a vertex $V_{1,j}$ or a ray $R_{1,j}$ as solution, with either objective value $\geqslant -r$, or $< -r$, or the knowledge that the problem is unbounded unbounded. In the first case, we know that none of the residual costs of the x-part of the main problem in V-representation can be negative, so the current solution is optimal if this is the case for Q as well. In the second case, the solution to this auxiliary LP provides a $V_{1,j}$ who's corresponding variable may enter the basis, and in the third it provides a $R_{1,j}$ who's corresponding variable may enter the basis.
Hence, this describes a column generation technique that uses LPs as subroutines to determine if reduced costs imply optimality.

The last problem is the following: with the revisited simplex method and the fect that we compute $V_{1,j}$ or $R_{1,j}$ at each iteration implies that we never have to compute the entire V-representation of P or Q. However, we need an initial basic feasible solution to the main problem in V-representation.

To do this efficiently, we start by finding vertices $v_P = V_{1,i}$ and $v_Q = V_{2,j}$ of P and Q (which are standard hence pointed) with phase 1 of the simplex method (or determine infeasibility). Next, we solve a sort of phase 1 for the decomposition method. We may assume that $Av_p + Bv_Q \leqslant a$, (or multiply this constraint by -1 to get the same initial LP and this inequality).

We solve $\min 1^t z$ st. $\begin{cases} AV_1\lambda_1 + AR_1\mu_1 + BV_2\lambda_2 + BR_2\mu_2 + z = a \\ \lambda_1, \mu_1 \geqslant 0, 1^t\lambda_1 = 1 \\ \lambda_2, \mu_2 \geqslant 0, 1^t\lambda_2 = 1 \\ z \geqslant 0 \end{cases}$, which has objective value 0 precisely if the main LP is feasible, and who's optimal basic solution will be a basic feasible solution to the main LP in V-representation, in this case. This problem seems to require knowing the V-representation of the polyhedra. The twist is that we can use the previously described method to solve this problem, as we can get a basic feasible solution for it formulaically. Indeed, we have a basis in the indices $(1, i)$, $(2, j)$ and the coordinates corresponding to z, with basic feasible solution $\lambda_{1,i} = 1$, $\lambda_{2,j} = 1$, $z = a - AV_{1,i} - BV_{2,j} \geqslant 0$ and all other coordinates 0.

Benders decomposition:

Just like for contraint generation is in a sense the dual to column generation, Benders decomposition is the dual to Dantzig-Wolfe decomposition. It applies to problems such as the following one, which arrise for example in stochastic programming: $\min \left(c^t x + \sum_{i \in [k]} f_i^t y_i \right)$ st. $\begin{cases} Ax = b \\ B_i x + C y_i = d_i, \forall i \in [k] \\ x, y_1, ..., y_k \geqslant 0 \end{cases}$.

The idea is that for fixed feasible x, we have to solve $\min f_i^t y_i$ st. $\begin{cases} Cy_i = (d_i - B_i x) \\ y_i \geqslant 0 \end{cases}$ for all i to get the best possible objective value for the main problem, which is then decoupled. Equivalently, we can solve the dual $\max \left((d_i - B_i x)^t z \right)$ st. $C^t z \leqslant f_i$.

10.8 Simplex specification: the network simplex method

Some optimization problems that have a combinatorial flavour can be phrased as LPs and solved by the simplex method. In such cases, we can interpret the notions from the simplex method in a combinatorial context. This can lead us to replace simplex steps by equivalent ones that use the combinatorial context, and that are faster or use less memory then the brute simplex. We can also hope to get a bound on the runtime of the simplex method from the combinatorial context. We give the example of the network simplex method.

The problem:
We consider a network flow problem in which we're given a digraph G, edge costs c and net production b of a resource at each vertex, so $b_v > 0$ if vertex v produces more then it consumes and $b_v < 0$ otherwise. The goal is to ship the resource in quantity $f_e \geqslant 0$ along edge e so that the net demands are met and no resource is stored: so for all v, we have $b_v + \sum_{e \in \delta^-(v)} f_e = \sum_{e \in \delta^+(v)} f_e$. We assume that the graph is undericted when orientation and parallel edges are ignored, for in the oppsite case, we solve the problem on the connected components. A necessary condition for such a flow f to exits is that "rien ne se perd, rien ne se crée, tout ce transforme", $\sum_{v \in V} b_v = 0$, as can be seen by summing the contraints $b_v + \sum_{e \in \delta^-(v)} f_e = \sum_{e \in \delta^+(v)} f_e$ over se vertices and noting that each edge has a head and a tail. Note however that it may not be sufficient, for example in a one-edge-digraph of edge (a, b) where $b_a = -1$ and $b_b = 1$. We want to do this at a minimum cost, given by $c^t f$, where c_e is the cost per unit resource transferred along e.
If A denotes the vertex-arc incidence matrix, so that $a_{ve} = \begin{cases} 1 : e \in \delta^+(v) \\ -1 : e \in \delta^-(v) \\ 0 : else \end{cases}$, then we can rephrase the problem as the LP $\min c^t f$ st. $Af = b$, with $f \geqslant 0$.

The network simplex method:
We almost have a standard formulation of the problem. However A doesn't have full rank: when we sum its rows, we get the zero vector as each row contains exactly two non-zero entries 1 and -1. We can therefore ignore the last row, corresponding to the last vertex r in some ordering, as we can dedue it from the others. Then, A (useing the same notation) will have full rank. Indeed, assume otherwise for contradiction. Then there is $\lambda \neq 0$ with $\lambda^t A = 0$. By considering a $\lambda_v \neq 0$ and an edge e incident to v (recall that G is connected), the condition on column e of $\lambda^t A = 0$ implies that the other endpoint of e must have same coefficient in λ. By conectedness, the is a path (possibly along reverse oriented edges) from v to r. By applying the argument, we see that the λ coefficients along the vertices of the path must be the same. At the last edge of the path (w, r), the colum of A has a single non-zero entry, since we deleted the last row corresponding to r. Therefore, at that column, $\lambda^t A = 0$ implies that $\lambda_v = \lambda_w = 0$, contradicting $\lambda_v \neq 0$.

We'll now investigate what a basis from the simplex method corresponds to in this context. We seek $|V| - 1$ columns, corresponding to edges, that are linearly independent. It's actually easier to find edges that correspond to dependent columns. For a cycle C, in the undirected sense, we chose an orientation on it and add the columns corresponding to edges that conform to the orientation and subtract those that don't. Observing the coordinates of that sum, we see that for each vertex only two non-zero terms appear, which are 1 and -1 for all orientations and summation signs. Thus, we've produced a non-zero combination of columns that is zero. Therefore, we should search among edge sets that don't contain cycles, aka. forests, as otherwise we could use the combination on the cycle to get a zero combination. Now the only forests that have $|V| - 1$ edges are trees, so that only trees could correspond to basic solutions of the simplex method. In fact, any of them does: if there were a linear combination of columns corresponding to a tree

that is zero, we could observe the coordinate of a leaf of a tree that isn't r (there are at least two leaves in any tree). Since it has a single non-zero entry in the combination, the corresponding coefficient must be zero. If we repeat this argument further and further on the edges with supposed non-zero coefficients, which are trees since deleting leaves from trees maintains the tree property, we see that the combination is the zeor combination, so that the edge corresponding to the tree are linearly independent.

Hence, a basic feasible solution of the LP corresponds to a tree T, where $f_e = 0$ for $e \notin T$ and $f_e \geqslant 0$ for $e \in T$. Note that for a given tree, it is easy to compute the corresponding flow and therefore check feasibility of the basic solution. Indeed, we start at a leaf l connected to the tree by (l, w), where by $f_e = 0$ for $e \notin T$ and $b_v + \sum_{e \in \delta^-(v)} f_e = \sum_{e \in \delta^+(v)} f_e$ it must hold $f_{wl} = b_l$. If we then ignore this leaf, deleting it mentally, we're faced with the same problem on a smaller tree with the same b, except for the new $b_w := b_w \pm b_l$, to account for $f_{wl} = b_l$ in the balance constraint $b_v + \sum_{e \in \delta^-(v)} f_e = \sum_{e \in \delta^+(v)} f_e$. We can therefore proceed leaf by leaf in this way to recover the flows. We check feasibility along the way with $f_e \geqslant 0$.

To start the simplex algorithm, we need an initial feasible tree. We can do this with a phase 1 that has a particular form in this combinatorial context. We use the network simplex algorithm on an auxiliary graph that is constructed as follows: we add a vertex s, that is connected to each vertex v with $b_v \geqslant 0$ by (v, s) and to each vertex v with $b_v < 0$ by (s, v). Then the edges incident to s form a tree, and this tree is feasible since the flow $f_e = |b_v|$ is feasible for it, where $b_s = 0$. On this auxiliary network, we minimise $c^t f$ where $c_e = \chi_{\delta(s)}$, so that basic feasible solution of objective value 0 corresponds to a feasible tree that uses no edges incident to the artificial s, hence is also a feasible tree in the original network.

Now, how do we improve the objective ? What do pivots and reduced costs look like ?
If we let variable e enter the base, then $e \notin T$ closes a unique cycle C in the tree. Increasing $f_e = 0$ by t while keeping the other non-tree-edge-flows zero requires adjusting the flows on the tree. It turns out that only changing flows on the cycle is necessary and sufficient. If we changed flow on a non-cycle variable, then we can consider a leaf among the edges who's flow gets adjusted in the connected component of that edge, when the cycle is deleted. The edge of this leaf can't have its flow adjusted. So only adjustments on the cycle C are possible.
We orient C in the direction of e. Increasing $f_e = 0$ by $t \geqslant 0$ while changing flows only on the cycle is done as follows: for edges $a \in C$ conforming to the orientation, we add t of flow, and for $a \in C$ not conforming to the orientation, we decrease t of flow. To maintain feasibility, flows have to be positive, so that the most we can increase is $t = \min_{a \in B}(f_a)$ where B is the set of edges of C not conforming to the orientation. The edge that achieves this minimum will be the leaving variable. If all edges on C conform to its orientation along e, so that C is a dicycle, then we can increase t indefinitely.

The reduced costs for basis B have form $\overline{c}^t = c^t - p^t A$ for the dual variable $p^t = c_B^t A_{*B}^{-1}$. Computing $p^t A$, we see that $\overline{c}_{(u,v)} = \begin{cases} c_{(u,v)} - (p_u - p_v) : u, v \neq r \\ c_{(u,v)} - p_u : v = r \\ c_{(u,v)} + p_v : u = r \end{cases}$, or $\overline{c}_{(u,v)} = c_{(u,v)} - (p_u - p_u)$ with the convention $p_r = 0$. Now, since the reduced cost are zeor on the basis, we get $c_{(u,v)} = p_u - p_v$ for $(u, v) \in T$ where T is the tree corresponding to basis B.
This actually allows us to compute p, and therefore also the reduced cost of edges not in the basis/tree. Indeed, we have $p_r = 0$ and for any undirected path from r to any other vertex v, we can use the relations $c_{(u,v)} + p_v = p_u$ or $p_v = p_u - c_{(u,v)}$ to compute the p-value at the next vertex on the path staring from r, until we reach v and get p_v. We can therefore compute these "potentials" p along the tree in a BFS type manner, knowing only the current basis/tree T, and compute the reduced costs of other edges via $\overline{c}^t = c^t - p^t A$.

So a step in the network simplex algorithm starts by finding potentials p associated to the tree solution T with a BFS-type algorithm, from which we get reduced costs of edges not in the tree, which tells us which edge to augment along, which we do by finding an edge on the closed cycle that has reverse orientation and who's flow is smallest, decreasing the flow on that edge and increasing it on the entering edge. On last trick for a clever implementation is that we don't have to compute potentials from scratch all the time. If T is the tree at a step, and edge e leaves and edge a enters, then the potentials of the connected component of $T\backslash e$ that contains the root r doesn't change, as we'd recompute them in the same way we did before. We therefore only need to compute the potentials of the other connected component of $T\backslash e$, which we can do also a tiny bit more efficiently: a was endpoints in both components, so that we know the potential of on endpoint, from which we deduce that of the other. We the treat the second endpoint as a root of the component of $T\backslash e$ that doesn't contain r, which is a tree, and use the same procedure to compute potentials on it.

We note that in the case that the cycle we pivot on is a directed one, then for the entering edge (u, v) we have $c_{(u,v)} - (p_u - p_v) < 0$, and by summing the costs on the rest on the cycle, which are in the tree so that $c_{(u,v)} = p_u - p_v$, we see that the potentials vanish, as they appear each twise with opposite sign, so that $\sum_{e \in C} c_e < 0$. We have thus detected a negative cost dicycle.

We close this section by discussing the running time of the network simplex method. By using anti-cycling rules, we can ensure that we never see the same trees twice as bases. We know that the number of trees is less then that in a complete graph on $|V|$ vertices, so that we get a bound on the number of iterations of $|V|^{|V|-2}$. We can get other bounds by closer inspection of the number of spanning trees: with the matrix-tree theorem and the Hadamar inequality, we can bound the number of iterations by $\left(\sqrt{4(|V|-2)+\Delta^2}\right)^{|V|}$, where Δ is the maximum degree in the graph when undirected. We mention that exponential runtime examples can indeed be constructed, like the Minty-Klee cube.

10.9 Solutions

11 Integer linear Programming

11.1 Introduction and examples

Definition:
An **integer programm** (IP) is an optimisation problem of the form

$$\min(c\cdot x) \text{ st. } \begin{cases} Ax \leqslant b \\ x \in \mathbb{Z}^n \end{cases}$$

A **mixed integer programm** (MIP) is an optimisation problem of the form

$$\min\left(c\cdot \begin{pmatrix} x \\ y \end{pmatrix}\right) \text{ st. } \begin{cases} A\binom{x}{y} \leqslant b \\ x \in \mathbb{Z}^n, y \in \mathbb{R}^k \end{cases}$$

We give examples of IPs with a combinatorial flavour.

The knapsack problem:
A robber breaks into your house with a bag (knapsack) of volume capacity W. The robber knows the volume w_i and the value v_i of your items $i \in [m]$. The robber wants to maximise the total value of objects to steal, subject to the constraint that the total volume of the objects to steal doesn't exceed the volume of the bag he'll carry them with.
We can model this as an IP by defining indicator variables x_i for the items so that $x_i = 1$ represents the robber stealing item i and $x_i = 0$ represents him leaving it. Then, the objective function is $\sum_{i\in[m]} v_i x_i$, the volume constraints are $\sum_{i\in[m]} w_i x_i \leq W$, and the final constraints are $0 \leq x_i \leq 1$ and $x \in \mathbb{Z}^n$.

Ex.Sudoku: Show how to solve a Sudoku with an integer program.

The bin packing problem:
We're moving to Berlin and are faced with the following problem. We have to pack our things into bags of unit volume and try to use as little bags that we'll have to carry up and down stairs.
If the item $i[n]$ has volume $a_i \leqslant 1$, we know that we'll have to use at most n bins b_j. We can use b_j as an indicator variable that tells us if we use bin j. We also use indicator variables x_{ij} to represent putting item i in bin j. Then our goal is to minimise $\sum_{j\in[n]} b_j$ subject to $\sum_{i\in[n]} a_i x_{ij} \leqslant b_j$ for all bins b_j and $\sum_{j\in[n]} x_{ij} = 1$, with $x_{ij}, b_j \in \{0,1\}$.

The (uncapacitated) facility location problem:
We're given a set of n sites at which we may open a store, the costs f_j of opening the store at location j, as well as data p_{ij} of the profit brought by one of m consumers i if the store is opened at j (if the store is closer to the consumer, the consumer might visit more often). If we use indicator/characteristic vectors, we encode by $x_j = 1$ that we open a store at location j, and by $y_{ij} = 1$ that consumer i shops at the store located in j. Consumers only go to one store of the chain, so $\sum_{j\in[n]} y_{ij} = 1$. We encode that fact that a store has to be open for consumers to shop in it by $\sum_{i\in[m]} y_{ij} \leqslant x_j$. Finally, subject to these constraints (and $x, y \in \{0,1\}^n, \{0,1\}^m$), we which to maximise profits $\sum_{ij} p_{ij} y_{ij} - \sum_j f_j x_j$.

Why study combinatorial problems as IPs ?

To answer this question, we'll give an illustrative example. We'll formulate the MST problem as a combinatorial one. Remember that trees are the graphs that have $|V|-1$ edges and no cycles. If we let x_e indicate the use of edge e, then the contraint $\sum_{e\in E} x_e = |V|-1$ models the first condition, and since $\sum_{e\in E(U)} x_e \leqslant |U|-1$ prohibits the existence of a cycle on vertex set $U \subseteq V$, the system $\begin{cases} \sum_{e\in E} x_e = |V|-1 \\ \sum_{e\in E(U)} x_e \leqslant |U|-1, \forall U \subseteq V, |U|>2 \\ x \in \{0,1\}^E \end{cases}$ indicates a tree of a graph. Finiding a MST then becomes solving the IP $\min \sum_{e\in E} c_e x_e$ on this solution set.

This may seem unnecessarily complicated when compared to the simple greedy algorithm solving the MST problem. However, it has the advantage that we didn't need to investigate the problem from a combinarorial persepective.

If we consider the problem of finding a spanning tree who's number of leaves is maximum in a connected graph and we attempt to find a combinatorial and efficient approach, we run into trouble. Indeed, it turns out that this problem is NP-complete, so it's likely that this problem has ne efficient algorithm to solve it. Additionally, it seems hard to extract anything from the MST problem that could help for this problem, when considering the combinatorial aspect.

On the oher hand, the IP version can be modified quite easily to solve the problem. Indeed, if l_v is to indicate wether vertex v is a leaf, then $l_v \leqslant \frac{1}{|\delta(v)|-1}\left(|\delta(v)| - \sum_{e\in\delta(v)} x_e\right)$ prohibits it from taking value 1 unless v is a leaf. If we solve $\max \sum_{v\in V} l_v$ for $\begin{cases} \sum_{e\in E} x_e = |V|-1 \\ \sum_{e\in E(U)} x_e \leqslant |U|-1, \forall U \subseteq V, |U|>2 \\ l_v \leqslant \frac{1}{|\delta(v)|-1}\left(|\delta(v)| - \sum_{e\in\delta(v)} x_e\right), \forall v \in V,\ |\delta(v)| \geqslant 2 \\ x \in \{0,1\}^E, l \in \{0,1\}^V \end{cases}$ then for the optimal solution, l_v will be precisely 1 if v is a leaf (otherwise we could get a larger objective by increasing it), so that the optimal value is the maximum number of leaves of a spanning tree.

The moral of the story is that finding IP formulations is easier then finding combinatorial algorithms. Of course, combinatorial algorithms for a specific problem can be (and often are) much more efficient then the general methods used to solve IPs. But, as the saying goes: "first do, then do efficiently".

11.2 LP relaxation and total unimodularity

We could try to adapt the simplex algorithm to solve (M)IP problems, but such an adaptation fails. We'll see that solving IPs can be achieved by solving LPs on specific polyhedra. The problem is that determining the H-description of such polyhedra can't be done efficiently, so the best we can get from the previous chapters will be a relaxation of the IP.

Definition:
The **integer hull** of a polyhedron P is the convex hull of its integer points $P_I = conv\,(P \cap \mathbb{Z}^n)$.
The **LP relaxation** of of an IP is the LP:

$$\min(c\cdot x) \text{ st. } \begin{cases} Ax \leqslant b \\ x \in \mathbb{R}^n \end{cases}$$

Ex.CHH: Show that the integer hull of the half space $\{x : a\cdot x \leqslant b\}$ for $a \in \mathbb{Z}^n$ is $\left\{x : \left(\frac{1}{g}a\right)\cdot x \leqslant \left\lfloor \frac{b}{g} \right\rfloor\right\}$, where $g = gcd(a_i)$. What's the relation between the integer hull P_I of a polyhedron in H-description $P = \cap_i H_i^+$ and the polyhedron $\cap_i (H_i^+)_I$, which we now know how to describe for rational entries ?

Proposition: *Equivalence of an IP with the integer hull LP*
Solving an IP with polyhedron P can be done by solving the LP on P_I.

Proof: The constraints $Ax \leqslant b$ and $x \in \mathbb{Z}^n$ are infeasable (P has no integer points) precisely when P_I is empty, so both programms are infeasible simultaneously. Note that P_I contains all points verfifying $\begin{cases} Ax \leqslant b \\ x \in \mathbb{Z}^n \end{cases}$, so that $\min\limits_{P_I}(c\cdot x) \leqslant \min\limits_{P, x\in\mathbb{Z}^n}(c\cdot x)$. In the unbounded cases, equality holds, so we're left with the case in which $\min\limits_{P_I}(c\cdot x)$ is finite and attained in x_o and our goal is to prove that $\min\limits_{P_I}(c\cdot x)$ is attained in an integer point x_{io} as well, so that $\min\limits_{P_I}(c\cdot x) = c\cdot x_{io} \geqslant \min\limits_{P, x\in\mathbb{Z}^n}(c\cdot x)$ and x_{io} solves the IP.
By the definition of P_I, we have the convex combination $x_o = \sum s_i x_i$ with $x_i \in \mathbb{Z}^n$. If one of these x_i had suboptimal value, $c\cdot x_j < \min\limits_{P_I}(c\cdot x)$, then scaling and summing with $c\cdot x_i \leqslant \min\limits_{P_I}(c\cdot x)$ would yield the contradiction $c\cdot x_o < \min\limits_{P_I}(c\cdot x)$. So there is in particular at last one $x_j := x_{io}$ such that $c\cdot x_j = \min\limits_{P_I}(c\cdot x)$, which we were looking for.

Proposition: *LP relaxation*
For an IP with polyhedron P, the LP relaxation gives a lower bound on the optimal value:

$$\min_{P}(c\cdot x) \leqslant \min_{P_I}(c\cdot x)$$

Proof: This is because $P_I \subset P$, which is due to P being convex and P_I being the convex hull of points in P. So all values of $c\cdot x$ on P_I are greater then $\min_P(c\cdot x)$.

Remark: The LP relaxation can yield terrible bounds. For example consider the polygon between the

lines $y = \frac{x}{k}$ and $y = 1 - \frac{x}{k}$ given by $\begin{cases} x \geqslant 0 \\ ky \geqslant x \\ ky \leqslant k - x \end{cases}$ for a parameter $k \in \mathbb{N}$. It's only integer points are $(0,0)$ and $(0,1)$, as for integer points with $x \geqslant 1$, $k - x \geqslant ky \geqslant x$ forces $y \geqslant 1$ and contradicts $x \geqslant 0$. For objective function $c = (-1, 0)$, the LP relaxation yields minimum $c \cdot \left(\frac{k}{2}, \frac{1}{2}\right) = -\frac{k}{2}$ while the only feasible values for the two integer points are 0. So the distance of the relaxation bound to the IP value can be arbitrarily large.

On the other hand, in some cases, the LP relaxation solves the IP.
We'll soon give precise and more computational criteria for P so that this happens.

Definition:
A polyhedron P is an **integer polyhedron** if $P = P_I$, aka. P is the hull of its integer points.

Proposition: *LP relaxation for integer polyhedra*
If P is an integer polyhedron, then the LP relaxation solves the LP.

Proof: this is due to to the *equivalence of an IP with the integer hull LP* for the case that $P = P_I$.

Ex.RelaxEx: Show that the converse doesn't hold by giving an example of an IP on a non-integer polyhedron, who's LP relaxation solves the IP.

Remark: An interesting fact is that the solution space of an IP can be represented in different ways. By that, we mean that $\left\{ x : \begin{cases} Ax \leqslant b \\ x \in \mathbb{Z}^n \end{cases} \right\} = \left\{ x : \begin{cases} A'x \leqslant b' \\ x \in \mathbb{Z}^n \end{cases} \right\}$ for different matrices A and vectors b. For example, the integer points of the cube $[0,1]^2$ are also those of $[0, 1.2] \times [-0.5, 1.3]$.
This observation raises the question of the difference between the relaxations for these different formulations. For the example of the cube, the first formulation is an integer polyhedron, so that the LP solves the IP, whereas the second doesn't. The next paragraph is a nontrivial example of this phenomenon.

Recall the knapsack problem from the previous section. From its definition, we can consider the relaxation with polyhedron $P = \left\{ x \in [0,1]^3 : \sum w_i x_i \leqslant W \right\}$.
We now derive a different formulation of the knapsack problem.
We call a set of items C a *minimal cover* for the problem if breakes the weight constraint, $\sum_{i \in C} w_i > W$, but removing any item restores it, $\forall j \in C$, $\sum_{i \in C \setminus j} w_i \leqslant W$. They are the inclusion minimal infeasible sets of items. Any feasible set of items for the knapsack instance (caracterised by x) can't contain all the items of a minimal cover (for $w_i \geq 0$), so $\sum_{i \in C} x_i \leqslant |C| - 1$, for otherwise, the total volume of that item set is at least $\sum_{i \in C} w_i > W$. The intersting thing is that the converse holds, if a set of items (caracterised by x) verifies $\sum_{i \in C} x_i \leqslant |C| - 1$ for all minimal covers C, then it is feasible for the knapsack problem.

Ex.Knap1: Prove the last claim.

From this new formulation, we get a new underlying polytope: $Q = \left\{ x \in [0,1]^3 : \sum_{i \in C} x_i \leqslant |C| - 1, \forall C \in K \right\}$ where K is the set of minimal covers of the knapsack instance. As we just saw above, P and Q have the same integer points, so $P_I = Q_I$.
The interesting fact is that depending of the knapsack instance, it's possible that $P \subsetneq Q$ or $Q \subsetneq P$, so that $\min_P(c \cdot x) \geqslant \min_Q(c \cdot x)$ or $\min_P(c \cdot x) \leqslant \min_Q(c \cdot x)$, so that one LP relaxation may yield a beter bound then the other.

For example, for 3 items of volume 2 and a bag of volume 3, $P = \{x, y, z \in [0,1] : 2x + 2y + 2z \leqslant 3\}$. The minimal covers are all subsets of items of size 2, since all items are feasible and deleting any item from the set of all 3 of them yields 2 items, which have volume 4>3, so $Q = \{x, y, z \in [0,1] : x + y \leqslant 1, x + z \leqslant 1, y + z \leqslant 1\}$. If we add up all inequalities from Q, we get the one defining P, so that $Q \subset P$. But $\left(1, \frac{1}{2}, 0\right) \in P \backslash Q$, so the inclusion is strcit, and Q yields a better relaxation.
Now, for 3 items of volume 1 and a bag of volume 1, $P = \{x, y, z \in [0,1] : x + y + z \leqslant 1\}$. The minimal covers are the same as before, for the same reasons, so $Q = \{x, y, z \in [0,1] : x + y \leqslant 1, x + z \leqslant 1, y + z \leqslant 1\}$. Now however, the inequality defining P implies those defining Q, since the variables are positive, so $P \subset Q$. This time, $\left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right) \in Q \backslash P$, so P is the better relaxation.

A more systematic study of how to obtain good relaxations with better formulations will be the object of the chapter on cutting planes.

Total unimodularity:
We now look for a computational criterion that allows us to tell if a polyhedron given by its H-presentation is has integer vertices, in which case it's LP relaxation solves the IP version. This is because the LP is in one of three cases, in all of which we can draw conclusions on the IP:

- If the LP is infeasible, then so is the IP, as the IP has the same constraints but with additional integrality
- If the LP is unbounded, then the underlying polyhedron must contain a ray in $rec(P)$. If $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$, then the ray has to be rational as well, REFERENCE RATIONAL POLYHEDRA. So by scaling it by a common multiple of its entries denominators, we get a integer vector in the ray. So finally, the IP is either infeasible (for example for $P = [1/3, 2/3] \times \mathbb{R} = \{(x, y) : 1/3 \leqslant x \leqslant 2/3\}$), or it has a feasible point and must contain all integer multiples of the ray spanned by the integer vector we found in $rec(P)$. This sequence of points will be unbounded for the objective function, so that the IP is unbounded too in this case.
- Finally, if the LP is feasible and bounded, it atteins it's optimum in a vertex (that we can recorver from the simplex algorithm). If the optimising vertex is integer, it's feasible for the IP and therefore solves the IP.

Recall the characterisation of vertices of the polyhedron $P = \{x : Ax = b, x \geqslant 0\}$ encountered in standard form LP:

Proposition: *A characterisation of vertices*
v is a vertex of $P = \{x : Ax = b, x \geqslant 0\}$ for $A \in \mathbb{R}^{m \times n} \Leftrightarrow v \geqslant 0$ is the unique solution to $A_{*B}x = b$ for some m linearly independent columns of A indexed by B

Now, $A_{*B}^{-1} = \dfrac{1}{det(A_{*B})}(A_{*B})_{adj}$ where $(A_{*B})_{adj}$ is the adjoint matrix of A_{*B} (who's entries are the minor of A_{*B}), if A has integer entries, so does $(A_{*B})_{adj}$, since its entries are sums and products of those of A_{*B}. So if $det(A_{*B}) = \pm 1$, we have the guarantee that A_{*B}^{-1} has integer entries. Finally, if b has integer entries, so does $A_{*B}^{-1}b$. Therefore, for integer data, the condition $det(A_{*B}) = \pm 1$ on any invertible A_{*B} ensures that all vertices are integer.
Summarising:

Definition/Proposition:
For $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$, the vertices of $P = \{x : Ax = b, x \geqslant 0\}$ are integer if for all submatrices A_{*B} of size $m \times m$ we have $det(A_{*B}) \in \{-1, 0, 1\}$, a criterion we'll call **unimodularity** of A.

In practice we ususally deal with polytopes of the form $P = \{x : Ax \leqslant b, x \geqslant 0\}$ which we convert to $Q = \{x : (A|I)x = b, x \geqslant 0\}$ by introducing slack variables. We therefore look for conditions on A so that $(A|I)$ is unimodular, to get a shortcut to integer vertices, for the polyhedra presented with inequalities only.

$(A|I)$ is unimodular if its $m \times m$ submatrices have determinant ± 1. These submatrices are obtained by choosing columns of A and colums of I. By rearanging the columns according to their origin and by rearranging the ordering of dimensions so that the columns of I chosen coorespond to the last numbered dimentions, we are dealing with submatrices of the form $\begin{pmatrix} A' & 0 \\ A'' & I \end{pmatrix}$. Their determinant is therefor $det(A')$, which we'd like to have in $\{-1, 0, 1\}$. This is the condition we retain:

Definition/Proposition:
For $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$, the vertices of $P = \{x : Ax \leqslant b, x \geqslant 0\}$ are integer if for all submatrices A' of A of size $k \times k$ for $k \leqslant m$ we have $det(A') \in \{-1, 0, 1\}$, a criterion we'll call **total unimodularity** of A.

Total unimodularity is a sufficient condition for showing that a polyhedron has integer vertices. However, there are polyhedra with integer vertices who's H-description is not totally unimodular. For example, if we translate by an integer vector the 3-crosspolytope, so as for it to be in $\mathbb{R}^3_+$, it's H-description is given by the matrix $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \\ & \dots & \end{pmatrix}$ and we see that the first 2×2 submatrix corresponding to the first two line and first and last columns has determinant $\begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix} = -2$.

We'll give an example of totally unimodular matrices encountered in LP relaxations of combinatorial optimization problems: the incidence matrices of graphs.

Definition:
For a graph (V, E), the matrix D of dimensions $|V| \times |E|$ defined by $d_{v,e} = \begin{cases} 1 \ if \ v \in e \\ 0 \ else \end{cases}$ is the **incidence matrix** of the graph.

For example, the incidence matrix appears in the LP relaxation two following problems:
Maximum matching: we look for a largest set of edges $M \subseteq E$ so that all vertices of V are incident to at most one edge of M. We can model and relax the problem as:

$$\max \left(\sum_{e \in E} x_e \right) \text{ st. } \begin{cases} \sum_{e \in \delta(v)} x_e \leqslant 1, \ \forall v \in V \\ x \geqslant 0 \end{cases} \quad \Leftrightarrow Dx \leqslant 1, \ x \geqslant 0$$

Minimum vertex cover: we look for a smallest set of vertices $C \subseteq V$ such that all edges have at least one end-point in C. We can model and relax the problem as:

$$\min \left(\sum_{v \in V} y_v \right) \text{ st. } \begin{cases} \sum_{v \in e} y_v \geqslant 1, \ \forall e \in E \\ y \geqslant 0 \end{cases} \quad \Leftrightarrow -D^t y \leqslant -1, \ y \geqslant 0$$

We'll prove the following propositions, so that these LPs solve the combinatorial problems (for bipartie graphs) by interpreting their solutions as indicator vectors: with the following proposition, we know their vertices to be integer, and the constraints allow for no other integers then 0 and 1 as entries.

Proposition:
The incidence matrix of a bipartite graph is totally unimodular.

Proof: We prove this by induction on he size k of the submarices. For $k = 1$, we have the entries of D which are 0 or 1. For the step, we disjoin nested cases and settle the easiest ones first. If the submatrix has a 0-column, the determinant is 0. Otherwise, if there's a colum, with only one non-zero entry, we may develop the determinant (Laplace-formula) along the column, obtain the determinant of another $(k-1) \times (k-1)$ submatrix as determinant and use the induction step to conclude. Finally, we treat the case in which all columns have more then one non-zero entry. Since this is the incidence matrix of a graph, this means that all columns have precisely two non-zero entries. Now, bipartiteness comes into play: the edges connect vertices from different partition sets, meaning that the two non-zero entries of each column correspond vertices of different partitions. So if we compute the determinant by summing separately the rows along the partition sets of the vertices they correspond to, we get the same row as sum, so that the determinant is 0.

Proposition:
If A is totally unimodular, so is its transposed A^t and it's opposite $-A$.

Proof: The submatrices of one are the transposed ones of the submatrices of the other, so that their determinants stay the same. For a change of sign, all determinants might undertake a change of sign, depending on dimension, but in all cases, the determinants remain in $\{-1, 0, 1\}$.

We give a nice application of LP theory to the previous combinatorial problems.
The previous LPs are dual to each other! Strong duality of LPs implies that their optimal values are the same. But we now know that these optimal values are attained in vertices corresponding to actual matchings and covers. This is summarized by:

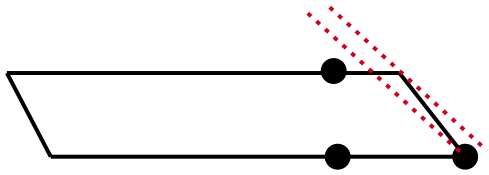
Theorem: *König, Egerváry, 1931*
In a bipartite graph the maximum size a matching is also the minimum size of a vertex cover.

Ex.EdgeCol: An edge coloring of a graph is an assignment of colors to edges so that two edges with a common endpoint have different edge colors. We're interested in the minimum number of colors necessary for such a coloring. Formulate this problem as an IP and interpret the dual of the LP relaxation.
Hints: Interpret coloring as partitioning the edges into matchings. Relate the dual to the maximum degree of vertices of the graph.

To comńclude, we give a final result that shows the strength of investigating these combinatorial problems from a polyhedral perspective. In b-matching, where given a number b_v per vertex v and seek a set of edges so that along these edges v is matched with at most b_v other vertices. From a combinatorial viewpoint, this is a problem quite new to us, and may require entirely new techniques. But for the bipartite case, when using the IP formulation with x_e indicating the use of edge e, the constraints are $\sum_{u \in \delta(v)} x_{vu} \leqslant b_v$. Indeed, the constraint matrix is the node-edge incidence matrix of a bipartite graph, so it's TU and the corresponding polyhedron is integer. So we can solve bipartite b-matching with an LP.

Ex.Ori: Consider the problem of orienting the edges of a graph so that in the resulting digraph, all vertices have in-degree at most k, for a given k. Give an algorithm to solve this.

We end this paragraph by an example that shows that MILPs can't be solved by solving a fully integer constraint version of them, and then solving an LP on the real variables, fixing the integer ones to an IP optimal solution:



Assume that we have a binary y-variable and a real x one. The red lines indicate the objective level sets. If we solve for x integer, we get the lower right integer point. Then, for that y value, solving the LP in x leads to the same point. Yet, the optimum is the upper right corner. By picturing the intersection of the level lines with the upper line staying fixed while we slide the intersection with the lower line to the right, we see that the distance between optimal solutions can become arbitrarily large.

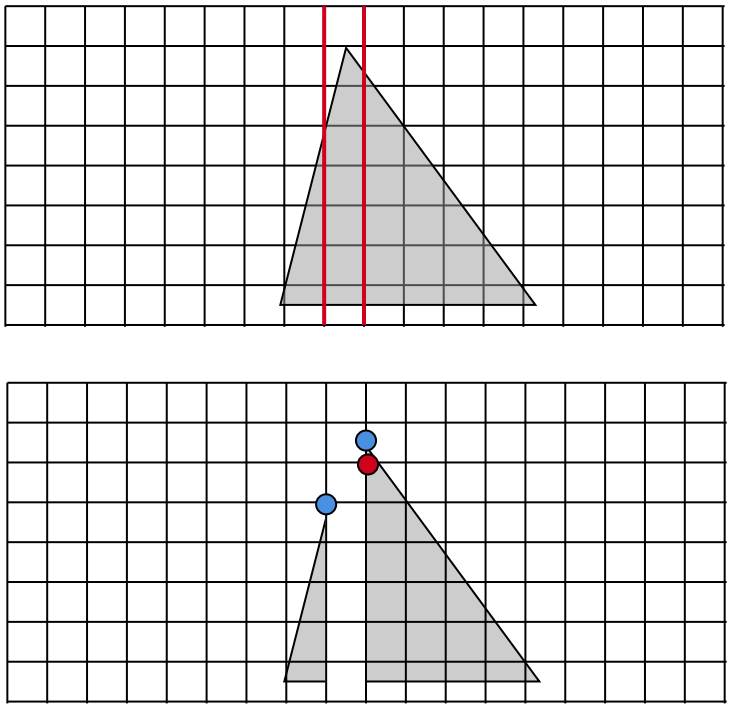
11.3 Branch-and-bound

INTERNAL NOTE: this section requires LP post-optimisation due to adding constraints.

We now make our first step in solving IPs.
The general idea is that of adding constraints to the LP relaxation of the IP that will make its solution "closer" to that of the original IP. The whole difficulty of this approach is finding constraints that exclude no feasible points of the IP. In branch-and-bound, we deal with this problem by reducing the problem to 2 sub-problems: one with the additional constraint, and one with its negation. The simplest such constraints is for example of form $x_i \leqslant k$ for $k \in \mathbb{Z}$. We can use it to split the IP on $\begin{cases} Ax \leqslant b \\ x \in \mathbb{Z}^n \end{cases}$ into two on $\begin{cases} Ax \leqslant b \\ x_i \leqslant k \\ x \in \mathbb{Z}^n \end{cases}$ and $\begin{cases} Ax \leqslant b \\ x_i \geqslant k \\ x \in \mathbb{Z}^n \end{cases}$ and by integrality of x, we can actually split $\begin{cases} Ax \leqslant b \\ x \in \mathbb{Z}^n \end{cases}$ into $\begin{cases} Ax \leqslant b \\ x_i \leqslant k \\ x \in \mathbb{Z}^n \end{cases}$ and $\begin{cases} Ax \leqslant b \\ x_i \geqslant k+1 \\ x \in \mathbb{Z}^n \end{cases}$. Indeed, the optimal solution to the first IP is in one of the two splitting halves of the polyhedra, so that the solution of the two sub-problems with the best objective value will be optimal solution to the first IP.

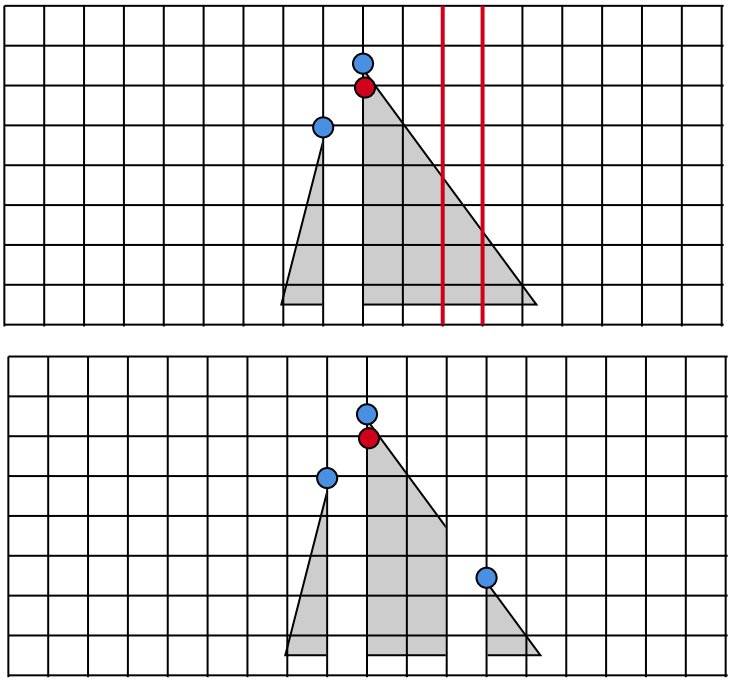
This may not seem like a great improvement at first, since the subproblems are as hard to solve as the main one. The interesting thing is to see how this split interacts with the LP relaxation of IPs. For the relaxations, the LP hasn't exactly split: the solutions for which $k < x_i < k+1$ are discarded. We therefore expect them to give more precise information then the relaxation on the initial polyhedron.

For example, consider the figure below: the grid joints represent the integer points and our polyhedron is the gray triangle. We want to maximise y for integer points in the polyhedron. The LP relaxation returns the top of the triangle as solution, which isn't integer.
We then make our first cut, cutting on $x \leqslant k$ and $x \geqslant k+1$, represented by the red lines.
In the next figure below, we get the two polyhedra resulting from the cut. The blue points are the solutions to the respective LP relaxations, and the red point is the optimal solution to the initial IP. We see that the solution on the right polyherdon is much closer to the IP solution then that of the initial relaxation, whereas the one of the second solution is much worse.



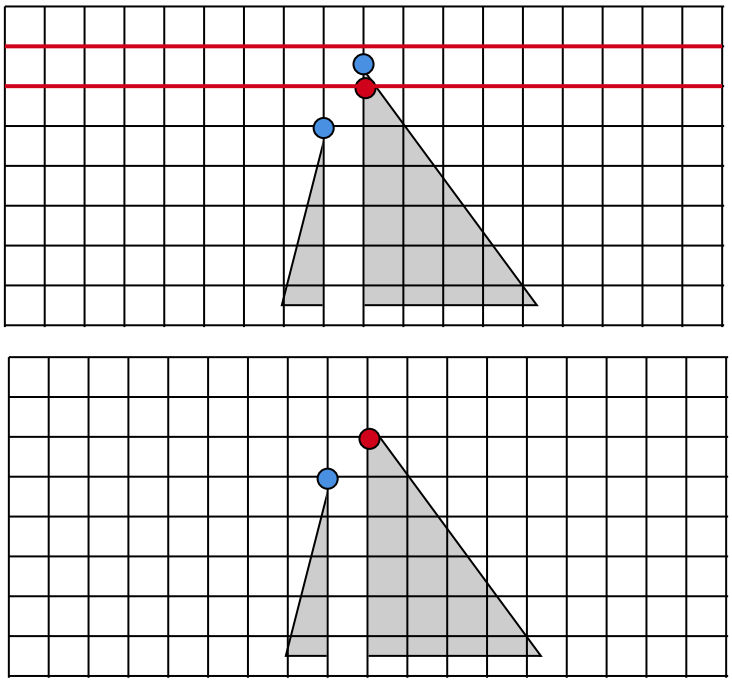
Note that the solution of the LP on the left polyhedron is integer. This is good because it allows us to say that all integer points of the left polyhedron will give an objective value less then that of this solution. There is therefore no need to investigate the left polyhedron further.

There is a second advantage of finding a integer solution to an LP after a split. If we make another split as in the figure below, we see that the LP on the right polyhedron has an optimal value less then that found in the left polyhedron.
Since the solution on the left polyhedron is integer and we now know it's value to be greater then the optimal one on the right polyhedron, we know that all integer points of the right polyhedron have sub-optimal values. We can therefore ignore the right polyhedron in our search.



Our first split seemed reasonable, because the we split close to the optimal solution of the LP relaxation, and intuition suggests that the optimal integer solution can't be far away from the LP solution. Also, this split prohibited the solution to the first LP from also being the solution to one of the two sub-LPs. Since our ultimate goal is to find an integer solution, this last fact is a sign of progress.
Our second split, for example, wasn't that helpful as the middle polyhedron has the same LP solution as the right one before the second split.
In branch-and-bound, we split with $x_i \leqslant k$ and $x_i \geqslant k+1$ where i is an index for which the previous optimal solution was fractional. This maintains the integral coordinates of the optimal solution.

With this splitting rule, the second split would be the one in the figure below. This split illustrates a last case that can appear in a split: on of the sub-problems is infeasible. Here, we see that the upper part of the split is empty. Of course, this restricts our search to the lower half.
In the lower half, the optimal value is attained by an integer point. As in the first cut this means that we don't need to investigate the polyhedron further.
Then, we are done, because all the sub-problems are either infeasible or have integer solutions. Since the solution of the IP must be feasible for on of the sub-problems, this means that the greatest integer solution among the solutions to the sub-problems (here, the blue dot and red dot) is the solution to the IP (here, the red dot).



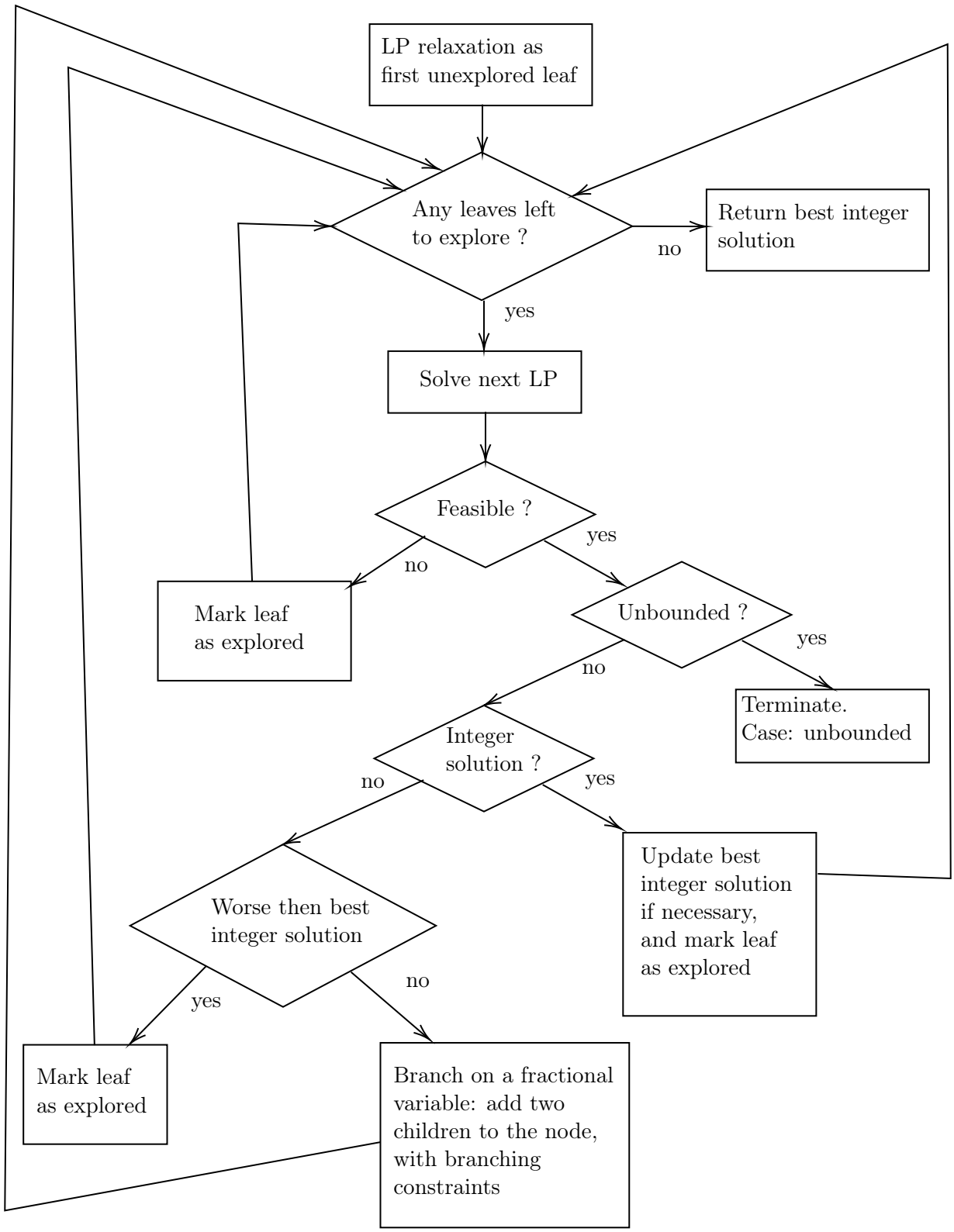
We now summarise the branch-and-bound procedure.
To keep track of the sub-problem hierarchy, we represent the problems with a binary tree. The nodes of this graph represent the problem on a particular polyhedron. The root of the tree is the initial problem. When we split, the node gets two new neighbours, its "children". Each child is the problem on one of the splitting halves of the previous polyhedron.
The leaves of the tree are the sub-problems to be investigated. We investigate them by solving their LP relaxation. Five cases can then occur:

- The LP is infeasible. Then there is on need to split further: we just ignore this node for the rest of the search.
- The LP is unbounded. Then the initial IP is unbounded too.

- The LP is feasible, but its solution has less optimal value then then that of a previously known integer feasible solution. Then we know that all integer solutions to the sub-problem are sub-optimal for the initial problem, and we can ignore the node for rest of the search.
- The LP is feasible, and has an integer optimal solution. Unless this solution falls into the previous case, it's our currently best known integer solution. We therefor ignore the previously best integer solution in our search, and let this new solution be the the new bound for optimal integer solutions.
- Finally, if the LP is feasible, with better solution then the best currently known integer solution, but with non-integral solution, then we split. We split with $x_i \leqslant k$ and $x_i \geqslant k+1$ where i is an index so that x_i is not integral for the current LP solution. There may be multiple choices for this index, so for concreteness, we'll choose the smallest such index.
  This case produces to new child-nodes, new leaves, to be investigated, whereas the current node seizes to be a leaf and will therefor not be considered in further study.

We keep on going like this until no more investigations need to be done. This means that all leaves in the tree correspond to infeasible problems, or problems with known integer solutions. When all leaves are in this state, the best known integer solution (that we kept track of along the way to discard LPs with sob-optimal solutions) is the solution to the IP.

The natural question is then to find out if this procedure terminates. Can we split indefinitely ?
This can't happen when the initial polyhedron is bounded. If we consider the splits on one variable, we'll arrive at inequalities $x_i \leqslant a_i$ and $x_i \geqslant b_i$ that are incompatible with the polyhedron, so that we end in infeasible nodes in these cases. Since there are such a_i and b_i for all indices i, we'll split at most $\Pi(b_i - a_i + 1)$ times.



Branch-and-bound can be adpted to MILPs by branching of fractional values of integer variables.

11.4 Gomory cutting planes

The idea of cutting plane algorithms for solving IPs is that of adding additional contsraints (cutting planes) that don't exclude interger point solutions, but exclude LP relaxation solutions. This has the effect of the relaxation with the additional constraint gives a better bound on the IP and most importantly, that an integer solution to the relaxation solves the IP. Indeed, since all integer point solutions verify the constraints, we have $\begin{cases} Ax \leqslant b \\ x \in \mathbb{Z}^n \end{cases} \Leftrightarrow \begin{cases} Ax \leqslant b \\ Cx \leqslant d \\ x \in \mathbb{Z}^n \end{cases}$ for cutting planes $c_i \cdot x \leqslant d_i$, so that a integer solution on the last problems relaxation is also optimal for the first problem.
These constraints are added successively and we want to find a method of adding these constraints so that we're guaranteed to have one of the relaxations have an integer solution after finitely many additions of constraints.

For example, for $\begin{cases} x + y \leqslant 1,5 \\ x, y \geqslant 0 \\ x \in \mathbb{Z}^n \end{cases} \Leftrightarrow (x, y) \in \{(0,0), (0,1), (1,0)\}$, the constraints $x + y \leqslant 1 + \frac{1}{10^k}$ are cutting planes for all $k \geqslant 1$, as they leave the set of integer solution unchanged. However, if we which to maximise $x+y$, none of the relaxations with underlying polyhedra $\begin{cases} x + y \leqslant 1,5 \\ x, y \geqslant 0 \\ x + y \leqslant 1 + \frac{1}{10^q}, q \leqslant k \end{cases} \Leftrightarrow$ $\begin{cases} x + y \leqslant 1 + \frac{1}{10^k} \\ x, y \geqslant 0 \end{cases}$ will have optimal integer solutions, so this is the type of cutting plane construction we which to avoid.

Gomory cutting planes:
Gomory cuts are cutting planes generated from the simplex algorithm.
For an IP given in LP standard form $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$, we solve its LP relaxation with the simplex algorithm. The constraint part of the final tableau is of form $(I|M)x = d$, where we've permuted variables so that the basic variables of basis B are in the first dimensions. If d is integral, then since the optimal solution is atteined in d (get the solution by setting non-basic variables to 0) and the LP relaxation solves the IP. Otherwise, there is an entry d_i that's not integer.

All solutions, including integral ones, must verify $x_i + \sum_{j \notin B} m_{ij} x_j = d_i$. By cutting m_{ij} into its whole and fractional parts, $m_{ij} = \lfloor m_{ij} \rfloor + (m_{ij} - \lfloor m_{ij} \rfloor)$ and similarly for d_i, we see that $x_i + \sum_{j \notin B} \lfloor m_{ij} \rfloor x_j + \sum_{j \notin B} (m_{ij} - \lfloor m_{ij} \rfloor) x_j = \lfloor d_i \rfloor + (d_i - \lfloor d_i \rfloor)$, and arraging integral and fractional parts on either side, we get $\sum_{j \notin B} (m_{ij} - \lfloor m_{ij} \rfloor) x_j - (d_i - \lfloor d_i \rfloor) = \lfloor d_i \rfloor - x_i - \sum_{j \notin B} \lfloor m_{ij} \rfloor x_j$. If we take a closer look at the right side of the equation, which looks oddly similar to the one we started with, recalling that $x \geqslant 0$, we have $x_i + \sum_{j \notin B} \lfloor m_{ij} \rfloor x_j \leqslant x_i + \sum_{j \notin B} m_{ij} x_j = d_i$. For non-integer x, this inequation can be tight, but for integer x, since then $x_i + \sum_{j \notin B} \lfloor m_{ij} \rfloor x_j$ is integer, we can give a better bound $x_i + \sum_{j \notin B} \lfloor m_{ij} \rfloor x_j \leqslant \lfloor d_i \rfloor$.

This rephrases to $\sum_{j \notin B}(m_{ij} - \lfloor m_{ij} \rfloor)x_j \geqslant (d_i - \lfloor d_i \rfloor)$ for integer solutions, an inequality violated by the optimal solution of the LP relaxation, for which the basic variables are 0, but $(d_i - \lfloor d_i \rfloor) > 0$ as d_i isn't integer.

Definition:
The inequality $\sum_{j \notin B}(m_{ij} - \lfloor m_{ij} \rfloor)x_j \geqslant (d_i - \lfloor d_i \rfloor)$ is a **Gomory cut / cutting plane**.

If the IP isn't in this standard form, we can't necessarily transform it to an IP in standard form. For example, $\begin{cases} 0 \leqslant x \leqslant \sqrt{2} \\ x \in \mathbb{Z} \end{cases}$ isn't the projection of $\begin{cases} x + s = \sqrt{2} \\ x, s \geqslant 0, x, s \in \mathbb{Z} \end{cases}$ as 1 is feasible in the first, but there is no integer slack s so that $1 + s = \sqrt{2}$.
This is problematic for our study because Gomory cuts are based on the standard form.
However, for a rational matrix $A \in \mathbb{Q}^{m \times n}$ and vector $b \in \mathbb{Q}^m$, we see that by multiplying $Ax \leqslant b$ by the gcd of all the entries denominators, we may assume that $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$. In that case, the slack $s = b - Ax$ is integer for integer x. So for rational A and b, we can bring the IP in standard form.

Remarks: We can also use the line corresponding to the costs to generate Gomory cuts. Note that the Gomory cuts coefficients are fractional. If the LP solution with the Gomory cut is still fractional, we can use Gomory cuts again. In that case, we'll have to bring the cut to integer coefficients by multiplying by denominators.

Indeed, the immediate question is whether we'll ever find an integer solution with successive Gomory cuts, or if this procedure could go on forever. The sequence z_n of optimal values obtained by iterating Gomory cuts is monotone and if we assume a minimization IP to be bounded, it must converge to some z.
TO COMPLETE: lexicographic stuff

MIP Gomory cuts:
We now consider MIPs, which differ from IPs in that some of there variables are integer- and others are real-valued. The feasible solutions of a MIP can be thought of as follows: when fixing the integer variables, the continuous variables describe a polyhedron, so the feasible set is a union of disjoint polyhedra, indexed by the values of the integer variables.
Similarly to IPs, we can try to solve MIPs by solving their LP relaxation and hoping that the optimal solution of the relaxation is feasible. As, with IPs, if the latter isn't the case, we can look for cutting planes that are valid for the feasible set and cut off the current relaxation optimum.
When solving the relaxation with the simplex algorithm, we can end up with a feasible optimum or an infeasible optimum. The latter case happens when for some basic integer variable x_i, the optimal tableau has form $x_i + \sum_{j \in N} a_{ij}x_j = b_i$ with $b_i \notin \mathbb{Z}$. If we denote by f_0 the fractional part of b_i, we know that for all feasible solutions of the MIP, $x_i - \lfloor b_i \rfloor = f_0 - \sum_{j \in N} a_{ij}x_j$ is a integer, so that we can perform a branch-and-bound style split. Indeed, the feasible solutions to MIP either verify $f_0 - \sum_{j \in N} a_{ij}x_j \geqslant 1$ or $f_0 - \sum_{j \in N} a_{ij}x_j \leqslant 0$.

At this stage, we could continue with a branch-and-bound approach, but we'll take a different one. Recall that in a cutting plane approach for MIPs, the goal was to find cutting planes valid for all polyhedra of the feasible set. This approach provides the following tool:

Proposition: *Disjunctive programming principle*
For polyhedra P and Q in the positive orthant $\mathbb{R}^n_+$ with valid inequalities $a\cdot x \leqslant b$ and $c\cdot x \leqslant d$ respectively, and for any $e \leqslant \min(a,c)$ (component-wise) and $f \geqslant \max(b,d)$, the inequality $e\cdot x \leqslant f$ is valid for $P \cup Q$.

With this, if we define P as the hull of the feasible MIP solutions for which $f_0 - \sum_{j\in N} a_{ij}x_j \geqslant 1$ holds and Q that of those for which $f_0 - \sum_{j\in N} a_{ij}x_j \leqslant 0$, then we can build new cutting planes for the MIP solutions.

Proof of the principle: $e\cdot x \leqslant a\cdot x$ and $e\cdot x \leqslant c\cdot x$ as $x \geqslant 0$ and $e \leqslant \min(a,c)$, and $a\cdot x \leqslant b \leqslant f$ and $c\cdot x \leqslant d \leqslant f$, so whenever $a\cdot x \leqslant b$ or $c\cdot x \leqslant d$ holds, $e\cdot x \leqslant f$ holds. This is the case on $P \cup Q$. □

We'll use this principle for our MIP and our inequalities $\sum_{j\in N} a_{ij}x_j \leqslant f_0 - 1$ and $\sum_{j\in N}(-a_{ij})x_j \leqslant -f_0$ to get the valid inequality $\sum_{j\in N}(-|a_{ij}|)x_j \leqslant \max(f_0-1, -f_0)$. This inequality does cut of the current relaxation solutions, as setting the non basic variables of N to 0 and remarking that $\max(f_0-1,-f_0) < 0$.

We can actually produce better inequalities, with a deeper analysis. First, when returning to $x_i + \sum_{j\in N} a_{ij}x_j = b_i$ with $b_i \notin \mathbb{Z}$ in the initial optimal tableau, we can note that for integer scalars s_k indexed over the integer variables (I), we have $x_i - \lfloor b_i \rfloor - \sum_{j\in I} s_j x_j = f_0 - \sum_{j\in N} a_{ij}x_j - \sum_{j\in I} s_j x_j$ integer for all feasible MIP solutions. Splitting on $\geqslant 1$ and $\leqslant 0$ as before, we get $f_0 - \sum_{j\in N} a_{ij}x_j - \sum_{j\in I} s_j x_j \leqslant 0$, equivalent to $-\sum_{j\in N} a_{ij}x_j - \sum_{j\in I} s_j x_j \leqslant -f_0$, as well as $f_0 - \sum_{j\in N} a_{ij}x_j - \sum_{j\in I} s_j x_j \geqslant 1$, equivalent to $\frac{f_0}{1-f_0}\left(\sum_{j\in N} a_{ij}x_j + \sum_{j\in I} s_j x_j\right) \leqslant -f_0$ (we do this to simplify the $f \geqslant \max(b,d)$ part of the disjunctive principle). To get the e of the disjunctive principle, we have to find $\min\left(\frac{f_0}{1-f_0}a_{ij}, -a_{ji}\right)$ for the continuous variables of $N\backslash I$ and $\min\left(\frac{f_0}{1-f_0}(a_{ij}+s_j), -(a_{ji}+s_j)\right)$ for the integer ones of $N \cap I$. For the first type of variables, $\min\left(\frac{f_0}{1-f_0}a_{ij}, -a_{ji}\right) = \begin{cases} \frac{f_0}{1-f_0}a_{ij} : a_{ij} \leqslant 0 \\ -a_{ji} : a_{ij} \geqslant 0 \end{cases}$ and for the second type, as we which to get a big e for an inequality that is as tight as possible, we need s_j to bring $(a_{ij}+s_j)$ as close to 0 as possible, as the minimum is over opposite sign multiples of $(a_{ij}+s_j)$.
If f_j denotes the fractional part of a_{ji}, then the closeset we can bring $(a_{ij}+s_j)$ to 0 is by setting $s_j = -\lfloor a_{ij} \rfloor$ or $s_j = -\lfloor a_{ij} \rfloor - 1$, in which case $(a_{ij}+s_j) = f_j$ or $(a_{ij}+s_j) = f_j - 1$. So in the first case the minimum is $-(a_{ji}+s_j) = -f_j$ and in the second it's $\frac{f_0}{1-f_0}(f_j-1)$. So we can choose $s_j \in \{-\lfloor a_{ij} \rfloor, -\lfloor a_{ij} \rfloor - 1\}$ depending on which of $-f_j$ and $\frac{f_0}{1-f_0}(f_j-1)$ is largest. This in turn depends on whether $f_j \geqslant f_0$ as

$\frac{f_0}{1-f_0}(f_j-1) \geqslant -f_j \Leftrightarrow f_j \geqslant f_0$.
Summerising, the cut we produced is:

Definition:
The MIP Gomory cut is

$$\sum_{j\in N\setminus I, a_{ij}\leqslant 0} \frac{f_0}{1-f_0} a_{ij}x_j - \sum_{j\in N\setminus I, a_{ij}\geqslant 0} a_{ij}x_j - \sum_{j\in N\cap I, f_j\leqslant f_0} f_j x_j + \sum_{j\in N\cap I, f_j\geqslant f_0} \frac{f_0}{1-f_0}(f_j-1)x_j \leqslant -f_0$$

INCLUDE: termination of algorithm...

Since IPs con be considered as MIP for which the set of continious variables is empty, an interesting question is to ask what Gomory mixed integer cut look like for IPs.
They have form $\sum_{j\in N, f_j>f_0} \frac{f_0}{1-f_0}(f_j-1)x_j - \sum_{j\in N, f_j\leqslant f_0} f_j x_j \leqslant -f_0$. Is this the integer Gomory cut in disguise or is it a different, better or worse cut ?
In case all $j \in N$ verify $f_j \leqslant f_0$, we do indeed recover the integer Gomory cut. But when $f_j > f_0$, we can compare the coefficients of the cuts with by equivalences $-f_j < \frac{f_0}{1-f_0}(f_j-1) \Leftrightarrow \frac{f_j}{1-f_j} > \frac{f_0}{1-f_0}$ where this last expression is true for $f_j > f_0$ as $x \mapsto \frac{x}{1-x}$ is increasing. Since the variables are positive, $-f_j < \frac{f_0}{1-f_0}(f_j-1)$ for $f_j > f_0$ implies that all variables verifying the Gomory mixed integer cut also verify the Gomory integer cut, so that the first cuts off more of the polyhedron, and is therefore better.

11.5 Branch-and-cut

One can combine branch-and bound and Gomory cuts into a single methods by letting the two alternate. There are multiple variants of branch-and-cut. We present the following: we adapt branch-and-bound by adding cutting planes to a sub-problem before branching on it.
To be precise, we first check if the problem is feasible, if it can't be pruned due to bounds, if it's unbounded and if it's optimal. After these checks, in branch-and-bound, we would branch on a variable. In branch-and-cut, we first start solving the sub-problem with Gomory cuts. That is, we add Gomory cutting planes in the way described in the previous chapter. We can set a limit on the number of cuts to add, say k cuts at most. If the sub-IP has been solved with less then these k Gomory cuts, then we treat this problem like one with optimal integer solution in branch-and-bound, and proceed as in branch-and-bound, which is by solving the next sub-problem if any are left. If the sub-IP hasn't been solved with less then these k Gomory cuts, we branch on a variable as in branch-and-bound.

Branch-and-cut terminates for the same reasons as branch-and-bound (under the same additional assumptions): in the worst case, the branch-and-bound sub-problem tree ends with as many leaves as feasible solutions to the IP. The hope with branch-and-cut is that, depending on the parameter k, certain problems will be solved with cuts only, so there is no need to branch.

11.6 Cutting plane algorithms for combinatorial problems

For problems with a particular structure in the constraints, such as IPs modelling a combinatorial problem, it may happen that we need exponentially many constraints. We now describe how to handle this problem.

TSP:
The travelling salesman prosblem (TSP) provides n cities and travelling costs $c_{ij} \geqslant 0$ between cities i and j (assumed symmetric) and asks for an order in which to visit all cities so that the total travelling cost is minimum. If the costs c_{ij} form a metric (they satisfy the triangular inequality), the tour we seek visits all cities only once, as if skipping previously visited cities forms a tour of lower cost.
We can model the problem by a complete graph K_n with edge weights c_{ij}, so that the tour we seek is a minimum cost cycle passing through all vertices. We can also give an IP formulation by defining indicator variables $x_{ij} \in \{0,1\}$ and objective $\sum_{i,j\in[n]} c_{ij}x_{ij}$. We can try to get a cycle by setting constraints $\sum_{j\in[n]} x_{ij} = 2$ for all vertices i: all cities are traversed once. If you think these constraints are sufficient, let the following be a lesson in rigour to you. Actually, these constraints allow for graphs that are multiple disjoint cycles ! Indeed, we can find such cycles sequentially, starting at a vertex and following the edges for which $x_{ij} = 1$ until we have a repetition of vertices, then continue with a vertex not on this cycle.

So we need constraints that prohibit multiple disjoint cycles, in terms of edges. To this end, we remark that if disjoint cycles occur, then we can partition the vertices into S and $[n]\backslash S$ with $\emptyset \subsetneq S \subsetneq [n]$, and have no edges between vertices of S and $[n]\backslash S$. We can prohibit this with constraints $\sum_{i\in S, j\notin S} x_{ij} \geqslant 1$ for all $\emptyset \subsetneq S \subsetneq [n]$, and more sharply $\sum_{i\in S, j\notin S} x_{ij} \geqslant 2$, as $\sum_{i\in S, j\notin S} x_{ij} = 1$ would imply that all $i \in S$ but a city $k \in S$ have their neighbours in S too so that by walking in S starting from k, we'll reach a contradiction once we run out of vertices of S to walk to.

Ex.TSPineq: We could have prohibited disjoint cycles diffeently then with the constraints $\sum_{i\in S, j\notin S} x_{ij} \geqslant 2$ for $\emptyset \subsetneq S \subsetneq [n]$. Indeed, to prohibit a cycle on vertices $\emptyset \subsetneq S \subsetneq [n]$, one can add the constraint $\sum_{ij\in E(S)} x_{ij} \leqslant |S| - 1$, where the summ is over the edges induced by S: such cycles have $S|$ edges. Show that these families of inequalities are equivalent.

We'll ignore the question of whether these inequalities do or not describe the integer hull of all the indicators of the Hamiltonian cycles. If we solve the LP relaxation of the IP, we'll get a lower bound on the best TSP tour.
Note however that these last inequalities are in number $2^n - 2$, so for a large number of cities, we can't use the simplex method, as our computer won't be able to store the tableaus. But there is hope.

The constraint can be reformulated as saying the graph has no cut of value less then 2. We can check this by finding all minimum s-t-separating cuts for capacities x_{ij}, for all $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs of cities, which can be derived from max-flow algorithms running in polynomial time and space. If one of these cuts has value less then 2, then the corresponding constraint $\sum_{i\in S, j\notin S} x_{ij} \geqslant 2$ is the violated one, and it provides the separating hyperplane, as the feasible y satisfy $\sum_{i\in S, j\notin S} y_{ij} \geqslant 2 > \sum_{i\in S, j\notin S} x_{ij}$. Otherwise, all cuts of the

graph along the bipartiton S and $[n]\backslash S$, which are s-t-separating cuts for some cities $s \in S$ and $t \notin S$ (not unique) and therefore have capacity greater then the minimum s-t-separating cuts, which we have checked to have capacity greater then 2, must therefore have capacity greater then 2, so that $\sum_{i \in S, j \notin S} x_{ij} \geqslant 2$. We therefore know that x is feasible.

With this remark, we present the following method. We start solving the LP without the cut-constraints. We then perform the check of the previous paragraph (this is called a separation oracle). If we get a constraint that is violated, then we add it to the LP a re-solve it. We keep resolving until we find a solution for which the separation oracle tells us that it's feasible.

Alternatively, one can just add the $\frac{n(n-1)}{2}$ additional inequations derived from the minimum cuts, as these imply all other cut-constraints.

11.7 Solutions

Ex.Sudoku:
We can use indicator variables x_{ijk} to express the fact that cell (i,j) of the Sudoku table has entry $k \in [9]$. The Sudoku table contains initial values, for which we add constraints $x_{ijk} = 1$. Next, an cell may contain exactly one entry: $\sum_k x_{ijk} = 1$. Lines, columns and boxes correspond to index sets S: for each such set, we want to put the entry k exactly once, so that $\sum_{ij \in S} x_{ijk} = 1$. This model uses $9^3 = 729$ variables and at least (most) $9^2 + (3.9).9 + 2.9^3 = 1782$ constraints.

Ex.CHH:
We'll prove that the points of $\{x \in \mathbb{Z}^n : a \cdot x \leqslant b\}$ verify the inequality $\left(\frac{1}{g}a\right) \cdot x \leqslant \left\lfloor \frac{b}{g} \right\rfloor$, so that their convex hull does too. This will prove $\{x \in \mathbb{Z}^n : a \cdot x \leqslant b\}_I \subseteq \left\{x : \left(\frac{1}{g}a\right) \cdot x \leqslant \left\lfloor \frac{b}{g} \right\rfloor\right\}$.
So for integer points x, since $\frac{1}{g}a$ is also an integer vector, $\left(\frac{1}{g}a\right) \cdot x$ is an integer with upper bound $\frac{b}{g}$, which we may lower to $\left\lfloor \frac{b}{g} \right\rfloor$.
We then prove by constadition that $\{x \in \mathbb{Z}^n : a \cdot x \leqslant b\}_I \subsetneq \left\{x : \left(\frac{1}{g}a\right) \cdot x \leqslant \left\lfloor \frac{b}{g} \right\rfloor\right\}$ is impossible, so that the sets are equal. So if there's a point y verifying $\left(\frac{1}{g}a\right) \cdot x \leqslant \left\lfloor \frac{b}{g} \right\rfloor$ but not in the convex set $\{x \in \mathbb{Z}^n : a \cdot x \leqslant b\}_I$, we can separate y from $\{x \in \mathbb{Z}^n : a \cdot x \leqslant b\}_I$ with the hyperplane $c \cdot x \leqslant d$, which is valid for $\{x \in \mathbb{Z}^n : a \cdot x \leqslant b\}_I$ but not for y.
We now show that there is an integer point x for which $\left(\frac{1}{g}a\right) \cdot x = \left\lfloor \frac{b}{g} \right\rfloor$, an in particular $a \cdot x \leqslant g \left\lfloor \frac{b}{g} \right\rfloor \leqslant b$, so that $x \in \{x \in \mathbb{Z}^n : a \cdot x \leqslant b\}_I$. This will prevent c and $\left(\frac{1}{g}a\right)$ from being parallel, as the halfspace $\left(\frac{1}{g}a\right) \cdot x \leqslant \left\lfloor \frac{b}{g} \right\rfloor$ contains points points on both sides of $c \cdot x = d$.
To prove this last claim, we use Bézout's identity (theorem from arithmetic) that provides an integer solution to $a \cdot x = g$, and by multiplying this by $\frac{1}{g}\left\lfloor \frac{b}{g} \right\rfloor$ and distributing $\left(\frac{1}{g}a\right) \cdot \left(\left\lfloor \frac{b}{g} \right\rfloor x\right)$, we get the desired result.
As we mentioned, we now now that c and $\left(\frac{1}{g}a\right)$ aren't parallel. A nice trick can then be used to find integer points on the plane $\left(\frac{1}{g}a\right) \cdot x = \left\lfloor \frac{b}{g} \right\rfloor$ for which $c \cdot x \leqslant d$ is false, contradicting its validity for $x \in \{x \in \mathbb{Z}^n : a \cdot x \leqslant b\}_I$ (since $\left(\frac{1}{g}a\right) \cdot x = \left\lfloor \frac{b}{g} \right\rfloor$ implies $a \cdot x \leqslant b$).
This trick uses a fact related to cross-products. We'll find a condition implied by parallelism, which c and $\left(\frac{1}{g}a\right)$ won't therefore have: if $\left(\frac{1}{g}a\right) = v$ and $c = \lambda v$ for some non-zerp scalar λ, the by looking at the components and solving equations for λ, we see that for all pair of indices $\frac{c_i}{v_i} = \frac{c_j}{v_j}$. So in our case, there must be indices so that $c_i v_j \neq c_j v_i$ so that up to permutation of indices, $c_i v_j - c_j v_i < 0$.
The points $p_\lambda = x + \lambda(v_i e_j - v_j e_i)$ for an integer point of $\left(\frac{1}{g}a\right) \cdot x = \left\lfloor \frac{b}{g} \right\rfloor$ are still on $\left(\frac{1}{g}a\right) \cdot x = \left\lfloor \frac{b}{g} \right\rfloor$ since $\left(\frac{1}{g}a\right) = v$ and $v \cdot (v_i e_j - v_j e_i) = 0$. So for all λ for which p_λ is integer, p_λ is in the integer hull and

$c\cdot p_\lambda \leqslant d$ must hold. But $c\cdot p_\lambda = c\cdot x + \lambda(c\cdot(v_ie_j - v_je_i)) > d$ for large enough λ as $c\cdot(v_ie_j - v_je_i) > 0$. This is the contradiction we were looking for.

Next, since $\cap_i \left(H_i^+\right)_I$ is convex and contains the points of $P\cap\mathbb{Z}^d = \left(\cap_i H_i^+\right)\cap\mathbb{Z}^d$, we have $P_I \subseteq \cap_i \left(H_i^+\right)_I$. Integer programming would be easy to solve if the converse inclusion would hold, as we could compute an H-description of P_I and solve the LP on it.

This isn't the case, however, as proves the following polyhedron: $\begin{cases} y \leqslant 2x \\ 2y \geqslant 1-x \\ y \leqslant 2(1-x) \\ 2y \geqslant x \\ x, y \geqslant 0 \end{cases}$ (draw it!).The first and third inequlity prohibit $y \geqslant 2$ and the third and fourth that $x \geqslant 2$ (by contradiction). All 0-1-points are infeasible. So the integer hull of this polyhedron is empty, yet the polyhedron isn't empty (it has $(1/2, 1/2)$) and all the hyperplanes defines halfspaces equal to their integer hull.

Ex.RelaxEx:

Consider the IP $\min(y)$ on the polyhedron $\begin{cases} x \leqslant 1 \\ 2y \geqslant x \\ x, y \geqslant 0 \end{cases}$ (draw it!). The polyhedron is not integer, as $\left(1, \frac{1}{2}\right)$ is a vertex. Yet solving the LP relaxation yields the solution $(0, 0)$ which is also feasible for the IP and thereby solves the IP.

Ex.Knap1:

We prove th contaposition: an x that is infeasible for the standard knapsack formulation must violate one of the $\sum_{i\in C} x_i \leqslant |C| - 1$ constraints for some cover C. So for an x with $\sum_{i\in[m]} w_ix_i > W$, we consider a subset S of smallest size so that $\sum_{i\in S} w_ix_i > W$: such a set exists as at least the set of all items $[m]$ has this property, and the size of a set is an positive integer. Such a set S must actually be a minimum cover, as otherwise, we could delete one of its element and obtain a set of smaller size for the property that S is minimal for. On that set, all $x_i = 1$, for otherwise we would get could delete the indices for which $x_i = 0$ from S and maintain $\sum_{i\in S\setminus\{i:x_i=0\}} w_ix_i > W$, yielding a smaller set for the property that S is minimal for. So finally, $\sum_{i\in S} x_i = |S| > |S| - 1$ and S is the minimal cover that x violates the constraint of, that we were looking for.

Ex.EdgeCol:

For each color, the edges of that color can't have their endpoints in common: they form a matching. So a coloring is a partitionning of edges into matchings. For the set S of matchings, we let y be the indicator vector of the matchings chosen for the coloring. Each edge should be colored with one color, so we get constraints $\forall e \in E$, $\sum_{M\in S: e\in M} y_M = 1$. Our goal is to minimise the number of colors, so our LP is $\min \sum_{M\in S} y_M$ st. $\begin{cases} \forall e \in E,\ \sum_{M\in S: e\in M} y_M = 1 \\ y \in \{0,1\}^{|S|} \end{cases}$. We can relax the LP relaxation further to $\min \sum_{M\in S} y_M$

st. $\begin{cases}\forall e \in E,\ \sum_{M \in S : e \in M} y_M \geqslant 1 \\ y \geqslant 0\end{cases}$, which gives a worse lower bound on the minimum coloring but who's dual is easier to compute. The dual is $\max \sum_{e \in E} x_e$ st. $\begin{cases}\forall M \in S,\ \sum_{e \in M} x_e \leqslant 1 \\ x \geqslant 0\end{cases}$. It asks for a maximum number of edges that aren't in a common matching. This happens when the each pair of such edges have an endpoint in common (otherwise, two disjoint edges form a matching of which we've taken more then one edge). This prohibits such edges to form a path of length 3 or more and the case of a triangle allows for no further edges. To be brief, the only subgraphs these edges can form are stars: one central vertex and only leaves. The size of a stars is the degree of it's central vertex. So the dual is actually looking for the vertex of largest degree. Weak duality now give a natural result: the minimum coloring is greater then the largest degree. This is true because at the vertex of largest degree, all edges must have different colors.

Ex.Ori:
The trick is to represent the problem as follows, reducing it to maximum bipartite b-matching. We represent the edges of the graph by a node set A and the actual nodes of the graph by a second node set B and link $\{u, v\} \in A$ to $u \in B$ and $v \in B$. In a b-matching with $b_{\{u,v\}} = 1$ and $b_v = k$, an edge is assigned to on of its endpoints, which will represent the endpoint the oriented edge will point to, and $b_v = k$ insures that at most k edges point to v, so that its in-degree is at most k. We can solve for the maximum bipartite b-matching by solving an LP, as discussed when we introduced b-matching. If the maximum such b-matching has size $|E|$, all edges can be oriented so as to have the desired property and conversely, such an orientation provides a solution of value $|E|$.

Ex.TSPineq: We relate the sums over the cut associated to S and the edges induced by S. We count all edges that have at least one of their enpoints in S, disjoining on the number of such endpoints: we get $\sum_{i \in S, j \notin S} x_{ij} + 2 \sum_{ij \in E(S)} x_{ij} = \sum_{i \in S, j \in [n]} x_{ij}$ as the edges in $E(S)$ are counted twice, for both endpoints. We'll make use of the other constraints, $\sum_{j \in [n]} x_{ij} = 2$, since $\sum_{i \in S, j \in [n]} x_{ij} = \sum_{i \in S} \sum_{j \in [n]} x_{ij} = 2|S|$. Now we can see that $\sum_{i \in S, j \notin S} x_{ij} \geqslant 2 \Leftrightarrow 2|S| - 2 \sum_{ij \in E(S)} x_{ij} \geqslant 2 \Leftrightarrow \sum_{ij \in E(S)} x_{ij} \leqslant |S| - 1$.

12 More approximation methods

12.1 LP rounding

One can model combinatorial optimization problems as IPs, as we've seen in the section on polyhedron combinatorics. Since solving the IPs exactly is hard, but solving their LP relaxation is easy, a interesting question is whether we can exploit the LP solution to obtain a feasible solution for the combinatorial problem, with value close to the optimal one.

Recall the minimum vertex cover problem introduced in the section on LP relaxations and total unimodularity:
Minimum vertex cover: we look for a smallest set of vertices $C \subseteq V$ such that all edges have at least one end-point in C. We can model and relax the problem as:

$$\min \left(\sum_{v \in V} y_v \right) \text{ st. } \begin{cases} \sum_{v \in e} y_v \geqslant 1, \ \forall e \in E \\ y \geqslant 0 \end{cases}$$

Given an optimal solution to the LP relaxation of the IP formulation of the problem, we can use the following heuristic to produce a vertex cover with good value: if the variable y_v associated to vertex v is close to 1, we add the vertex to the cover set. For example, for a parameter k, we add the vertex to the cover if $y_v \geqslant \frac{1}{2}$. Does this rule produce a vertex cover ? And if so, how good is this cover ?
The inequalities $y_u + y_v \geqslant 1$ per edge (and $y \geqslant 0$) force one of the variables to be greater then $\frac{1}{2}$ (otherwise, both would be strictly less then $\frac{1}{2}$, so that $y_u + y_v < 1$): so all edges have an endpoint in the set $C = \left\{ v : y_v \geqslant \frac{1}{2} \right\}$, which is therefore a cover.
To judge the quality of the cover, we analyse the LP solution's value: $z_{LP} = \sum_{v \in V} y_v = \sum_{v \in C} y_v + \sum_{v \in V \setminus C} y_v \geqslant \frac{1}{2}|C|$ by the definition of C and positivity. For the minimum size of a vertex cover OPT, we know that the relaxation provides a lower bound $OPT \geqslant z_{LP}$, so that our rounding heuristic provides $\frac{|C|}{OPT} \leqslant 2$. This heuristic is known as "LP rounding", and in the case of minimum vertex covers, it provides a 2-approximation algorithm.

Ex.RoundMatch: Find an approximation algorithm for the maximum matching problem (same section as minimum vertex cover) using LP rounding. This won't be actually be an approximation algorithm for general graphs: you have to add an assumption to the solution of the LP relaxation for this to work.

12.2 Randomised LP rounding

In our previous rounding heuristic for the minimum vertex cover problem, we can improve the heuristic to better fit our intuition by including a vertex v in the cover with probability y_v, for an optimal solution y of the LP relaxation of the IP formulation of the problem.
With this heuristic, by using Y_v as the indicator random variable that indicates whether v is included in the cover, the cover has size $C = \sum_{v \in V} Y_v$ which we expect to be $E(Y) = \sum_{v \in V} y_v = z_{LP} \leqslant OPT$. So if C turns out to be a cover, we expect it to be optimal !

Unfortunately, the probability p that C is a cover is hard to bound. By bounding the opposite event that an edge isn't covered by the union of the events that a particular edge isn't covered, we get $1 - p \leqslant \sum_{uv \in E} (1 - y_u)(1 - y_v)$. In an attempt to get a clearer bound, we can use $1 - x \leqslant e^{-x}$, so that $(1-y_u)(1-y_v) \leqslant e^{-(y_u+y_v)} \leqslant e^{-1}$ as $y_u + y_v \geqslant 1$, to get $1-p \leqslant |E|e^{-1}$. This bound is awful, but we can improve on it. We'll show that using the same procedure with different probabilities for vertex inclusion, we get $1-p \leqslant |E|e^{-(k+1)\ln(|E|)} = \frac{1}{|E|^k}$, which is a great bound, for a parameter $k > 0$. Backtracking our reasonning, we see that we can get this desired bound if the probability of edges uv not being covered is $[(1 - y_u)(1 - y_v)]^{(k+1)\ln(|E|)}$. This can be achieve by including a vertex v to the cover with probability $1 - (1 - y_v)^{(k+1)\ln(|E|)}$.

However, this new probability affects the expected size of the cut: now, it's $E(Y) = \sum_{v \in V} 1 - (1 - y_v)^{(k+1)\ln(|E|)}$. To get a better picture of this expectation, we can use the inequality of the mean (calculus) on the segment $[0, y_v] \subset [0, 1]$ and bound the derivative of $f(x) = (1 - x)^{(k+1)\ln(|E|)}$ by $(k + 1)\ln(|E|)$ to get $E(Y) = \sum_{v \in V} 1 - (1 - y_v)^{(k+1)\ln(|E|)} \leqslant (k+1)\ln(|E|) \sum_{v \in V} y_v = (k+1)\ln(|E|)z_{LP} \leqslant (k+1)\ln(|E|)OPT$.
We therefore obtain a $(k + 1)\ln(|E|)$-approximation algorithm who's output is a cover with probability $p \geqslant 1 - \frac{1}{|E|^k}$.

Ex.MaxCutLPround: Consider the Max-cut problem on graph (V, E) with positive edge weights w that asks for a bipartition of vertices into $A \subseteq V$ and $V \backslash A$ so that the weight of the edges in the cut of this bipartition $\sum_{e \in \delta(A)} w_e$ is maximum. Model this as an IP and derive a $\frac{1}{2}$-approximation algorithm for it using randomised rounding.

12.3 Primal-dual method

So far, our strategy to get an approximation ratio for some LP rounding approximation algorithm has been the following: for a minimisation problem with optimal value OPT and LP relaxation value P, we try to get $ALG \leq \alpha OPT$ form a bound of type $ALG \leq \alpha P$ since $P \leq OPT$.
Another strategy could be the following: we consider the dual to the LP, which has optimal value D, and weak duality provides $D \leq P$ so that $ALG \leq \alpha D$ also provides $ALG \leq \alpha OPT$. We could therefore try to use the dual to get heuristics.

The hitting set problem:
In the hitting set problem, we are given a population E and categories $T_i \subset E$. We which to find a committee of individuals $e \in E$ such that all categories are represented in it. Hiring individual e in this committee incurs cost c_e, so our objective is to achieve our representation goal with lowest cost.
We use the characteristic vector x of E so that the conditions translate to $\begin{cases} \sum_{e \in T_i} x_e \geqslant 1, \forall i \in [n] \\ x_e \in \{0,1\}, \forall e \in E \end{cases}$ and we minimise $\sum_{e \in E} c_e x_e$. We can relax this to minimising $\sum_{e \in E} c_e x_e$ over $\begin{cases} \sum_{e \in T_i} x_e \geqslant 1, \forall i \in [n] \\ x_e \geqslant 0, \forall e \in E \end{cases}$ as all optimal solutions on it must verify $x_e \leqslant 1$, since otherwise, one could decrease one of the variables to 1 without violating constraints (the constraints not containing the decreased variable can be ignored, and those containing it are still true if we decrease it to 1, as the variables are positive), but decreasing the objective.
The dual of this LP is that of maximising $\sum_{i \in [n]} y_i$ such that $\begin{cases} \sum_{i : e \in T_i} y_i \leqslant c_e, \forall e \in E \\ y_i \geqslant 0, \forall i \in [n] \end{cases}$.
Complementary slackness informs us that the primal variables that are non-zero are those for which the inequalities of the dual are tight. We can use this as a heuristic to get an algorithm which won't require solving an LP.

The algorithm works as follows. We start with $y = 0$ and raise coordinates of y until one of the inequalities $\sum_{i : e \in T_i} y_i \leqslant c_e$ becomes tight (in which case it stays tight for the rest of the algorithm, by positivity of the variables and the only allowed action being increasing them) and add the corresponding e to the hitting set (committee) A. This algorithm terminates when no more y_i may be increased (by a non-zero amount) without one of the constraints being violated. This is known as a primal-dual algorithm.

We consider the value of the primal-dual algorithm $\sum_{e \in A} c_e$. By our algorithm, we can replace the costs by tightness $\sum_{e \in A} c_e = \sum_{e \in A} \sum_{\substack{i \, : \, e \in T_i \\ tight}} y_i$ where in the second sum y_i appears once for all $e \in A \cap T_i$ with corresponding tight inequality. This happens at most $|A \cap T_i| \leqslant max_i |T_i|$ times, so that $\sum_{e \in A} c_e \leqslant max_i |T_i| \sum_{i \in [n]} y_i$ and by weak duality, we have $\sum_{e \in A} c_e \leqslant max_i |T_i| \sum_{i \in [n]} y_i \leqslant max_i |T_i| . OPT$ where OPT is the value of the cheapest hitting set. This means that the primal-dual algorithm is a $max_i |T_i|$-approximation algorithm.

The Steiner tree problem:
In the Steiner tree problem, we're given a connected graph with positive edge weights and a set of targets/terminals $\tau \subset V$. The problem consists in finding a minimum weight subgraph that connects all

nodes from the terminal to each other. This graph must be a tree, as such a graph must also be inclusion minimal for connectivity, because of positive weights.

To model the problem as an IP, we use the usual indicators x for the edges. If the graph connects all the vertices of τ, then for any node set S "separating" τ in the sense that $\emptyset \neq S \cap \tau \neq \tau$, the cut $\delta(S)$ must contain at least one edge (for example on a path from $t_1 \in S \cap \tau$ to $t_2 \notin S \cap \tau$). This actually characterises this property of connectivity: conversely, if all such $\delta(S)$ are non empty, we start with $S = \{t_1\} \subset \tau$ and iteratively augment S along edges of its cut $\delta(S)$ until all nodes of τ are contained in S. The loop invariant of this miniature algorithm is that the number of nodes in S allways increases, so that the algorithm terminates, and that the set of edges augmented along connects the nodes of S, allowing us to obtain with the desired connectivity property for τ.

So the problem can be modeled by the IP $\min \sum c_e x_e$ subject to $\begin{cases} \sum_{e \in \delta(S)} x_e \geqslant 1, \forall \emptyset \neq S \cap \tau \neq \tau \\ x_e \in \{0,1\} \end{cases}$.

We can relax it to $\min \sum c_e x_e$ subject to $\begin{cases} \sum_{e \in \delta(S)} x_e \geqslant 1, \forall \emptyset \neq S \cap \tau \neq \tau \\ x_e \geqslant 0 \end{cases}$ as no optimal solution has entries greater then 1, since those could be lowered to 1 without violating constraints.

The dual of $\min \left(c \cdot x : x^t A \geqslant b^t, x \geqslant 0\right)$ is $\max(b \cdot x : Ax \leqslant c, x \geqslant 0)$, so the dual to the privious problem is $\max \sum_{\emptyset \neq S \cap \tau \neq \tau} y_S$ subject to $\begin{cases} \sum_{\emptyset \neq S \cap \tau \neq \tau \atop e \in \delta(S)} y_S \leqslant c_e, e \in E \\ y \geqslant 0 \end{cases}$.

Ex.STPD: Find a primal-dual algorithm for the Steiner tree problem.

The prize collecting Steiner tree problem:
A telecommunication company wants to build a network on a graph. Their center is the node r, building cables on edge e costs $c_e > 0$ and connecting node v to r brings income $\pi_v > 0$. To maximise profit is to minimise costs, and maximise the sum of the incomes of visited nodes. By adding the constant $-\sum_{v \in V} \pi_v$ to the objective, we see that the last task is equivalent to minimising the sum of the incomes of nodes left out of the network.
We models this as an IP with indicator variables x_e for the edges and y_v for the nodes, representing wether the edge is used and the node is connected.
The problem is to minimise the objective $\sum_{e \in E} c_e x_e + \sum_{v \in V} \pi_v (1 - y_v)$.

Ex.PCSTcons: Show that the constraints $\sum_{e \in \delta(S)} x_e \geqslant y_v$ over all $S \subseteq V \backslash r$ and all $v \in S$ force the graph to have the following property: for an optimal solution, $y_v = 1 \Leftrightarrow r$ and v are connected in the graph induced by $\{e : x_e = 1\}$. Here, $\delta(S)$ is the set of edges with exactly one endpoint in a subset of vertices S. Next, show that the graph induced by $\{e : x_e = 1\}$ for the optimal solution to the IP is a tree.

Rounding via primal-dual, exercise 7.6.

12.4 TSP

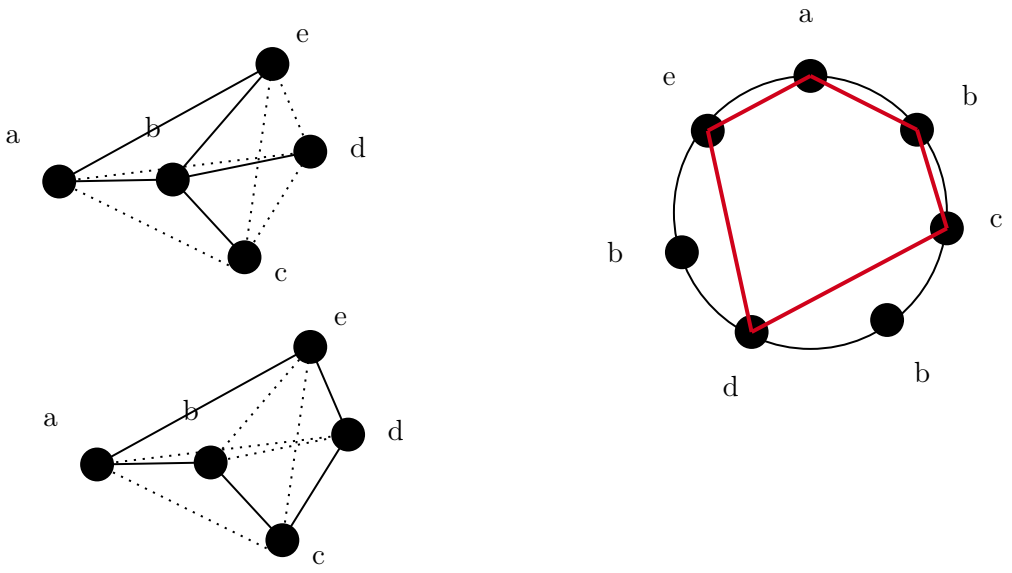
We saw that the TSP is inapproximable. If we add constraints on the weights of the TSP, the situation changes drastically, as we can then use a technique called "short-cutting".

Definition:
The **metric TSP** is the TSP but for positive weights satisfying the triangular inequality $w_{uv} \leqslant w_{us} + w_{sv}$.

By remarking that if we delete an edge in a TSP tour, we obtain a tree spanning the vertices, we know that for the optimal value OPT of a TSP tour and a MST on the associated complete weighted graph of value τ, we have $OPT \geqslant \tau$.

The technique of "short-cutting" is the following: if we have a circuit (edges may be repeated) that visits all vertices, we can create a TSP cycle with smaller value then it. To build it, we consider then order in which the vertices first appear on the circuit, an we build the TSP cycle by taking the edges in that order. By the triangular inequality, this edge will have smaller weight then that of the walk on the cycle connecting these two vertices. Also, this walk can't contain any unvisited vertices, as this would contradict the order of appearance property of our construction. So the resulting cycle visits all vertices and has smaller total weight then the circuit.

For example in the figure below, we start with a graph on the top left and a circuit on the right. Short-cutting results in the red cycle, which corresponds to the cycle in the complete graph on the bottom left.



We can relate TSP cycles and spanning trees as follows: we walk around the tree so as to pass each edge of it exactly twice and get a circuit passing all vertices this way. To see that this can be done, use induction on the vertices by considering the circuit for the tree without a leaf, and inserting the leaf into the circuit by tacking the edge going to it twice back and forth.
This "double tree" heuristic provides a 2-approximation algorithm by short-cutting the circuit we built. Indeed, the circuit has weight at most $2\tau \leqslant 2OPT$, as the edges are take twice at most.

We can improve this approach to the Chirstofides heuristic. In it, we replace the previous circuit by an Eulerian circuit. To get an Eulerian circuit, we need all vertices to have even degree, which isn't naturally provided for the spanning tree. We'll therefore add edges to this tree in a clever way.

We want to add edges between the odd (degree) vertices of the tree to make them even. On such edge per pair of odd nodes would be enough, so that adding a perfect matching between odd vertices would solve the problem. Perfect matching exist if the number of odd vertices are even in number which is the case (due to the handshake lemma and a split on parity). We can take a minimum perfect matching of the odd vertices, which turns out to have value less then $\frac{1}{2}OPT$. To see this, we'll build a perfect matching of the odd vertices $\frac{1}{2}OPT$. We'll build it from the optimal TSP tour: we build a cycle of the odd vertices by short-cutting the optimal TSP tour in order of appearance of the odd vertices on it. This is an even cycle of value less then OPT. By 2-coloring its edges, each color forms a perfect matching of the odd vertices, one of which must have value less then $\frac{1}{2}OPT$ (for the weights are positive).

In conclusion, the Eulerian circuit on the MST joined by the minimum perfect of its odd vertices, has value at most $OPT + \frac{1}{2}OPT$. So we end up with a TSP cycle of value at most $\frac{3}{2}OPT$ by short-cutting one last time.

Finally, we discuss the nearest/cheapest insertion heuristic, who's analysis is also based on spanning trees. In the nearest-insertion heuristic, we start our cycle at an arbitrary vertex v_0 and at each iteration, we start by finding a vertex v_k who's closest to the current vertices of the cycle $v_0, ..., v_{k-1}$ (in the sense that it minimises $\min_i(c_{uv_i})$ for $u \in V \backslash \{v_0, ..., v_{k-1}\}$). We then insert this vertex in our cycle in the cheapest way possible, in the sense that we replace the edge $(v_i v_{i+1})$ by $(v_i v_k)$ and $(v_k v_{i+1})$, so as to minimise the net insertion cost $c_{v_i v_k} + c_{v_k v_{i+1}} - c_{v_i v_{i+1}}$, and over all $i \leqslant k-1$. The algorithm ends once the all vertices are visited by the tour.
The nearest-insertion heuristic provides a 2-approximation algorithm. To see this, we must remark 2 things. First, for the sequence of edges $(v_i v_k)$ attaining the minimum defining v_k in the kth iteration, these edges actually constitute a minimum spanning tree of K_n as they are added in the same way as for Prim's/Jarnik's algorithm. We've seen previously that the value of this tree τ is less then that of the value OPT of the optimal TSP tour. So the goal is to show that our output cycle has value less then 2τ. To this end, we have the second remark: if $(v_j v_k)$ attains the minimum defining v_k in the kth iteration and if $(v_i v_{i+1})$ is replaced as it minimises $c_{v_t v_k} + c_{v_k v_{t+1}} - c_{v_t v_{t+1}}$, then $c_{v_i v_k} + c_{v_k v_{i+1}} - c_{v_i v_{i+1}} \leqslant 2c_{jk}$. If we sum these inequalities, we get the final value of the output tour on the left and 2τ on the right, showing the desired approximation property. So, to see that $c_{v_i v_k} + c_{v_k v_{i+1}} - c_{v_i v_{i+1}} \leqslant 2c_{jk}$, note that $c_{v_i v_k} + c_{v_k v_{i+1}} - c_{v_i v_{i+1}} \leqslant c_{v_j v_k} + c_{v_k v_{j+1}} - c_{v_j v_{j+1}}$ by definition of i, and that $c_{v_j v_k} + c_{v_k v_{j+1}} - c_{v_j v_{j+1}} \leqslant 2c_{jk}$ which is equivalent to $c_{v_k v_{j+1}} \leqslant c_{v_j v_{j+1}} + c_{jk}$ (we abused notation $c_{jk} = c_{v_j v_k}$), which is the triangular inequality.

12.5 Cuts and Metrics

Multiway cut:
In the multiway cut problem, we're given a graph $G = (V, E)$ with positive edge weights w and a set of nodes $S \subseteq V$ to separate: we look for the minimum weight set of edges F so that in $(V, E \backslash F)$, there is exactly one element of S in each connected component.
We can model this as an IP by letting these connected components play an important role: if C_s is the component of $s \in S$, then we consider variables $x_{v,s}$ indicating if $v \in C_s$ and $z_{e,s}$ indicating if $e \in \delta(C_s)$. So if edge e is in the multiway cut, exactly two $z_{e,s} = 1$, corresponding to the two s of the connected components the endpoints of e are in. This way, our objective to minimise is $\frac{1}{2} \sum_{e \in E} w_e \sum_{s \in S} z_{e,s}$.
As for constraints, we want one component per vertex, so $\sum_{s \in S} x_{v,s} = 1$ for all $v \in V$, and for coherence $x_{s,s} = 1$. Next, we note that for edge $\{u, v\}$, $|x_{u,s} - x_{v,s}|$ indicates whether both endpoints are or aren't both in C_s. So adding constraints $z_{\{u,v\},s} \geqslant x_{u,s} - x_{v,s}$ and $z_{\{u,v\},s} \geqslant x_{v,s} - x_{u,s}$ (equivalent to $z_{\{u,v\},s} \geqslant |x_{u,s} - x_{v,s}|$) for all edges and components forces one exactly two of the $(z_{e,s})_{s \in S}$ to be 1, indicating that e cuts these components apart.

This LP relaxation of this IP turns out to have a beautiful geometric formulation. Since the weights are positive and we minimise the objective, we expect the optimal solution to be tight for one of $z_{\{u,v\},s} \geqslant x_{u,s} - x_{v,s}$ and $z_{\{u,v\},s} \geqslant x_{v,s} - x_{u,s}$ (otherwise we could decrease $z_{\{u,v\},s}$ without violating any other constraints, thereby decreasing the objective). In other words, for an optimal solution (x^*, z^*) we have $z^*_{\{u,v\},s} = |x^*_{u,s} - x^*_{v,s}| = \max(x_{u,s} - x_{v,s}, x_{v,s} - x_{u,s})$. Then the optimal value can be re-written as $\frac{1}{2} \sum_{e \in E} w_e \sum_{s \in S} z^*_{e,s} = \frac{1}{2} \sum_{\{u,v\} \in E} w_{\{u,v\}} \|x^*_u - x^*_v\|_1$ by considering the x^*_u to be in $\mathbb{R}^{|S|}$. Then $\sum_{s \in S} x^*_{v,s} = 1$ means that x^*_u is in the standard $(|S| - 1)$-simplex in $\mathbb{R}^{|S|}$, and $x^*_{s,s} = 1$ means that x^*_s is the s-th unit vector of $\mathbb{R}^{|S|}$, a vertex of the standard $(|S| - 1)$-simplex.

How can we get an approximation algorithm from this interpretation ?
The solutions of the LP relaxation valid for the IP modeling the problem correspond to the unit vectors e_i, the vertices of the simplex, as they represent the unique choice of component to put the node into. We can guess that if a variable x_v is "close" to one of these unit vectors, then putting v in the corresponding component should yield a good result.
There are different ideas that can come to mind when making the decision of how to interpret x_v. For example, we could triangulate the simplex along a Voronoi diagram on the vertices of the simplex and let v be in component i if x_v is in the Voronoi cell of e_i. Another idea is that of computing the distance $d_{v,i}$ of x_v to e_i and letting v be in component i with probability $\frac{1/d_{v,i}}{\sum_{i \in S} 1/d_{v,i}}$.
However, when it comes to judging the quality of an approximation algorithm, it's often useful to be able to easily relate it to the optimal value of the main problem. In the case where a relaxation is used, we want to be able to easily relate it to the relaxation, as we can then relate it to the optimal value of the main problem. This is where the distance interpretation of the objective of the relaxation comes into play: if we use this notion of distance to decide which is the closest e_i to x_v, we should be able to analyse the resulting algorithm more easily.
We could put v in component i if $\|v - e_i\|_1$ is minimum among the $\|v - e_j\|_1$ for $j \in S$.
But in a final step, we introduce randomisation. Instead, we'll pick a random radius $r \in]0, 2[$ and a random order in which to traverse S, and v in component i if $\|v - e_i\|_1 \leqslant r$, passing over the $i \in S$, putting the unattributed v into the component of $j \in S$ that comes last. The choice $r \in]0, 2[$ is due to the fact that for any two points in the simplex a and b, $\|a - b\|_1 \leqslant \sum |a_i - b_i| \leqslant \sum |a_i| + |b_i| = 2$ (H-description of

the simplex $\sum x_k = 1$).

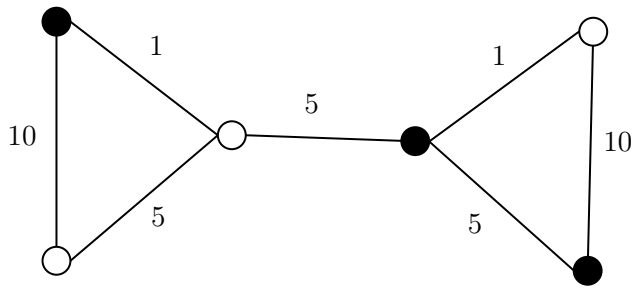
For such an algorithm, edge $\{u, v\}$ is in the cut if there is an $i \in S$ that isn't the last to appear in the random ordering of S so that exactly one of u or v verifies $\|x_u - e_i\|_1 \leqslant r$ or $\|x_v - e_i\|_1 \leqslant r$. This endpoint will be attributed to component i, while the other will be attributed to another. It turns out that bounding this probability is easier then computing it.
With a union bound, we can upper-bound the probability of this occuring by $\sum_{i \in S} P_i$, where P_i is the probability of this event for $i \in S$ being the first element in the random ordering of S for which this happens.

Tree metrics and buy-at-bulk network design:

12.6 Local Search

A local search algorithm works by creating a sequence of feasible solutions to an optimization problem who's objective value increases, until a stopping criterion is reached. Many algorithms we've seen so far can be considered local search algorithms: for example, the simplex method moves from vertex to vertex, improving the cost at each iteration, and the Ford-Fulkerson algorithm improves flow by augmenting paths.
Local search may not terminate in a global optimum of the problem, in the sense that the algorithm may find a solution it can't further improve, but this solution isn't optimal. In such cases, one hopes that the algorithm is at least an approximation algorithm. We now study a case in which this is true.

We'll provide a $\frac{1}{2}$-approximation algorithm for max-cut for edge weights in $\mathbb{N}$ using local search. We start with a feasible solution for the max-cut problem, a bipartition (for example the bipartition $V = \emptyset \amalg V$), and try to improve it. The idea is that we can look at nodes individually and see which partition set would suit them best. If the bipartition is into $V = A \amalg V\backslash A$ and a node $v \in A$ satisfies $\sum_{u\in\delta(v)\cap A} w_{uv} > \sum_{u\in\delta(v)\cap V\backslash A} w_{uv}$, then this means that by changing v to $V\backslash A$, total weight of the cut increases by $\sum_{u\in\delta(v)\cap A} w_{uv} - \sum_{u\in\delta(v)\cap V\backslash A} w_{uv} > 0$, as the egdes of $\delta(v)\cap A$ enter the cut while those of $\delta(v)\cap V\backslash A$ leave it.
We can repeat such improvements until all nodes $v \in A$ satisfy $\sum_{u\in\delta(v)\cap A} w_{uv} \leqslant \sum_{u\in\delta(v)\cap V\backslash A} w_{uv}$ and no further improvement can be done. Before asking if this procedure ever terminates, we ask: is such a bipartition an optimal one ?



In the above graph, all nodes yield no improvement to the cut when switching partition. Yet, if we let the endpoints of the edge joining the triangles switch partition sets simultaneously, we improve the cut by $2(5-1) = 8$, showing that the cut wasn't optimal.

However, an un-improvable bipartition has a value that who's ratio to the optimal one can be bounded. Indeed, summing $\sum_{u\in\delta(v)\cap A} w_{uv} \leqslant \sum_{u\in\delta(v)\cap V\backslash A} w_{uv}$ over $v \in A$ provides $2\sum_{u\in E(A)} w_{uv} \leqslant \sum_{uv\in\delta(A)} w_{uv}$ as each edge of with both endpoints in A is sumed once for each endpoint and as each edge of $\delta(A)$ has one of its endpoints v in A and the other in $\delta(v) \cap V\backslash A$. This identity can push us towards finding a bound of $W = \sum_{e\in E} w_e$ by a factor of the weight of the cut $\sum_{uv\in\delta(A)} w_{uv}$, so that by positivity of weights, W upper-bounds the maximum cut and the algorithm turns out to be an approximation algorithm.
So if we do the same for the other partition set $V\backslash A$ as we did for A, we obtain $2\sum_{u\in E(V\backslash A)} w_{uv} \leqslant \sum_{uv\in\delta(V\backslash A)} w_{uv}$. By adding the two identities together, dividing by 2 on both sides and adding $\sum_{uv\in\delta(A)} w_{uv}$

to both sides of the inequation, we obtain $W \leqslant 2 \sum_{uv \in \delta(A)} w_{uv} \Leftrightarrow \frac{1}{2} W \leqslant \sum_{uv \in \delta(A)} w_{uv}$. This means that if our improvement algorithm terminates, the bipartition it ends up with is greater then half the optimal one, making this a $\frac{1}{2}$-approximation algorithm for max-cut.

We conclude by showing that the algorithm really works, so that the algorithm terminates. A strategy to prove termination for local search algorithms is to show that the ever increasing value increases by at least some minimum amount, and that it is bounded: this way, it can't improve forever, as it would eventually breach the bound.
Here, we improve the cut by a quantity of form $\sum_{u \in \delta(v) \cap A} w_{uv} - \sum_{u \in \delta(v) \cap V \setminus A} w_{uv} > 0$. By considering integer weights, this means $\sum_{u \in \delta(v) \cap A} w_{uv} - \sum_{u \in \delta(v) \cap V \setminus A} w_{uv} \geqslant 1$ so that we increase the cut by at least 1. As the cut is at most $W = \sum_{e \in E} w_e$ by positivity of weights, we make at most $\frac{W}{1} = W$ improvements.

12.7 Solutions

Ex.RoundMatch:
Here we round by considering $M = \left\{e : x_e > \frac{1}{2}\right\}$: this is indeed a matching, as at any vertex, only one edge of M may be present, as otherwise, we'd violate a constraint because $\sum_{e\in\delta(v)} x_e > 2\frac{1}{2} = 1$. A problematic case can arise for cases such as odd cycles on which the x_e have value $\frac{1}{2}$ or complete graphs on n vertices on which the x_e have value $\frac{1}{n-1}$: this may be an optimal LP solution for which this heuristic produces an empty matching. Then $|M| = 0$, but non-empty matchings exist, so that it's impossible to obtain an approximation algorithm from this heuristic. So we must restrict ourselves to special cases, and add the assumption that we deal with graphs for which the LP relaxation returns a solution for which M isn't empty.
To get a ratio, we analyse the LP solution: $\sum_{e\in E} x_e = \sum_{e\in M} x_e + \sum_{e\in E\setminus M} x_e \leqslant 1.|M| + \frac{1}{2}(|E| - |M|)$, since the constraints imply $x_e \leqslant 1$.
Finally, using $OPT \leqslant z_{LP}$ (relaxation), we have $1 \leqslant \frac{OPT}{|M|} \leqslant \frac{1}{2} + \frac{|E|}{|M|} \leqslant \frac{1}{2} + |E|$, so that $\frac{|M|}{OPT} \geqslant \frac{1}{0,5 + |E|}$.

Ex.MaxCutLPround:
We use variable y_v to indicate if $v \in A$. We want the edge $\{u, v\}$ in the cut if $y_v + y_u \in \{0, 1, 2\}$ has value 1. If $x_{\{u,v\}}$ indicates if $\{u, v\}$ is in the cut, then $x_{\{u,v\}} \leqslant y_v + y_u$ prohibits $\{u, v\}$ from being in the cut when $v \notin A$ and $u \notin A$ and $x_{\{u,v\}} \leqslant 2 - (y_v + y_u)$ prohibits $\{u, v\}$ from being in the cut when $v \in A$ and $u \in A$. If only one endpoint is in A, then these bound become $x_{\{u,v\}} \leqslant 1$, so that $x^*_{\{u,v\}} = 1$ for a maximum solution to the objective $\sum_{e\in E} w_e x_e$, as $w \geqslant 0$, so that if $x^*_{\{u,v\}} = 1$ weren't the case, we could increase $x^*_{\{u,v\}}$ to 1 without violating other constraints (as none involve the variable), thereby increasing the objective and contradicting maximality.
Therefore the solutions to $\max \sum_{e\in E} w_e x_e$ st. $\begin{cases} x_{\{u,v\}} \leqslant y_v + y_u, \forall\{u, v\} \in E \\ x_{\{u,v\}} \leqslant 2 - (y_v + y_u), \forall\{u, v\} \in E \\ x \in \{0, 1\}^{|E|},\ y \in \{0, 1\}^{|V|} \end{cases}$ indicate the maximum cuts of the graph.
For optimal solutions x and y to the LP relaxation of this IP formulation, we remark that $x_{\{u,v\}} = \min(y_v + y_u, 2 - (y_v + y_u))$, as by the constraints $\leqslant$holds and if it were strict, we could increase $x_{\{u,v\}}$ without violating constraints, thereby increasing the objective and contradicting maximality.
In our randomised rounding algorithm, we let v be in A with probability y_v (independently of the others). The probability of $\{u, v\}$ being in the cut is then $P((v \notin A \cap u \in A) \amalg (u \notin A \cap v \in A)) = y_u(1 - y_v) + y_v(1 - y_u)$.
To relate this to $x_{\{u,v\}}$ by a quality ratio r, we seek $r > 0$ so that $y_u(1-y_v)+y_v(1-y_u) \geqslant r \min(y_v+y_u, 2-(y_v+y_u))$ as this makes the algorithm an r-approximation algorithm, as weights are positive (multiply by $w_{\{u,v\}}$ and sum).
To find r, we look for $\min\left(\frac{x + y - 2xy}{\min(x + y, 2 - x - y)}\right)$ over $[0, 1]^2$. We disjoin two cases, the first being $x + y \leqslant 2 - x - y$ which happens on the standard 2-simplex in the plane ($x + y \leqslant 1$). Here, we minimise

$\frac{x+y-2xy}{x+y}$ who's differential $\left(\frac{-2y^2}{(x+y)^2}, \frac{-2x^2}{(x+y)^2}\right)$ doesn't vanish in the interior, so that the minimum of the function must be attained on the boundary. We know that the extremum we look for is a minimum as the function tends to $+\infty$ as we go to $(0,0)$. There are no critical points on the boundaries $x=0$ and $y=0$, but there is one on the boundary $y=1-x$, at $\left(\frac{1}{2},\frac{1}{2}\right)$ where the function takes value $\frac{1}{2}$. The second case can be reduced to the first by noticing that the function is symmetric wrt. $y=1-x$, as $2-(1-x)-(1-y)=x+y$ and $(1-x)+(1-y)-2(1-x)(1-y)=x+y-2xy$. So $r=\frac{1}{2}$ does the job, and it's the best possible ratio for this technique.

Ex.STPD:
We use the classic primal-dual heuristic with a twist. We follow the idea develloped to show the converse of the prolyhedral version of the connectivity property. So we start with a set $T=\{t_1\}\subset\tau$ and increase y_T until one of the constraints $\sum_{\substack{\emptyset\neq S\cap\tau\neq\tau \\ e\in\delta(S)}} y_S \leqslant c_e$ becomes tight, and add the (a) corresponding edge e to the tree we're building, adding its new endpoint to T. We repeat this step until T is connects all the terminals, which we check by looping on $\neg\tau\subset T$. The resulting graph is a tree connecting all the terminals for the same reasons as before. Let's investigate its weight. The weight is $\sum_{e\in T} c_e = \sum_{e\in T} \sum_{\substack{\emptyset\neq S\cap\tau\neq\tau \\ e\in\delta(S),\ tight}} y_S$, so that we count y_S at most $|T\cap\delta(S)|$ times, implying $\sum_{e\in T} c_e \leqslant \sum_{\emptyset\neq S\cap\tau\neq\tau} |T\cap\delta(S)| y_S$. By a crude bound $|T\cap\delta(S)|\leqslant|T|\leqslant|V|-1$ and weak duality, we can conclude that $\sum_{e\in T} c_e \leqslant (|V|-1)\sum c_e x_e \leqslant (|V|-1)OPT$, so that this is an $(|V|-1)$-approximation algorithm.

Ex.PCSTcons:
If $y_v=1$, then $\sum_{e\in\delta(v)} x_e \geqslant 1$, so that v has at least one edge of the graph induced by $\{e : x_e=1\}$ (we'll call it T). For one such edge, we add it's other endpoint to $S=\{v\}$ and repeat for $\sum_{e\in\delta(S)} x_e \geqslant y_v = 1$. We keep doing this, adding successively nodes to S that are coonet to S by an edges of T, implicitely building a tree rooted a v, until we encounter r: this must happen, as there are only finitely many vertices, so the worst case scenario is that S becomes $V\backslash r$. There is therefore a path from v to r in T.
Conversely, if there is a path from v to r in T, for any set S with $v\in S$ and $r\notin S$, there must an edge of T in the cut $\delta(S)$. Thus for all such S, $\sum_{e\in\delta(S)} x_e \geqslant 1 \geqslant y_v$, so that we may set $y_v=1$ without violating constraints. Since the solution is minimal and $y_v=1$ yields smaller cost then $y_v=0$, $y_v=1$.
Finally, T must be a tree, because we know that it must contain paths to all vertices for which $y_v=1$ and also minimise the first part of the objective function. So it must use as little edges as possible while staying connected: this defines trees.

13 Computational geometry

13.1 Convex hulls

13.2 Polyhedra

H-redundancy:
Given a polyhedron $Ax \leqslant b$, we'd like to know if an inequality $A_{i*}x \leqslant b_i$ is redundant, in th sense that all points verifying $A_{[m]\setminus i}x \leqslant b_{[m]\setminus i}$ also verify $A_{i*}x \leqslant b_i$. Solving this problem is equivalent to solving an LP, since if we know how to solve it, one can perform binary search on δ by checking for redundancy of $c \cdot x \leqslant \delta$ in $\begin{pmatrix} A \\ c \end{pmatrix} x \leqslant \begin{pmatrix} b \\ \delta \end{pmatrix}$ as routine.
Conversely, we can solve this problem with an LP: $\max A_{i*}x$ st. $A_{[m]\setminus i}x \leqslant b_{[m]\setminus i}$. If the optimum is $\leqslant b_i$, then line i is redundant and otherwise it isn't. If the program is infeasible, then the line is technically redundant, though this is meaningless as the polyhedron is empty.

H-dimension:
Given a polyhedron $Ax \leqslant b$, we'd like to know it's dimension, in the sense that we'd like to know indices of the inequalities that are always tight, as well as a point in the relative interior of the polyhedron, or a way to tell if the polyhedron s empty.
Again, this is equivalent to solving an LP, as one can perform binary search on δ by checking for emptiness of $\begin{pmatrix} A \\ c \end{pmatrix} x \leqslant \begin{pmatrix} b \\ \delta \end{pmatrix}$ as routine.
Conversely, we can use LPs to solve this problem. We can solve 2 LPs per line, $\max A_{i*}x$ st. $Ax \leqslant b$ and $\min A_{i*}x$ st. $Ax \leqslant b$: if the value coincide, then the inequality is tight for all points. The intersection of these hyperplanes will be the affine hull of the polyhedron. For those lines I for which the the values are different, we get points $x_{i,max}$ and $x_{i,min}$. Then $x = \sum_{i \in I} \frac{1}{2|I|}(x_{i,max} + x_{i,min})$ is in the polyhedron as a convex combination and it verifies $A_I x < b_I$, showing that it's in the relative interior of the polyhedron (as we'll soon show). If the LPs are infeasible, or the max is less then the min, then the polyhedron must be empty.

Alternative methods with different runtime exist. First, we solve $\max y$ st. $Ax + y\mathbf{1} \leqslant b$ and disjoin cases. If the optimum is $y^* > 0$ (possibly infinite), then the polyhedron is full dimensional and corresponding x^* is an interior point. Indeed, $Ax^* < b$ and for $r = \frac{1}{2} \min\left(\frac{b_i - A_{i*}x^*}{\|A_{i*}^t\|}\right) > 0$, for all $v \in B(0,r)$, we have $A(x^* + v) \leqslant Ax^* + \left(\left\|A_{i*}^t\right\| \|v\|\right)_i$ by Cauchy-Schwartz and then $A(x^* + v) < b$ by our choices.
If the optimum is $y^* < 0$, the the polyhedron must be empty, as the existence of an x for which $Ax \leqslant b$ can be extended to a solution of the LP by $y = 0$, contradicting maximality.

If the optimum is $y^* = 0$, things get interesting. We consider the dual $\min\left(b^t s\right)$ st. $A^t s = 0$, $1^t s = 1$ and $s \geqslant 0$, which must be feasible by duality, for which complementary slackness yields interesting results. Since $s^* \neq 0$ by $1^t s^* = 1$ and $s^* \geqslant 0$, the set $I = \{i : s_i^* > 0\}$ isn't empty, which corresponds to the set of lines for which $A_{i*}x \leqslant b_i$ is tight, by complementary slackness for any feasible x, as any feasible x can be extended to an optimal solution of the LP by $y = 0$.
We now know that the hyperplanes indexed by I , of which there is at least one, contain the affine hull of the polyhedron. We can then reduce dimension by solving $A_I x = b_I$ as $Bz + x_0$ where B is a basis of the kernel of A_I, via Gaussian elimination. Substituting in the initial equations, we get a new polyhedron $(AB)z \leqslant (b - Ax_0)$ in a lower dimension, since z has lower dimension then x.
We then iterate this procedure, taking at most d iterations as we reduce dimension by at least one at each step.

V-dimension:
Given a polyhedron P with vertices V and ray generators R, how does one find the dimesnion of the polyhedron ?

By considering the homogenization H_P of the polyhedron with ray generators $\begin{pmatrix} V & R \\ 1^t & 0^t \end{pmatrix}$, for which the polyhedron is the intersection of H_P with the hyperplane $x_{d+1} = 0$, considered inside that hyperplane, we simplify the analysis. Indeed, affine independent points of P correspond to linear independent points of H_P. Therefore, the dimension of P is $rank\left(\begin{pmatrix} V & R \\ 1^t & 0^t \end{pmatrix}\right) - 1$.

V-redundancy:
Given a polyhedron $P = \left\{Vc + Rp : c, p \geqslant 0, 1^t c = 1\right\}$, how do we tell is a generator $v \in V$ or $r \in R$ is redundant, in the sense that $P = \left\{(V \backslash v)c + Rp : c, p \geqslant 0, 1^t c = 1\right\}$ for example ?
We can actually solve this problem by using H-redundancy. This is because of a connection between H- and V-redundancy for cones. Indeed, if i is redundant for the H-cone $Ax \leqslant 0$, then there is no x for which $A_{[m]\backslash i} x \leqslant b_{[m]\backslash i}$ and $A_{i*}x > b_i$ so that by Farkas' lemma (v.2 in Ziegler) there must exist $s \geqslant 0$ so that $A^t_{i*} = \sum_{j \in [m]\backslash i} s_j A^t_{j*}$, meaning that A^t_{i*} is redundant for the cone generated by A^t. Since the converse direction hold, we get the equivalent.
So to find the redundant generators of $P = \left\{Vc + Rp : c, p \geqslant 0, 1^t c = 1\right\}$, we homogenize it (as redundancy is maintained when homogenizing) and use the equivalence and the algorithm for H-redundacy. To see that redundancy is maintained when homogenizing, assume that $v = (V \backslash v)c + Rp$ or $r = (R \backslash r)p$, then equivalently $\begin{pmatrix} v \\ 1 \end{pmatrix} = \left(\begin{pmatrix} V \\ 1 \end{pmatrix} \backslash \begin{pmatrix} v \\ 1 \end{pmatrix}\right) c + \begin{pmatrix} R \\ 0 \end{pmatrix} p$ since $1^t c = 1$ and $\begin{pmatrix} r \\ 0 \end{pmatrix} = \left(\begin{pmatrix} R \\ 0 \end{pmatrix} \backslash \begin{pmatrix} r \\ 0 \end{pmatrix}\right) p$. Here, we use the fact that that a ray is redundant if it is redundant in R.

Fourier-Motzkin elimination:
Fourier-Motzkin elimination has many aspects to it. It can be used to solve the linear system feasibility problem, thereby solving LPs, it can be used to compute the projection of a polyhedron on a coordinate-subspace and it can be used to find the H-description of a V-cone. It also provides a constructive proof of the Farkas Lemma.

Fourier-Motzkin elimination eliminates a variable x_i in a system $Ax \leqslant b$ as follows. We disjoin indices on the signs of coefficients at i into $U = \{j : a_{ji} > 0\}$, $E = \{j : a_{ji} = 0\}$ and $U = \{j : a_{ji} < 0\}$. We can then rewrite the system as $\begin{cases} x_i + \sum_{k \neq i} \frac{a_{jk}}{|a_{ji}|} x_k \leqslant \frac{b_j}{|a_{ji}|}, j \in U \\ -x_i + \sum_{k \neq i} \frac{a_{jk}}{|a_{ji}|} x_k \leqslant \frac{b_j}{|a_{ji}|}, j \in L \\ \sum_{k \neq i} a_{jk} x_k \leqslant b_j, j \in E \end{cases}$ or alternatively $\begin{cases} x_i \leqslant \frac{b_j}{|a_{ji}|} - \sum_{k \neq i} \frac{a_{jk}}{|a_{ji}|} x_k, j \in U \\ x_i \geqslant \sum_{k \neq i} \frac{a_{jk}}{|a_{ji}|} x_k - \frac{b_j}{|a_{ji}|}, j \in L \\ \sum_{k \neq i} a_{jk} x_k \leqslant b_j, j \in E \end{cases}$. If this system has a solution, that is if we can find actual x such that this holds, then in particular for all pairs $j \in U$ and $j' \in L$, $\sum_{k \neq i} \frac{a_{jk}}{|a_{ji}|} x_k - \frac{b_j}{|a_{ji}|} \geqslant \frac{b_{j'}}{|a_{j'i}|} - \sum_{k \neq i} \frac{a_{j'k}}{|a_{j'i}|} x_k$.
If we consider this as a system over variables $[n]\backslash i$, then we get a total of $|L| \times |U| + |E|$ inequalities. We can then investigate the converse: if this new system has a solution, can we extend it to the a solution of the previous system ?

If the previous system has a solution, then for it $\max_{j \in L} \left(\sum_{k \neq i} \frac{a_{jk}}{|a_{ji}|} x_k - \frac{b_j}{|a_{ji}|} \right) \leqslant \min_{j \in U} \left(\frac{b_j}{|a_{ji}|} - \sum_{k \neq i} \frac{a_{jk}}{|a_{ji}|} x_k \right)$,
so if we choose x_i in the $\left[\max_{j \in L} \left(\sum_{k \neq i} \frac{a_{jk}}{|a_{ji}|} x_k - \frac{b_j}{|a_{ji}|} \right), \min_{j \in U} \left(\frac{b_j}{|a_{ji}|} - \sum_{k \neq i} \frac{a_{jk}}{|a_{ji}|} x_k \right) \right]$ non-empty interval,

then we recover a solution to the previous system (in the cases that U or L are empty, think of it as $\max = -\infty$ and $\min = +\infty$).

Fourier-Motzkin elimination of i corresponds to projecting a polyhedron on the space $x_i = 0$, in the sense that given an H-description of the polyhedron, the system we derived during elimination of variable i is an H-description of the $x_{[n]\setminus i}$ such that there exists an x_i for which x is in the initial polyhedron. We get in particular the the projection of an H-polyhedron is still an H-polyhedron.
Fourier-Motzkin can be used to solve the LP-feasibility by eliminating all variables one after the other. In the final step, after eliminating $x_1, ..., x_{n-1}$, we get a system with equations of type $x_n \leqslant b'_i$ or $x_n \geqslant b'_i$ which has a solution if the lower bounds are lower then the upper bounds. This is a priori not efficient methods, since at each step, the new system has $|L| \times |U| + |E|$ inequalities. If the initial number of inequalities was m, then since $|L| \times |U| + |E| = |U|((m - |E|) - |U|) + |E| \leqslant \frac{(m-|E|)^2}{4} + |E| \leqslant \max\left(m, \frac{m^2}{4}\right)$, we have a squared number of inequalities in the worst case. So after n eliminations starting with m equations, we get $\frac{m^{(2^n)}}{4^{(2^n+1)}}$ (since the powers of 4 grow as $g_{k+1} = 2g_k + 1$, so $g_k = 1 + 2^k$) in the worst case (a priori).
Finally, Fourier-Motzkin can be used to get the H-description of a V-cone with the following trick. We can write $C = \{Bs | s \geqslant 0\}$ as $C = \{x : x = Bs, s \geqslant 0\}$ which is $C = \left\{x : \begin{cases} x \geqslant Bs \\ x \leqslant Bs \end{cases}, s \geqslant 0\right\}$ in H-description. This isn't an H-description of C as we're not in the dimension of x. However, we can now use Fourier-Motzkin to eliminate the s_i successively. The system we obtain after these eliminations is one for x only and has form $Ax \leqslant 0$, since at all elimination stages $b = 0$. If it has a solution, then by backtracking Fourier-Motzkin (successive extensions), we can find an (or multiple) s that produce it in the V-cone.

Farkas Lemmas:

Cone Frakas Lemma:

For a V-cone $B\mathbb{R}^d_+$ and a vector b, either $b \in B\mathbb{R}^d_+$ or he can separate b from $B\mathbb{R}^d_+$ by a vectorial hyperplanes, so that $h^t b < 0$ and $h^t B\mathbb{R}^d_+ \geqslant 0$.

We can solve LP-feasibility problem $b = Bs$ and $s \geqslant 0$ for the first problem and LP $\min h^t b$ st. $h^t B \geqslant 0$ for the second (as conic combinations won't affect dot-product signs).

We can also give constructive proof with Fourier-Motzkin. Since $\begin{cases} b = Bs \\ s \geqslant 0 \end{cases} \Leftrightarrow \begin{cases} b \geqslant Bs \\ b \leqslant Bs \\ s \geqslant 0 \end{cases}$, we can treat b like a variable and eliminate s in the last H-polyhedron to get a system $Ab \leqslant 0$, which has a solution precisely when the previous system does, that is when $b \in B\mathbb{R}^d_+$. So when this isn't the case $Ab \leqslant 0$ must be infeasible, so that there is a line with $A_{i*}b > 0$, and we can set $h^t = -A_{i*}$.

Standard Frakas Lemma:

A polyhedron in standard form $Ax = b$, $x \geqslant 0$ is either non-empty, or we can find a contradictory combination of its defining equations in the form of $\lambda^t A \geqslant 0$ and $\lambda^t b < 0$ for some λ.

Proof: This is just the cone version said differently: for a fixed b, b is in the cone when the system $b = A^t\lambda$, $\lambda \geqslant 0$ has a solution for λ, and the separating hyperplane, when expressions are transposed, provides $A^t h \leqslant 0$ and $b^t h > 0$.

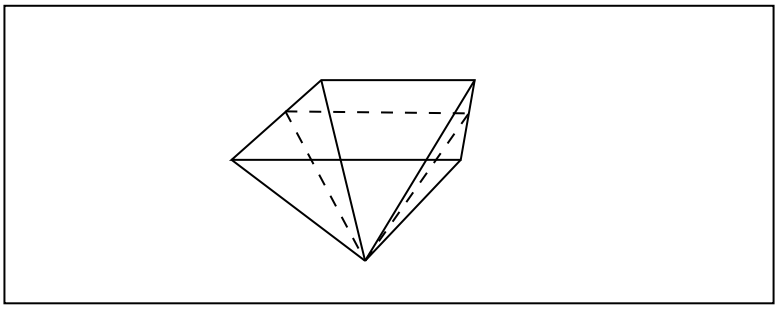
Polyhedron Frakas Lemma:

An H-polyhedron $Ax \leqslant b$ is either non-empty, or we can find a contradictory combination of its defining equations in the form of $\lambda^t A = 0$ and $\lambda^t b < 0$ for some $\lambda \geqslant 0$.

Proof: It's equivalent to the standard-Farkas, useing the transforms of standardisation. Indeed, if $Ax \leqslant b$ has a solution, then we can split it along signs and add slack so as to get a solution $(A, -A, I)\begin{pmatrix} x^+ \\ x^- \\ s \end{pmatrix} = b$ with $\begin{pmatrix} x^+ \\ x^- \\ s \end{pmatrix} \geqslant 0$, and vice-versa by forgetting slack and $x = x^+ - x^-$. By the standard-Farkas, the last existance equivalent to the non-existance of c such that $c^t(A, -A, I) \geqslant 0$ and $c^t b < 0$. To conclude, we note that $c^t(A, -A, I) \geqslant 0$ is equivalent to $\pm c^t A \geqslant 0$, so $c^t A = 0$ and $c \geqslant 0$, which are the conditions from the polyhedron-Farkas.

Computing the V-desciption of an H-cone:
We start from the following remark, which provides an V-description of a certain type of H-cones. They are of form $Ax \leqslant y$ and arise in LP-feasibility for cones (phase 1 of the simple algorithm), and decomposing x into any basis b_i of $\mathbb{R}^d$ yields feasible points $x_i(b_i, Ab_i)$, to generate the y, we add the $(y_i - (Ax)_i)(0, e_i)$, which is a conic combination as $Ax \leqslant y$ (where e_i is in the dimensions of y), so that $(x, y) = \sum x_i(b_i, Ab_i) + \sum (y_i - (Ax)_i)(0, e_i)$. We therefore get a set of generators made of the $\pm(b_i, Ab_i)$ and $(0, e_i)$.
Next, the H-cone $Ax \leqslant 0$ is the intersection of $Ax \leqslant y$ with the subspace $y = 0$, aka the intersection with the subspaces $y_i = 0$ for all $i \in [m]$. The key point is that for a V-cone, we can compute the generators of it's intersection with a coordinate subspace. So we can do this for the V-descrption of $Ax \leqslant y$ we just established, and proceed successively to find the V-descriptions of intersection with the $y_i = 0$, until we have a V-description of $Ax \leqslant 0$.

The idea of this intermediate step is that to get the extremal rays of the cone on the subspace, we need to look for conic combinations of extremal ray on either side of the subspace:



In the figure, the intersection of this truncated cone with the hyperplane is shown in dotted lines. The edges of the intersection are conic combinations of those of the cone.

We therefore divide generators g_i according to $U = \{j : g_{ij} > 0\}$, $E = \{j : g_{ij} = 0\}$ and $L = \{j : g_{ij} < 0\}$, if we seek the intersection with $x_i = 0$. If $i \in U$ and $k \in L$, then $g_{ji}g_k + (-g_{jk})g_j$ is a conic combination who's jth coordinate is 0, as desired. The question is if any conic combination of g_i for which the jth coordinate is 0 can be written as a conic combination of $g_{ji}g_k + (-g_{jk})g_j$ for all pairs $i \in U$ and $k \in L$ and of g_i for $i \in E$, since this would give us a generating set of the intersection of the cone with the subspace.

So if $x = \sum_i s_i g_i$ is such a combination, then we split $x = \sum_U s_i g_i + \sum_E s_i g_i + \sum_L s_k g_k$ and note the requirement that $\sum_U s_i g_{ij} + \sum_L s_k g_{kj} = 0$. This rephrases to $\sum_U s_i g_{ij} = \sum_L s_k(-g_{kj}) > 0$, a quantity we'll call W. To introduce the pairings, we express $\frac{1}{W}Wx$ by "crossing" the sets U and L, so that $x = \frac{1}{W}\sum_U \left(\sum_L s_k(-g_{kj})\right) s_i g_i + \sum_E s_i g_i + \frac{1}{W}\sum_L \left(\sum_U s_i g_{ij}\right) s_k g_k$. We group expressions to get $x = \sum_{i\in U, k\in L} \frac{s_i s_k}{W}(g_{ji}g_k + (-g_{kj})g_i) + \sum_E s_i g_i$. This is a conic combination of the vectors we expected to generate the intersection of the cone and the subspace !

Note that we get $|U| \times |L| + |E|$ generators for $|U| + |L| + |E|$ ones at each intersection, so that there may be an exponential number of generators at the end of the process.

Double description methods:
The double description methods are essentially more efficient methods for the task of the previous paragraph.

We note that the task of finding the V-description of an H-cone allows to do the reverse and the same for polyhedra. For the reverse, we perform the main direction for the duals, and use the explicit formulas for duals, which switch between V-/H-description. For polyhedra, we use homogenization and scaling.

The standard double description method assumes that we deal with a H-cone $Ax \leqslant 0$ that is pointed and irredundant. We can reduce the problem for arbitrary H-cones to this case. First, we compute a basis $(b_i) \approx B$ of the lineality space $Ax = 0$, which we complete to an basis of all of the space with $(q_i) \approx Q$, which are orthogonal to the lineality space (find a base of $B^t q = 0$). The intersection of the cone with the orthogonal to the lineality space is a pointed cone described by $A(Qy) \leqslant 0$: it's pointed since if it contained a line, that line would also be one of the main cone, hence it would be in the lineality space, and can't be in its orthogonal complement. We can remove redudancies of this cone by the method described at the start of this chapter. We can then use the standard double description method to get V-descrption R for $(AQ)y \leqslant 0$. Then QR generates the cone that is the intersection of the main cone with the orthogonal to the lineality space. All that we have left to do now is to find generators of the lineality space, since any point of the cone is the sum of it's projection on the lineality spaces orthogonal and some vector of the lineality space. It turns out that the (b_i) and $b' = -\sum b_i$ (non-zero as B is linearly independent) generate the lineality space as a cone, and are in minimum number for this property. Indeed, since B is a basis, the elements of the lineality space can be written as $x = \sum s_i b_i$, which, if it isn't a conic combination already (if $\min_j(s_j) \geqslant 0$), can be written as $x = \sum(s_i - \min_j(s_j))b_i + |\min_j(s_j)|b'$, which is a conic combination (if $\min_j(s_j) < 0$). We can even show that this number of generators is minimum. If the dimension of the lineality space is l, then $l + 1$ generators at least are necessary: for less then l, the lineality space isn't spanned, and for l exactly, for generators $g_1, ..., g_l$, we can consider the hyperplane spanned by $g_1, ..., g_{l-1}$ and a point orthogonal to it, on the other side of it then g_l, which can't be generated as its component

along g_l would have to be negative.

The standard double description method is iterative and maintains a pair of matrices (A, R) such that the H-cone $Ax \leqslant 0$ and the V-cone $cone(R)$ coincide. We start with a pair $\left(A_{i*}, R^{(1)}\right)$ and at each step, for a subset $K \subseteq [m]$ and $i \in [m]\backslash K$, starting from $\left(A_{K*}, R^{(k)}\right)$, we build $\left(A_{(K\ \cup i)*}, R^{(k+1)}\right)$, until $K = [m]$. Initially, for $\left(A_{i*}, R^{(1)}\right)$, since $A_{i*} \neq 0$, we can take for $R^{(1)}$ the generators of the space $A_{i*}x = 0$ (as described for the lineality space), together with $-A_{i*}$. In the step, we perform a method similar to the one developed in the previous section.

For $\left(A_{K*}, R^{(k)}\right)$ and $i \in [m]\backslash K$, the generators r of $R^{(k)}$ fall into three categories: those $R_+^{(k)}$ for which $A_{i*}r > 0$, those $R_0^{(k)}$ for which $A_{i*}r = 0$ and those $R_-^{(k)}$ for which $A_{i*}r < 0$. Only the latter two are in the final cone, as the first violate $A_{i*}x \leqslant 0$. We again consider the intersections of segments spanned by generators on either side of $A_{i*}x = 0$ and show that they generate $A_{(K\ \cup i)*}x \leqslant 0$. Similarly to the previous section, the intersections are $r_{jh} = (A_{i*}r_j)r_h - (A_{i*}r_h)r_j$, for $r_j \in R_+^{(k)}$ and $r_h \in R_-^{(k)}$. They are in the cone generated by $R^{(k)}$ as conic combinations and satisfy $A_{i*}x = 0$, so that by the recursion hypothesis, they are in $A_{(K\ \cup i)*}x \leqslant 0$, and so are their conic combinations. To see that they actually generate the cone, together with the generators of $R_-^{(k)}$ and $R_0^{(k)}$, we consider a point x in the cone $A_{(K\ \cup i)*}x \leqslant 0$, which is also in the cone $A_{K\ *}x \leqslant 0$, and has expression $x = \sum \lambda_t r_t$ there, for $r_t \in R^{(k)}$ and $\lambda \geqslant 0$. If no generator of $R_+^{(k)}$, we are done. Otherwise, we will rewrite this expression to see that its in the cone of $R^{(k+1)}$. Since $A_{i*}x \leqslant 0$ and one generator r_j of $R_+^{(k)}$ is present, there must also be one $r_h \in R_-^{(k)}$ (otherwise $A_{i*}x > 0$). In fact, there must be $r_j \in R_+^{(k)}$ and $r_h \in R_-^{(k)}$ such that $\lambda_h(A_{i*}r_h) + \lambda_j(A_{i*}r_j) \leqslant 0$, for otherwise, we'd get $\sum_{r_h \in R_-^{(k)}} \lambda_h(A_{i*}r_h) + |R_-^{(k)}|\lambda_j(A_{i*}r_j) > 0$ and by considering this for the r_j for which $\lambda_j(A_{i*}r_j)$ is minimum, we'd get a bound $A_{i*}x = A_{i*}\sum \lambda_t r_t > 0$, contradicting that x is in the cone.
For such j and h, by adding $x + \dfrac{\lambda_j}{(A_{i*}r_h)}r_{jh} = \left(\sum \lambda_t r_t\right) + \left(\dfrac{\lambda_j}{(A_{i*}r_h)}(A_{i*}r_j)r_h - \lambda_j r_j\right)$ and noting that the right side doesn't contain r_j anymore, and is still an conic combination of $R^{(k)}$ as $\lambda_h + \dfrac{\lambda_j}{(A_{i*}r_h)}(A_{i*}r_j) \geqslant 0$ (as $(A_{i*}r_h) < 0$), while the left side is a point of $A_{(K\ \cup i)*}y \leqslant 0$, we can do this iteratively until the right side contains no more generators of $R_+^{(k)}$. The left side will then by $x + \sum \mu_t r_t$ for $\mu < 0$ and r_t of the form $r_{jh} = (A_{i*}r_j)r_h - (A_{i*}r_h)r_j.$, so that by bringing $\sum \mu_t r_t$ to the right, we get a conic combination of $R^{(k+1)}$. This concludes the proof that $\left(A_{(K\ \cup i)*}, R^{(k+1)}\right)$ is a DD-pair.

At this stage, the number of generators increases by $|R_+^{(k)}||R_-^{(k)}| - |R_+^{(k)}|$ at each iteration, which may lead to a much to large number of generators. We can improve this with some geometry, by noticing that most new generators we created are unnecessary, and only the "boundary" ones matter.
So instead of considering all pair of $R_+^{(k)}$ and $R_-^{(k)}$, intuition suggests that looking at the adjacent ones is enough. In fact, intuition suggests that these pairs form extreme rays of the cone of the next iteration, and that together with the extreme rays in $R_-^{(k)}$ and $R_0^{(k)}$ we have all the extreme rays of the new cone.
Indeed, if $r_i \in R_+^{(k)}$ and $r_j \in R_-^{(k)}$ are adjacent extreme rays of the cone generated by $R^{(k)}$, then they are in a common 2-dimensional face of the cone, which they generate. The intersection point from the construction of the previous paragraphs will be in that face as a conic combination, and also in the new halfspace $A_{i*}x \leqslant 0$ considered for iteration k to $k+1$. Now $A_{i*}x = 0$ can't be parallel (contain) to the face of dimension 2 the adjacent extreme rays are in, as these rays take non-zero value for $A_{i*}r$. Thus, the

intersection of the face of dimension 2 and $A_{i*}x = 0$ is a one dimensional space of the cone, containing r_{ij}, which is therefore an extreme ray.
We will now show that all extreme rays of $R^{(k+1)}$ have been considered in this process. If r is such an extreme ray, then either it belonged to $R_-^{(k)}$ or $R_0^{(k)}$, or it didn't, in which case it must be on $A_{i*}x = 0$. So $i \in eq(r)$ and $rank(A_{eq(r)*}) = d-1$, and therefore $rank(A_{eq(r)\setminus i*}) = d-2$: this means that r was on a 2-dimensional face of the cone of iteration k, generated by $R^{(k)}$. This face had 2 extreme rays generating it, each on a different of the sides of $A_{i*}x = 0$, as otherwise r couldn't be on it, extreme rays that would have been adjacent. So r is a multiple of some r_{ij}, and hence all extreme rays are considered.

So if we start the DD-method with $\left(A_{K_1*}, R^{(1)}\right)$ where $R^{(1)}$ is made of the extreme rays of the cone precisely, then our the described procedure will maintain the property that the $R^{(k)}$ are made precisely of the extreme rays of the corresponding cone, in particular the last iteration, for the actual cone of interest.
Two problems have to be addressed: first, how do we get the initial $\left(A_{K_1*}, R^{(1)}\right)$, and second, how do we test for adjacency ?
The first problem can be handled by choosing A_{K_1*} so as to have full rank d (find it's echelon form). Note that the rank must be d as we assume the cone to be pointed. Then we can use the equivalence $\lambda = A_{K_1*}(-x) \geqslant 0 \Leftrightarrow x = A_{K_1*}^{-1}\lambda, \lambda \geqslant 0$ to see that the cone $A_{K_1*}x \leqslant 0$ is generated by the columns of $A_{K_1*}^{-1}$, which form extreme rays as $A_{K_1*}A_{K_1*}^{-1} = 1$ shows that the eq set of the columns of $A_{K_1*}^{-1}$ are of size $d-1$.
The second can be handled with brute force: we can test if $A_{(eq(r_i)\cap eq(r_j))*}x = 0$ has dimension 2 for all pairs $r_i \in R_+^{(k)}$ and $r_j \in R_-^{(k)}$.

Polyhedron intersection and separation:
We ask how to tell if to polyhedran in H-description intersect, and how to find a separating hyperplane given by the convex separation theorems if they're disjoint.
We'll find two methods for the first question, and show that the second one can't help us answering the second on efficiently. For the first problem, we're dealing with a feasibility LP by solving $Ax \leqslant b$ and $Cx \leqslant d$ simultaneously.
The separation problem can be solved if one finds a pair of points minimising the Euclidean distance for the polyhedra. We can find the minimum distance for the maximum norm as follows. For polyhedra $Ax \leqslant b$ and $Cx \leqslant d$, we can solve $\min \delta$ st. $\begin{cases} Ax \leqslant b \\ Cy \leqslant d \\ x_i - y_i \leqslant \delta \\ y_i - x_i \leqslant \delta \end{cases}$. This problem is feasible precisely when the polyhedra are non empty (then for a feasible point on can set $\delta = \|x-y\|_\infty$) and it's always bounded, as $|x_i - y_i| \leqslant \delta \Leftrightarrow \begin{cases} x_i - y_i \leqslant \delta \\ y_i - x_i \leqslant \delta \end{cases}$ can thus $0 \leqslant \delta$. At minimum, δ is the distance of the closets pair x and y of the two polyhedra, for $\|.\|_\infty$: this is because $|x_i - y_i| \leqslant \delta$ for all i, therefore $\|x-y\|_\infty \leqslant \delta$ and we'd contradict minimality if $\|x-y\|_\infty < \delta$, as we can then decrease δ without violating other constraints. Thus we're actually minimising $\|x-y\|_\infty$ over pairs of points in the two polyhedra.

The optimum is $\delta = 0$ precisely if the polyhedra intersect, for if they do, $0 \leqslant \min \delta \leqslant \|x-x\|_\infty = 0$ as x is a point of both polyhedra (set $y = x$), and if $\delta = 0$, then for the associated x and y of the polyhedra, $0 \leqslant \|x-y\|_\infty \leqslant \delta = 0$ thus $x = y$ and that point is in both polyhedra, which intersect.

Unfortunately, the pair of points minimising the $\|.\|_\infty$ norm may not be that minimising the $\|.\|_2$, which has drastic consequences for separation. Consider the polyhedras $\{0\}$ and $(1,1) + \left\{z : \left(\cos(\theta), \sin(\theta)^t z \geqslant 0\right)\right\}$

for parameter $\theta \in]0, \pi/4]$. The minimum $\|.\|_\infty$ distance is always achieved by 0 and $(1,1)$ only, as for all other points of the second polyhedron (for $z \neq 0$), since one of $1+z_1$ and $1+z_2$ must be strictly greater then 1, for of $z_1\cos(\theta)$ or $z_2\sin(\theta)$ must be strictly positive. Yet, as θ approaches 0, any hyperplane normal to $(1,1)$ won't separate the polyhedra, as the point $(1,1) + \left(1, -\frac{\cos(\theta)}{\sin(\theta)}\right)$ will be on the origins side. SO one can not hope to mimic the separation proof for the Euclidean norm.

ADD: separation of polyhedra by homogenizing, separating cone generators (watch out for case of shared generators in the recession cone, ex.: two parallel rays have each two generators, one being shared), then intersect with z=1 to get separation of polyhedra.

Separating a point from a V-polyhedron:
We can use V-homogenization and cone-Farkas to solve the problem constructively, since cone-Farkas is. Indeed, for a point x in tha polyhedron, $\begin{pmatrix} x \\ 1 \end{pmatrix}$ can be written as a conic combination of $\begin{pmatrix} V & R \\ 1^t & 0 \end{pmatrix}$. If it isn't in the polyhedron, cone-Farkas provides h such that $h^t \begin{pmatrix} V & R \\ 1^t & 0 \end{pmatrix} \geqslant 0$ and $h^t \begin{pmatrix} x \\ 1 \end{pmatrix} < 0$. This rewrites to $h_{[d]}^t V \geqslant -h_{d+1}$, $h_{[d]}^t R \geqslant 0$ and $h_{[d]}^t x < -h_{d+1}$ which describe a hyperplane directed by $h_{[d]}$ that separates the point from the polyhedra, as any convec and conic combination y of V and R will satisfy $h_{[d]}^t y \geqslant -h_{d+1}$, due to $h_{[d]}^t V \geqslant -h_{d+1}$, $h_{[d]}^t R \geqslant 0$ and positivity of combinations.

Pivoting algorithm for V-representations:
For a polyhedron P of form $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$, we can use the techniques of the simplex method to find all vertices and rays in a DFS manner. We start with a phase 1 simplex algorithm to determine a vertex of P, or more precisely a feasible basis of it. We then perform DFS on the graph of the polyhedron. At each step, we consider a feasible basis in the frontier at perform all possible pivots for it (whereas in the simplex algorithm, we only pivoted on variables with negative reduced cost, which doesn't exist in this context): if the pivot leads to a ray, we create a neighbour node indicating that we've encountered a ray, that we consider explored by default; if the pivot leads to a basis we do nothing ; if the pivot leads to a new basis, we add that basis to the frontier. When all possible $(n-m)$ pivots have been performed, we mark the basis as explored. We stop when the frontier is empty.
We remark that degeneracy is no problem here: pivot rules indicate that there must be at least one pivot leading to a different vertex, if one exists. For if v is a vertex and w a different one, objective $c = w - v$ is strictly larger for w then it is for v, so that a pivot rule on the simplex method with this objective would imply that we eventually find a pivot that leaves for a new vertex.

We now explore the idea of putting a polyhedron $Ax \leqslant b$ in standard form and seeking a relation between the vertices of the standard form an those of $Ax \leqslant b$. Once we've found the vertices and rays of $\begin{cases} Ax^+ - Ax^- + s = b \\ x^+, x^-, s \geqslant 0 \end{cases}$, note that we can get the point of $Ax \leqslant b$ by applying the linear map $(x^+, x^-, s) \mapsto x^+ - x^-$. Since convex and conic combinations are preserved under linear maps), we see that the images of the vertices and rays of the standard form generate a V-representation of $Ax \leqslant b$. But not all images need to be vertices or rays themselves: consider projecting an Egyptian pyramid on its base, so that the top vertex of the pyramid becomes an interior point, but the 4 base vertices are the vertices of the projection. We therefore have to check which of them truly are vertices or rays, which we can do with the H-characterisations of vertices and rays. For example for images of vertices, we check if they're vertices if the tight inequalities on them form linear independent rows of A.

Integer hulls:

13.3 Triangulations

13.4 Protein folding on lattices

HP lattice model and Newmans Algo (LatticeProtein...)

13.5 Distance geometry, graph embedding, rigidity

"Euclidean dist geo" chap 3 and 4
"Combinatorial rigidity" and other books and papers

13.6 Knots

"NP–HARD PROBLEMS NATURALLY ARISING IN KNOT THEORY" DALE KOENIG AND ANASTASIIA TSVIETKOVA

13.7 Solutions

14 Probabilistic methods and derandomization

14.1 Local search with random walks

14.2 Second moment method

We consider the following problem: a class of m students is visiting a city, and each student s has a set F_s of attractions/sites, which is a subset of n possible attractions/sites X, that they want to see. The goal is to split the attractions/sites in two disjoint groups A and B that are subsets of X, such that for each student, at least one of $F_s \cap A$ or $F_s \cap B$ is empty, and we can therefore split the students into two groups, according to what they want to see. We call (A, B) a separator (which exist, for there always is $(\emptyset, X)$ that counts) and say that its size is $\min(|A|, |B|)$, an our goal is have a large and similar amount of attractions to see for both groups, in the sense that we seek the separator of largest size ($(\emptyset, X)$ has size 0).

We investigate the following strategy: we split the students into two groups independently and uniformly at random and take A to be the attractions/sites for which the students desiring to visit them are all in the first group, and B to be the attractions/sites for which the students desiring to visit them are all in the second group. We assume that all attractions/sites are desired by at least one student (otherwise, there's not point considering it). They will form a separator, as they're disjoint, as each student is in one group only, and attractions that are desired by no students, that hence conventionally satisfy the definitions of A and B, don't exist. We also introduce the parameter r that bounds the number of attractions/sites students want to visit in the sense $|F_s| \leqslant r$. What can we say about the size of the separator ?

For an attraction x, we denote by Z_x the random variable indicating $x \in A$, so that if d_x students desire x, then $P(Z_x = 1) = \dfrac{1}{2^{d_x}}$, the probability that all of them are in the first group of students, so that $x \in A$. The size of A is the random variable $Z = \displaystyle\sum_{x \in X} Z_x$. If Y is the size of B, then our goal is to determine $P(\min(|A|, |B|) \geqslant k) = P(Z \geqslant k \cap Y \geqslant k) = 1 - P((Z < k \cup Y < k) \geqslant 1 - (P(Z < k) + P(Y < k))$, so our goal is to upper bound $P(Z < k)$, which works well with the Chebyshev inequality, to get a lower bound on the probability that the size of the separator is at least k.

We can state the Chebyshev inequality as $P(Z < (1-\delta)E(Z)) < \dfrac{Var(Z)}{\delta^2 E(Z)^2}$, where we can bound $E(Z) = \displaystyle\sum_{x \in X} E(Z_x) = \sum_{x \in X} \frac{1}{2^{d_x}} \geqslant n \frac{1}{2^{\frac{1}{n}\sum_{x \in X} d_x}} = \frac{n}{2^d}$ with the arithmetic-geometric mean, letting the average number of students desiring $d = \dfrac{1}{n}\displaystyle\sum_{x \in X} d_x$ appear, so that $P\left(Z < (1-\delta)\frac{n}{2^d}\right) < \dfrac{Var(Z)}{\delta^2 E(Z)^2}$ and $P\left(Z < (1-\delta)\frac{n}{2^d}\right) < \dfrac{Var(Z)2^d}{\delta^2 E(Z) n}$ (we keep one expectation for future purposes).

To bound the variance, we distribute $Var(Z) = \displaystyle\sum_{x \in X} Var(Z_x) + \sum_{x \neq y \in X} Cov(Z_x, Z_y)$. First, we have $Var(Z_x) = E\left(Z_x^2\right) - E(Z_x)^2 = E(Z_x) - E(Z_x)^2 \leqslant E(Z_x)$ as this is an indicator variable (0-1-valued and positive), so $\displaystyle\sum_{x \in X} Var(Z_x) \leqslant E(Z)$. Next, we note that if for x and y, no students exist that want to visit both, then Z_x, Z_y are independent, and then $Cov(Z_x, Z_y) = 0$. The pairs of attractions for which this is not the case can be bounded by $(r-1)\displaystyle\sum_{x \in X} d_x$, as for each x, there are at most $(r-1)$ other attractions each student wanting to visit x also wants to visit. With $Cov(Z_x, Z_y) = E(Z_x Z_y) - E(Z_x)E(Z_y) \leqslant E(Z_x Z_y) \leqslant E(Z_x) = \dfrac{1}{2^{d_x}}$ (they are indicator variables), we get $\displaystyle\sum_{x \neq y \in X} Cov(Z_x, Z_y) \leqslant (r-1)\sum_{x \in X} d_x \frac{1}{2^{d_x}}$.

To let $E(Z)$ appear in the previous bound we use the inequality $n\displaystyle\sum_{i \in [n]} a_i b_i \leqslant \left(\sum_{i \in [n]} a_i\right)\left(\sum_{i \in [n]} b_i\right)$ for sequences on positive numbers where a is non-increasing and b non-decreasing. This inequality can be shown

with $0 \geqslant \sum_{i>j\in[n]} (a_i - a_j)(b_i - b_j) = \sum_{i>j\in[n]} a_i b_i + a_j b_j - a_i b_j - a_j b_i = (n-1)\sum_{i\in[n]} a_i b_i - \sum_{i\neq j\in[n]} a_i b_j = n\sum_{i\in[n]} a_i b_i - \sum_{i,j\in[n]} a_i b_j$. It then follows that $\sum_{x\neq y\in X} Cov(Z_x, Z_y) \leqslant \frac{(r-1)}{n}\left(\sum_{x\in X} d_x\right)\left(\sum_{x\in X} \frac{1}{2^{d_x}}\right) = (r-1)dE(Z)$.

Finally, we get $Var(Z) \leqslant ((r-1)d+1)E(Z) \leqslant rdE(Z)$ by combining bounds and noting that $d \geqslant 1$ as all $d_x \geqslant 1$ by assumption. Hence $P\left(Z < (1-\delta)\frac{n}{2^d}\right) \leqslant \frac{Var(Z)2^d}{\delta^2 E(Z)n} \leqslant \frac{rd2^d}{\delta^2 n}$ and $P\left(\min(|A|, |B|) \geqslant (1-\delta)\frac{n}{2^d}\right) > 1 - 2\frac{rd2^d}{\delta^2 n}$, since we can do the same for B, or more precisely Y.

For a large number of sites and a small average desirability, so that $\frac{rd2^{d+1}}{n} < \frac{1}{4}$ for example, then with $\delta = \sqrt{\frac{rd2^{d+1}}{n}} \in [0,1]$ for some $q \in]0,1[$, we get $P\left(\min(|A|, |B|) \geqslant \frac{n}{2^{d+1}} > 4rd\right) > 0$. So we can get a separator of size at least $\frac{n}{2^{d+1}} > 4rd$ with small but non-zero probability from this procedure.

14.3 Basics of derandomization

We mentioned an approximation algorithm for max-exact-q-SAT that for m clauses we can expect to find an assignment satisfying at least $m\left(1-\frac{1}{2^k}\right)$ clauses. The outputed assignment was set by choosing the truth assignment for each variable independently and at random.

Method of conditional probabilities:
We now investigate a technique that "derandomizes" certain randomised algorithms.
The idea of using conditional probabilities to derandomise the algorithm for q-SAT is that it's fairly easy to computed conditioned expectations for this problem, and that these conditioned expectations can serve as a search heuristic when branching on the truth assignments of variables.
If the variables are $x_1,..,x_n$ and we use a branching procedure that helped us set the values of $x_1,...,x_k$, then we can set x_{k+1} by comparing $E(C|x_1=s_1,...,x_k=s_k,x_{k+1}=true)$ and $E(C|x_1=s_1,...,x_k=s_k,x_1=false)$, where C is the number of satisfied clauses and the s_i are the set assignments of the $x_1,...,x_k$. We should choose the truth assignment for which the expectation is biggest to get as many satisfied clauses as possible.
The immediate question is how to compute expectations of form $E(C|x_1=s_1,...,x_k=s_k)$. If we've kept track of the clauses satisfied for $x_1=s_1,...,x_{k-1}=s_{k-1}$, then we can check the remaining clauses on whether they contain x_k or not. If they don't, the probability that they are satisfied, conditioned on $x_1=s_1,...,x_k=s_k$ is $\left(1-\frac{1}{2^k}\right)$, by independence. If they do then either $x_k=s_k$ satisfies the clause, or $x_1=s_1,...,x_k=s_k$ isn't enough to satisfy the clause, a number between $q-k$ and $q-1$ unsettled variables could still satisfy it in future iterations. In the first case, the clause contributes with 1 to the conditional expectation, in the second, we have to count the number u of unsettled variables in the clause, and contribute $\left(1-\frac{1}{2^u}\right)$ to the expectation, again by independence. So computing the conditional expectation requires at most $O(mn)$ counts (for m clauses).

At each step, one per variable, we compute two conditional expectations and compare them, keeping the assignment for which the expectation was maximum. So the algorithm runs in $O\left(mn^2\right)$. In the last step, we compute $E(C|x_1=s_1,...,x_n=true)$ and $E(C|x_1=s_1,...,x_n=false)$, which corresponds to the actual (not just expected) number of satisfied clauses for these assignments. The question is how high the number of satisfied clauses is at termination.
Here, we use the following to formalise our intuition: if $P(C_i|x_1=s_1,...,x_{k-1}=s_{k-1})$ is the probability of satisfying clause i knowing $x_1=s_1,...,x_{k-1}=s_{k-1}$, then $P(C_i|x_1=s_1,...,x_{k-1}=s_{k-1})=P(C_i\cap(x_k=true)|x_1=s_1,...,x_{k-1}=s_{k-1})+P(C_i\cap(x_k=flase)|x_1=s_1,...,x_{k-1}=s_{k-1})$. Then, using the law $P(A\cap B|C)=P(A|B\cap C)P(B|C)$, and having $P(x_k=s_k|x_1=s_1,...,x_{k-1}=s_{k-1})=\frac{1}{2}$ by definition (independence), we get $P(C_i|x_1=s_1,...,x_{k-1}=s_{k-1})=\frac{P(C_i|x_1=s_1,...,x_k=true)+P(C_i|x_1=s_1,...,x_k=false)}{2}$.
Therefore, we also have $E(C|x_1=s_1,...,x_{k-1}=s_{k-1})=\frac{E(C|x_1=s_1,...,x_k=true)+E(C|x_1=s_1,...,x_k=false)}{2}$ by linearity.
In particular $E(C|x_1=s_1,...,x_{k-1}=s_{k-1})\leqslant\max(E(C|x_1=s_1,...,x_k=true),E(C|x_1=s_1,...,x_k=false))$. This means that the conditional expectations for the assignment we successively construct are increasing at each step ! Since $E(C)\geqslant m\left(1-\frac{1}{2^k}\right)$ initially, we will have $E(C|x_1=s_1,...,x_n=s_n)\geqslant m\left(1-\frac{1}{2^k}\right)$ at termination, so that this forms a $\left(1-\frac{1}{2^k}\right)$ approximation algorithm.

ADD: derandomisation of maximum edge bipartite subgraph via conditional proba derandomisation as

exercise

14.4 The local lemma and its applications

When attempting to find an approximation algorithm for exact-k-SAT by choosing the value of each variable uniformly at random we're faced with the following probabilistic problem. For concreteness, assume we have m clauses on n variables. We can easily compute the probability that clause i isn't satisfied, an event denoted by A_i, as $P(A_i) = \frac{1}{2^k}$, since for the clause to be false, all its k (exactly) literals have to be, which happens with probability $\frac{1}{2^k}$, due to independence.

The problem is that of computing $P\left(\overline{A_1} \cap ... \cap \overline{A_m}\right)$, the probability that all clauses are satisfied, since we have a priori no information on the dependence of the A_i. If we can establish that $P\left(\overline{A_1} \cap ... \cap \overline{A_m}\right) > 0$, we've shown that the instance is satisfiable by the probabilistic method.
This is where the local lemma comes into play:

Erdös-Lovasz-Spencer local lemma:

For events A_i, we consider their dependecy graph G in which A_i and A_j are adjacent if they're dependent, in the sense that theyre not adjacent if they're in a maximum set of mutually independent events.
If $4 \max_i(\deg_G(A_i)) \max_i(P(A_i)) \leqslant 1$, then $P\left(\overline{A_1} \cap ... \cap \overline{A_m}\right) > 0$.
Alternatively, if one can find $x_i \in [0,1)$ so that $P(A_i) \leqslant x_i \prod_{j \in \delta_G(A_i)} (1 - x_j)$,
then $P\left(\overline{A_1} \cap ... \cap \overline{A_m}\right) > \prod_{j \in [m]} (1 - x_j) > 0$.

If we have $4 \max_i(\deg_G(A_i)) \max_i(P(A_i)) \leqslant 1$ and the dependency graph has no isolated vertices, then we can find the x_i from the second part explicitly. To get this, we write $p = \max_i(P(A_i))$ and $d = \min_i(\deg_G(A_i))$. Note that $4pd \leqslant 1$ as $d \leqslant \max_i(\deg_G(A_i))$. Setting all $x_j = \frac{1}{d+1} \in [0,1)$, we get $x_i \prod_{j \in \delta_G(A_i)} (1 - x_j) \geqslant \frac{1}{d+1}\left(1 - \frac{1}{d+1}\right)^d$. We can bound this further with the exponential to $\frac{1}{d+1}\left(1 - \frac{1}{d+1}\right)^d \geqslant \frac{1}{e(d+1)}$. Now, with $4dp \leqslant 1$, we get $ep(d+1) \leqslant \frac{e}{4} + ep \leqslant (e+1)p \leqslant 4p \leqslant 4pd \leqslant 1$ so that $\frac{1}{e(d+1)} \geqslant p = \max_i(P(A_i))$.

For exact-k-SAT, this yields the following result:

Exact-k-SAT:

If for each clause there is another that shares at least one variable, and any two clauses share at most 2^{k-2} variables, then there is a satisfying assignment for the instance.

Indeed, the first condition, implies that $d \geqslant 1$ and the second one that $\max_i(\deg_G(A_i)) \leqslant 2^{k-2}$. Since $\max_i(P(A_i)) = 2^{-k}$, we get $4 \max_i(\deg_G(A_i)) \max_i(P(A_i)) = 1 \leqslant 1$ and the local lemma applies, so that we can conclude with the probabilistic method.

Proof of the local lemma:

Venkatesh's treatment is better then ours.
We prove this by induction on m, by proving $P\left(A_1|\overline{A_2},...,\overline{A_m}\right) \leqslant 2p$ and $P\left(A_1|\overline{A_2},...,\overline{A_m}\right) \leqslant x_1$ by induction (we prove both results in parallel).
The last step is obtained by repeatedly applying the law $P(C|A,B)P(B|A)P(A) = P(A \cap B \cap C)$ to get $P\left(\overline{A_1} \cap ... \cap \overline{A_m}\right) = \prod_{j=1}^{m} P\left(\overline{A_i}|\overline{A_1},...,\overline{A_{i-1}}\right)$. With the previous bounds, we then get $P\left(\overline{A_1} \cap ... \cap \overline{A_m}\right) \geqslant (1-2p)^m > 0$ in the first case, as $p \leqslant \frac{1}{4}$ when $4dp \leqslant 1$, and $P\left(\overline{A_1} \cap ... \cap \overline{A_m}\right) \geqslant \prod_{j\in[m]} (1-x_j)$ in the second case.

We now turn to the induction. For $m = 1$, these are the assumptions together with $1 \leqslant 2$ and $x_j \in [0,1)$.
For the step, we start by ordering the events (re-indexing if necessary) so that A_2 to A_k are the ones dependent on A_1. By repeatedly applying the law $P(A|B,C) = \frac{P(A \cap B|C)}{P(B|C)}$, we get $P\left(A_1|\overline{A_2},...,\overline{A_m}\right) = \frac{P\left(A_1 \cap ... \cap \overline{A_k}|\overline{A_{k+1}},...,\overline{A_m}\right)}{P\left(\overline{A_2} \cap ... \cap \overline{A_k}|\overline{A_{k+1}},...,\overline{A_m}\right)}$. We can then bound the top by sub-events $P\left(A_1 \cap ... \cap \overline{A_k}|\overline{A_{k+1}},...,\overline{A_m}\right) \leqslant P\left(A_1|\overline{A_{k+1}},...,\overline{A_m}\right)$. Then with mutual independence $P\left(A_1|\overline{A_{k+1}},...,\overline{A_m}\right) \leqslant P(A_i)$, so that the top is bounded by p or $x_i \prod_{j\in\delta_G(A_i)} (1-x_j)$.
To bound the bottom, we use a opposites and a union bound $P\left(\overline{A_2} \cap ... \cap \overline{A_k}|\overline{A_{k+1}},...,\overline{A_m}\right) = 1 - P\left(A_2 \cup ... \cap A_k|\overline{A_{k+1}},...,\overline{A_m}\right) \geqslant 1 - \sum_{j=2}^{k} P\left(A_j|\overline{A_2},...,\overline{A_m}\right)$. Here we use the induction hypothesis on families of events of size $< m$, to get bound $1 - \sum_{j=2}^{k} P\left(A_j|\overline{A_2},...,\overline{A_m}\right) \geqslant 1 - 2p(k-1)$ and since $k - 1 \leqslant \max_i(\deg_G(A_i))$, we get $1 - 2p(k-1) \geqslant \frac{1}{2}$, so that in the end $P\left(A_1|\overline{A_2},...,\overline{A_m}\right) \leqslant 2p$ as desired.
For the parallel conditions, we repeatedly use $P(A|B,C)P(B|C) = P(A \cap B|C)$ to rewrite the bottom as $P\left(\overline{A_2} \cap ... \cap \overline{A_k}|\overline{A_{k+1}},...,\overline{A_m}\right) = \prod_{j=2}^{k} P\left(\overline{A_j}|\overline{A_{k+1}},...,\overline{A_m}\right)$. Here the induction hypothesis and $x_j < 1$ tells us that $\prod_{j=2}^{k} P\left(\overline{A_j}|\overline{A_{k+1}},...,\overline{A_m}\right) \geqslant \prod_{j\in\delta_G(A_i)} (1-x_j)$, so that we retrieve the desired bound $P\left(A_1|\overline{A_2},...,\overline{A_m}\right) \leqslant x_1$.

ADD: exercise disjoint cycles from chapter in extremal combi book ?

14.5 The Moser-Tardos algorithm

In this section, we study how to use the local lemma to develop algorithms that actually find the structures that the local lemma asserts exist.

The Moser-Tardos algorithm works in a setting where the events A_i depend on random variables X_i, which are chosen independently at random. For k-SAT, the X_i where the random truth assignments for variable i and the A_j are the events that clause j is unsatisfied. The algorithm works as follows. Start with a set of random values for the X_i and check if one of the A_i occurs. If it doesn't, we have $\bigcap_{i\in[m]} \overline{A_i}$ as desired for this assignment, and we stop. Otherwise, we select an A_i that occurs and consider the set of variables $vbl(A_i)$ that determine A_i (for k-SAT, the variables in the unsatisfied clause). We re-sample these variables independently according to their distribution and check if we have $\bigcap_{i\in[m]} \overline{A_i}$. We do these re-sampling steps until $\bigcap_{i\in[m]} \overline{A_i}$ occurs.

The algorithm may therefore not terminate. However, we can show that under the assumptions of the local lemma, the algorithm has a finite expected runtime dependent on the x values from the local lemma.

Moser-Tardos:

For events A_i, we consider their dependency graph G in which A_i and A_j are adjacent if they're dependent, in the sense that theyre not adjacent if they're in a maximum set of mutually independent events.
If one can find $x_i \in [0,1)$ so that $P(A_i) \leqslant x_i \prod_{j\in\delta_G(A_i)} (1-x_j)$, then the Moser-Tardos algorithm has an expected runtime of at most $\sum_{i\in[m]} \frac{x_i}{1-x_i}$.

The question is how often one expects to see event A_i be selected due to occurring for the current sample, for all i. The algorithm corresponds to a random process in which we can keep track of the A_i that are selected. More precisely, an instance of the algorithm corresponds to a random, possibly infinite sequence $S_1, S_2, ...$of A_i that are selected. We're then interested in the (possibly infinite) number N_i of times A occurs in the sequence.
PROBLEM: probabilistic justification that $E(N_i)$ is finite ???

We will build a structure called a witness tree that is associated to the sequence $S_1, S_2, ...$ and to A_i. To get bounds, combine the local lemma assumptions to a study these trees as originating from a Galton-Watson process.
Galton-Watson:
We construct a (possibly infinite) labeled tree rooted at A_i as follows. Start by adding the root r and label it $l(r) = A_i$. In each iteration, go over the vertices v added during the previous iteration and do the following: for all $A_j \in \overline{\delta}(l(v))$, create a child u with label $l(u) = A_j$ with independent probability x_j. Stop only if no children were added in the previous iteration.
We now seek the probability for a finite tree T rooted with label A_i to occur as outcome of this process. For each vertex v in the tree, the probability it was included is $x_{l(v)}$, for vertices not in the tree to not have been included, the probability is $1 - x_{l(u)}$ for all parents v such that $l(u) \notin \overline{\delta}(l(v))$.

By independence, we get $\frac{1}{x_i}\left(\prod_{v\in T} x_{l(v)}\right)\left(\prod_{v\in T}\prod_{l(u)\notin\overline{\delta}(l(v))}(1-x_{l(u)})\right)$ where the first term cancels out with the next one so as to get the right probability, due to the root being labeled deterministicly. Since all vertices but the root are the children of exactly one parent, we can rewrite the probability as $\frac{1-x_i}{x_i}\left(\prod_{v\in T}\frac{x_{l(v)}}{1-x_{l(v)}}\right)\left(\prod_{v\in T}\prod_{\overline{\delta}(l(v))}(1-x_{l(u)})\right)$. We can recognize the local lemma assumption's expression in that probability.

14.6 Solutions

15 Expander graphs

15.1 Definitions and key properties

As you can tell from the titles of the previous chapters, expander graphs have many applications. Expander graphs arise in the context of spectral graph theory, in which we use the eigenvalues of the adjacency matrix of a graph to deduce some of its properties.

Recall that the adjacency matrix of a graph is a matrix with entries $a_{uv} = \chi_{\{u,v\}\in E}$. It's symmetric, and has therefore real eigenvalues and a orthonormal basis of eigenvectors, by the spectral theorem. Spectral graph theory tries to relate these eigenvalues to the graph.
An example is for example the following property:

Largest eigenvalue of d-regular graphs:

The largest absolute value of eignevalues of the adjacency matric of a d-regular graph is d.

Proof: If A is the adjacency matrix of the graph, then $A \times 1_V$ is the vector with coordinates $\deg(v)$. For d-rgular graphs, we get the nice case that $A \times 1_V = d \cdot 1_V$, so that d is a eigenvalue. To see maximality, take a eigenvector w with values λ, so that $Aw = \lambda w$, and look at the norm $\|.\|_\infty$. Let's call u the coordinate in which the max-norm is attained: we have $\left|\sum_{v\in\Gamma(u)} w_v\right| = |\lambda w_u|$, so that we get bound $|\lambda|\|w\|_\infty = |\lambda||w_u| \leqslant \sum_{v\in\Gamma(u)} |w_v| \leqslant d\|w\|_\infty$ by d-regularity.

A more important property is:

The expander mixing lemma:

For a d-rgular graph, and notation $e(S,T) = |\{\{u,v\} \in E : u \in S, t \in T\}|$ for the number of edges between $S \subseteq V$ and $T \subseteq V$ (note that we allow S and T to overlap), and λ for the second largest absolute value of eignevalues of the adjacency matrix of the graph, we have:

$$\left|e(S,T) - \frac{d}{|V|}|S||T|\right| \leqslant \lambda d\sqrt{|S||T|}$$

Proof: We can write $e(S,T) = \sum_{\{u,v\}\in E: u\in S, t\in T} 1 = 1_S^t A 1_T$ for indicator vectors $1_S, 1_T$ of the vertex sets. Now, what is this product in the spectral basis ? We may take a basis containing eigenvector $w_1 = \frac{1}{\|1_V\|_2}1_V$, as eigenspaces are orthogonal (recall that for eigenvalues λ, μ and eigenvectors x_λ, x_μ, we can write $x_\lambda^t A x_\mu$ as $\lambda x_\lambda^t x_\mu$ and $\mu x_\lambda^t x_\mu$, so that when the eigenvalues are different, $x_\lambda^t x_\mu = 0$). By denoting the other eigenvectors with w_i, for eigenvalues λ_i in decreasing order of indices, we have $e(S,T) = d\frac{\left(1_S^t 1_V\right)\left(1_T^t 1_V\right)}{\|1_V\|_2^2} + \sum_{i=2}^{|V|} \lambda_i \left(1_S^t w_i\right)\left(1_T^t w_i\right)$, recalling that the $\left(1_T^t w_i\right)$ are the coordinates of 1_T in that basis. Now, $\left(1_S^t 1_V\right)\left(1_T^t 1_V\right) = |S||T|$ and $\|1_V\|_2^2 = |V|$.

By rewriting $e(S,T) - \frac{d}{|V|}|S||T| = \sum_{i=2}^{|V|} \lambda_i \left(1_S^t w_i\right)\left(1_T^t w_i\right)$, we can bound and rewrite as follows $\left|e(S,T) - \frac{d}{|V|}|S||T|\right| \leqslant \sum_{i=2}^{|V|} |\lambda_i| \left|\left(1_S^t w_i\right)\left(1_T^t w_i\right)\right| \leqslant \lambda \sum_{i=1}^{|V|} \left|\left(1_S^t w_i\right)\left(1_T^t w_i\right)\right|$. By interpreting the latter factor as $\sum_{i=1}^{|V|} \left|\left(1_S^t w_i\right)\left(1_T^t w_i\right)\right| = (W1_S)^t(W1_T)$, using Cauchy-Schwartz provides final upper-bound $\|W1_S\|_2\|W1_T\|_2 = \|1_S\|_2\|1_T\|_2 = \sqrt{|S||T|}$, where we used orthonomrality of W.

Here are a fun corollary of the expander mixing lemma as exercise:
Exercise 1: For a d-regular graph G with second largest absolute value of eigenvalues λ, show that if $\lambda > 0$, then $\chi(G) \geqslant \frac{d}{\lambda}$.

15.2 Derandomising algorithms with expanders

We'll describe a general method for the following problem. Suppose we have a binary decision problem (for example: is this graph Hamiltonian ?) and a probabilitsitc algorithm A that for an instance x of the problem will output a decision in $\{0,1\}$, based on m random coin-tosses $r \in \{0,1\}^m$ it performs for its execution, such that the probability of taking the wrong decision can be bounded by $P(A(x,r)\ is\ false) \leqslant p_A$.

An approach to improve the decision is to repeat this algorithm $2t+1$ times, an then give as final output the majoritary output among the $2t+1$ runs. For this to yield a false output, there have to be at least $t+1$ false outputs among $2t+1$ the runs. Using the bounds on the upper tail of the binomial distribution (Hoeffding's inequality), we can estimate that error probability with $e^{-2(2t+1)\left(1-p_A-\frac{t}{2t+1}\right)^2}$ (assuming $\dfrac{t}{2t+1} \geqslant p_A$).

An alternative appraoch is he following idea. Choose only one r at random, then select m more r' based on this r, according to some "diversification criterion". Then, run the algorithms $A(x,r')$ and take the majority output.
Here, onely one random string had to be generated, while we hope that our "diversification criterion" provides sufficiently different samples to get a good estimate on how A answers on average.
The way we'll choose the r' is through expander graphs: we'll choose a vertex r at random and take all $r' \in \Gamma(r)$, using the intuition that in expander graphs, all vertices are globally close to each other, so that $\Gamma(r)$ is a diversified sample.

So, we consider a d-regular graph on 2^m vertices (representing the $r \in \{0,1\}^m$) which has seconf larest absolute value of eigenvalues λ. With our procedure, the probability of ending up with a false output can be bounded as follows:

Mixing lemma bound:

The probability of a false output is at most $\dfrac{1}{2^m}\left(\dfrac{\lambda}{d}\right)^2\left(\dfrac{2}{1-2p_A}\right)^2$, assuming $p_A < \dfrac{1}{2}$.

Proof: Fix an instance x. For it, we have region of failure $T = \{r \in \{0,1\}^m : A(x,r)\ is\ false\}$, so that when we take an r at random, we have $P(A(x,r)\ is\ false) = \dfrac{|T|}{2^m} \leqslant p_A$. We can do the same for the algorithm B we just explained, calling its region of failure S. Our goal is then to upper-bound $|S|$, as we have $P(B(x,r)\ is\ false) = \dfrac{|S|}{2^m}$.
When B fails at r, it was because the majority of the $A(x,r')$ failed for $r' \in \Gamma(r)$. Since we're in a d-regular graph, that's at least $\dfrac{d}{2}$ failures. So for each vertex in S, it must be connected to at least $\dfrac{d}{2}$ vertices of T, so that $e(S,T) \geqslant \dfrac{d|S|}{2}$. We can then apply the mixing lemma to get bound $e(S,T) - \dfrac{d|S||T|}{2^m} \leqslant \lambda\sqrt{|S||T|}$.
We'll lower-bound the left with $\dfrac{d|S|}{2} - d|S|p_A$ and the right with $\lambda\sqrt{|S|p_A}$, so that rearranging yields $|S| \leqslant \left(\dfrac{\lambda}{d}\right)^2\left(\dfrac{2}{1-2p_A}\right)^2$ when $p_A < \dfrac{1}{2}$.

We see that the ratio $\frac{\lambda}{d}$ makes an appearance, and that we which to build a graph for which this is small. Ramanujan graphs fit this description, for example. We'll see them in the section on constructions.

COMPLETE: with 23.3 of extremal combi book

15.3 Error correcting codes with expanders

Recall the context of error correcting codes.
We have a binary word $w \in \{0,1\}^k$ of length k which we encode into $f(w) \in \{0,1\}^n$, where $f(w)$ can be computed in polynomial time, and is injective. The image of f is a code $C \subseteq \{0,1\}^n$. We want to be able to recover w from $f(w)+\mu$ where $\mu \in \{0,1\}^n$ is noise (think of addition as in $\mathbb{Z}_2$). This is where the distance of the code comes into play. The Hamming distance $\Delta(x,y)$ of binary words x, y is the number of letters in which they differ. We want $\Delta(f(x), f(y))$ to be large, so that we can distinguish $f(x), f(y)$ for relatively small noise. The distance of the code is $d(C) = \min_{X,Y \in C} \Delta(X,Y)$ and serves as a lower bound on the distance of codewords. If $\Delta(0,\mu) < \frac{d(C)}{2}$, then $\Delta(f(x), f(x)+\mu) < \frac{d(C)}{2}$, so that $\Delta(f(y), f(x)+\mu) \geqslant \frac{d(C)}{2}$ for all other y, since otherwise $\Delta(f(x), f(y)) \leqslant \Delta(f(x), f(x)+\mu) + \Delta(f(x)+\mu, f(y)) < d(C)$, contradicting minimality. So for less then $\frac{d(C)}{2}$ changes to the code transmitted, one can still recover the correct codeword, by looking for the closest codeword t the received message. We also want the tasks of finding the closest codeword and it's encoding word to be efficient.

Expander codes are a method for using a linear code on small code words repeatedly, so that the distance of the expander code increases more then linearly with the number of code applications.
Recall that a code is linear when $f(x)+f(y)$ is a codeword for any x, y, in the sense $\forall x, y \exists z, f(x)+f(y) = f(z)$.
COMPLETE: examples of linear codes.

An expander code is built from a (d, n, λ)-expander graph as follows. We create 2 copies of the vertices of the graph, the bipartition sets L and R. We add an edge between a vertex of L and one of R if the vertices the corresponded to in the expander where adjacent.
FIX: its still d-regular, but there is no explanation why λ is the same...

15.4 Pseudo-random generators with expanders

chapter 7,8 of ExpanderGraphApplications

15.5 Constructing expanders

Ramanujan graphs:
Ramanujan graphs originate in the slightly esoteric study of representations of algebraic structures at graphs. In a finite field $\mathbb{F}_p$ for a prime p, given two elements $a \neq 0$ and b, we can write each element as a product with a and as a sum with b, so that there are c and d such that $ac = b + d$. Let's represent this as a graph where we represent pairs (a, b) with $a \neq 0$ as vertices and we connect them if $ac = b+d$.

Such graphs are of interest to us since we can build and study them systematically. For example, we can show that such a graph is $(p-1)$-regular: for a fixed (a, b) with $a \neq 0$, there are $(p-1)$ non-zero elements that we can represent simulaneously as ac and $b + d$, which correspinds to a neighbour (c, d), each different as the element $ac = b + d$ is different.
We will now derive such properties of the graph by mixing algebraic and graph-theoretic notions and arguments.

Lemma:

For $p > 5$, if $a = c$ or $b = d$, the corresponding vertices have no common neighbours, and otherwise, they have exactly 1 neighbour.

Proof: To see the first case, we use contradiction. If $a = c$ and the vertices had common neighbour (e, f), we'd have $b + f = ae = ce = d + f$ so that $b = d$, and a similar argument holds when $b = d$, as $a, c \neq 0$ are invertible. To see the second case, solve $\begin{cases} ae = b + f \\ ce = d + f \end{cases}$ be subtracting to get $(a-c)e = (b-d)$. Since $(a-c)$ and $(b-d)$ are non-zero in this case, we get $e = (a-c)^{-1}(b-d) \neq 0$. The solution, and hence the neighbour is unique as $f = ae - b = ce - d$ is uniquely determined. Note that in the solution must have existed in the first place, as otherwise, in all cases two different vertices have no common neighbours, so that edges form a matching, so that $|E| \leqslant 2|V|$. However, regularity implies $2|E| = |V|(p-1)$, so that $p \leqslant 5$, a case we purposefully exclude.

We will now derive a bound on λ:

Bound:

For the type of Ramanujan graphs we just built, $\lambda \leqslant \sqrt{3p}$.

Proof: This bound is obtained with some manipulations. If A is the adjacency of the graph, then recall that A^2 has its entries counting the length 2 u-v-walks. By the previous lemma, its off-diagonal entries are 1 if $a \neq c$ and $b \neq d$, and 0 if $a = c$ or $b = d$. Its diagonal entries are $p-1$, by regularity (walk forth and back to the neighbours). Another fact we noted is that 1_V is an eigenvector, and that we can build a base of eigenvectors all orthogonal to 1_V. To estimate the other eigenvalues μ, we may therefore consider them having eigenvectors x_μ orthogonal to 1_V. We can then write $A^2 = (p-2)I + J - E$, where J is the all-1 matrix, and E is the error matrix defined by this decomposition. The point is that we have, by multiplying with x_μ, the identity $\mu^2 x_\mu = (p-2)x_\mu - Ex_\mu$, since $Jx_\mu = 0$ as $1_V^t x_\mu = 0$. This says that $\left(p - 2 - \mu^2\right)$ is an eigenvalue of E. Taking a closer look at the 0/1-matrix E, we realize that its the incidence matrix of a graph with same vertex set as the inicial one, but with edges precisely if $a = c$ or $b = d$. This graph is $(2p-3)$-regular, as we have p neighbours of an (a, b) for which $a = c$, $p-1$ neighbours of an (a, b) for

which $b = d$, and we counted the case $a = c$ and $b = d$ in both cases, which are the same vertex. We can therefore use the fact that its anbsolute eigenvalues are bounded by $(2p - 3)$ to get $|p - 2 - \mu^2| \leqslant 2p - 3$ so that $\mu^2 + 2 - p \leqslant 2p - 3$ which imples $\mu \leqslant \sqrt{3p}$.

We can upperbound $\frac{\lambda}{d} \leqslant \frac{\sqrt{3p}}{(p-1)} = O\left(\frac{1}{\sqrt{p}}\right)$, which yields a decent bound in the derandomization from the previous sections. However, note that the construction required $n = p(p-1)$, so that we don't exactly meet the framework of that context.

COMPLETE: chapter 5 of ExpanderGraphApplications and chaper 15.2 on Ramanujan graphs from extremal combi book

15.6 Solutions

Exercise 1:
By applying the mixing lemma with $S = T$ and where S is an independent set, so that $e(S, T) = 0$, we get $|S| \leqslant \frac{\lambda}{d}|V|$. Since a coloring can be though of as a partition of vertices into independent sets, by spliting $|V|$ into the sum of the partition sets and using the bound, we get $|V| \leqslant \chi(G)\frac{\lambda}{d}|V|$ from which the result follows.

16 Basics in convex optimization

16.1 Convex sets, functions and optimization problems

Convex programs
Semidefinite programming
Max volume ellipsoids, in particular for geometric ILP algorithm by Lenstra (ref. lectnotes-GIP Paat Weismantel pdf-page 30).

Form differentiable optimization, we know first and second order conditions to compute local optima. We're interested in finding a large class of functions for which the local optima are also global for two reasons. The first is that for such functions, we expect local search algorithms to lead to a global optimum, and the second is that we may need to use such algorithms, in case the equations from the first order condition are difficult to solve. Additionally, we hope that this class of functions need not be differentiable, while keeping most of the properties of their differentiable counterparts.

We'll work on convex function domains, as these allow us to easily use arguments from 1D calculus.

Definition:

A set $C \subset \mathbb{R}^d$ is **convex** if it contains all segments between endpoints in C: for all $x, y \in C$ and $s \in [0,1]$, we have $sx + (1-s)y \in C$.

What goes "wrong" with differentiable functions that have local non-global optima ?
For example, consider $f : C \subset \mathbb{R}^d \to \mathbb{R}$, where C is convex (and open) and f is twice continuously differentiable, so that x and y are local minima but $f(x) > f(y)$. We know that for both points, the gradient is zero and the hessian is positive semidefinite. Yet, if we observe f on the segment $[x, y]$, $f(x) > f(y)$ implies that the derivative must have been non-zero along that segment (finite difference theorem), so it must have increased and decreased along it, as it's zero at both endpoints. A strong condition that prohibits this is when restrict the class of function studied to those who's hessian is positive semidefinte on all of C. Then the second derivative along the segment can't change sign, so that the first derivative is monotone: if it's non-zero, one of the endpoints will violate the first order condition, and of it stays at zero along the segment, we can't have $f(x) > f(y)$.

We'll now try to get rid of the differentiable conditions. We'll consider the Taylor expansion with rest $f(y) = f(x) + \nabla f(x)^t(y-x) + (y-x)^t \nabla^2 f(z)(y-x)$ with $z \in [x, y]$. The condition that $\nabla^2 f(z)$ is positive semidefinite on all of C provides $f(y) \geqslant f(x) + \nabla f(x)^t(y-x)$. Note that this weaker condition also provides the result we seek: if x satisfies the first order conditions, and the previous inequality holds on all of C, then x is a global minimum.
In fact, the two conditions are equivalent as the contrapositive of the converse holds: if there was a point $x \in C$ and a direction d so that $d^t \nabla^2 f(x) d < 0$, then for $\varepsilon > 0$ so that $d^t \nabla^2 f(y) d < 0$ for all $y \in B(x, \varepsilon)$ and $y = x + \varepsilon d$, the Taylor expansion with rest would result in $f(y) < f(x) + \nabla f(x)^t(y-x)$.

In the final step, using condition $f(y) \geqslant f(x) + \nabla f(x)^t(y-x)$ twice on an point $sx + (1-s)y$ of the segment $[x, y]$, getting $f(x) \geqslant f(sx + (1-s)y) + \nabla f(sx + (1-s)y)^t(x - sx - (1-s)y)$ and $f(y) \geqslant f(sx+(1-s)y) + \nabla f(sx+(1-s)y)^t(y - sx - (1-s)y)$, we obtain $sf(x) + (1-s)f(y) \geqslant f(sx+(1-s)y)$ as consequence. This last condition is the one that will define the class of functions that we want to work with, as it doesn't require differentiable, but has the desired properties when the function is additionally

differentiable. Indeed, if a function satisfies $sf(x)+(1-s)f(y) \geqslant f(sx+(1-s)y)$ and is differentiable, then $f(y) \geqslant \dfrac{f(sx+(1-s)y)-f(x)}{(1-s)} + f(x)$ and taking the limit $s \to 1$ provides $f(y) \geqslant f(x) + \nabla f(x)^t(y-x)$.
If a function satisfies $sf(x) + (1-s)f(y) \geqslant f(sx + (1-s)y)$ for all $x, y \in C$ and $s \in [0,1]$ and we consider two points with $f(x) > f(y)$, then x can't be a local minimum, as in particular for $s \to 1$, $f(x) = \max(f(x), f(y)) > sf(x) + (1-s)f(y) \geqslant f(sx + (1-s)y)$.

As a conclusion:

Definition:

We consider a convex $C \subset \mathbb{R}^d$.
We call a function $f : X \to \mathbb{R}$ **convex** if it satisfies $sf(x) + (1-s)f(y) \geqslant f(sx + (1-s)y)$ for all $x, y \in C$ and $s \in [0,1]$. If f is continuously differentiable, then this is equivalent to $f(y) \geqslant f(x) + \nabla f(x)^t(y-x)$ on all of C, and if f is twice continuously differentiable, then this is equivalent to $\nabla^2 f(z)$ being positive semidefinite on all of C.
For such functions, critical points can't be saddle points and local minima are global ones.

We now define the standard convex optimization problem:

Definition:

Consider convex functions $f_0, f_1, ..., f_k : \mathbb{R}^d \to \mathbb{R}$, matrix $A \in \mathbb{R}^{m \times d}$ and vector $b \in \mathbb{R}^d$.
The standard **convex optimization problem** is $\min(f_0(x))$ st. $\begin{cases} f_i(x) \leqslant 0, \forall i \in [k] \\ Ax = b \end{cases}$.

In such a problem, we minimise a convex function over a convex space. Indeed is x and y are feasible solutions, then for $s \in [0,1]$, $A(sx + (1-s)y) = sAx + (1-s)Ay = sb + (1-s)b = b$ and $f_i(sx + (1-s)y) \leqslant sf(x) + (1-s)f(y) \leqslant 0$, so that $sx + (1-s)y$ is a feasible solution as well. For such problems, local minima are global ones.

Example:
Consider the problem of finding a pair of points in polyhedra $Ax \leqslant b$ and $By \leqslant d$ that are closest to each other, in the sense that they minimise $\|x - y\|_2$. The norm function is convex by the triangular inequality and $Ax - b$ is convex by linearity.

16.2 Lagrange relaxation and duality

In constrained differentiable optimization, we proceed differently as in the unconstrained case. If we seek the lowest point on the unit circle, aka. a solution to $\min y$ st. $x^2 + y^2 = 1$, the first order condition fails. However, by parameterise the circle as $(\cos(t), \sin(t))$, we turn the problem in to an unconstrained one. In general, if $f, g : \mathbb{R}^d \to \mathbb{R}$ and we seek a solution to $\min f(x)$ st. $g(x) = 0$, and we can parameterise the set of feasible solutions near a feasible solution x via $p(y) : B_{\mathbb{R}^m}(0, r) \to \mathbb{R}^d$, so that $g(p(B_{\mathbb{R}^m}(0, r))) = 0$ and $p(0) = x$, then x is a local minimum if 0 is for $f \circ p$. Since the latter problem is unconstrained, we can use the first order condition (assuming a smooth parameterisation) to find $\nabla f(p(0))^t D_0 p = 0$. If we note that differentiating $g \circ p$ yields $\nabla g(p(0))^t D_0 p = 0$, we can draw an interesting conclusion when the feasible set is $(d-1)$-dimensional: in that case, $\nabla f(x)$ and $\nabla g(x)$ must be orthogonal to the hyperplane spanned by the columns of $D_0 p$, aka. they must be parallel. This is the case in our example, as the lowest point on the unit circle is one where the normal to the tangent to the circle points down.

The latter condition states that $\nabla f(x) = \mu \nabla g(x)$ for some scalar μ, and is a first order condition for constrained differentiable problems. By notincing that it can be reformulated as $\nabla_x (f - \mu g)(x, \mu) = 0$, where ∇_x is differentiation wrt. the variable x of $(x, \mu) \mapsto f(x) - \mu g(x)$, and that $\nabla_\mu (f - \mu g)(x, \mu) = 0 \Leftrightarrow g(x) = 0$, this first order condition for constrained problems of form $\min f(x)$ st. $g(x) = 0$ becomes the first order condition for the unconstrained problem $\min\limits_{x,\mu}(f(x) - \mu g(x))$.

We can do some loose generalisations of these arguments. For a problem of form $\min f(x)$ st. $g(x) \leqslant 0$, point verifying $g(x) < 0$ are interior points to the feasible set, and we can use the regular first order condition. Note that in that case $\nabla f(x) = \mu \nabla g(x)$ still holds, as we can set $\mu = 0$. Further more, in this particular case, we can say more about the scalar μ. Indeed, recall that the gradient points towards the steepest ascending direction, as $f(y) = f(x) + \nabla f(x)^t (y - x) + O\left(\|y - x\|^2\right)$ and by choosing a direction δ and considering $\dfrac{f(x + t\delta) - f(x)}{t} = \nabla f(x)^t \delta + O(t)$ we se that $f(x + t\delta) - f(x)$ has the sign of $\nabla f(x)^t \delta$ for $t > 0$ small enough.
Now if $\mu > 0$ in $\nabla f(x) = \mu \nabla g(x)$, $\nabla f(x)$ would point in the same direction of $\nabla g(x)$. Then for small $t > 0$, the point $y = x - t\nabla f(x) = x - t\mu \nabla g(x)$ would verify $g(y) < g(x)$, so that it would be feasible, and $f(y) < f(x)$, so that x can't be a local minimum. Therefore, if x is a local minimum for the problem $\min f(x)$ st. $g(x) \leqslant 0$,, we can summarize the above cases in the necessary condition that there exist a $\lambda \geqslant 0$ so that $\nabla f(x) = \lambda \nabla g(x)$.

Again, this reformulates to the regular first order condition on $\min\limits_{x,\lambda}(f(x) - \lambda g(x))$, but st. $\lambda \geqslant 0$ this time. This leads us to the study of:

Definition:

For an optimization problem of form $\min(f(x))$ st. $\begin{cases} g_i(x) \leqslant 0, \forall i \in [r] \\ h_j(x) = 0, \forall j \in [s] \end{cases}$, we consider the **Lagrangian** $L(x, \lambda, \mu) = f(x) + \lambda^t g(x) + \mu^t h(x)$ with domain $\mathbb{R}^d \times \mathbb{R}^r_{\geqslant 0} \times \mathbb{R}^s$.

As for convexity in the previous section, we motivated the Lagrangian in a differentiable setting, but we'll study it for non-differentiable cases as well.

Since the first order conditions of a minimization problem and its Lagrangian are linked (in the differentiable setting), we'll investigate optimizing the Lagrangian and its relation to the main problem.
First note that for a feasible solution to the main problem x, we have $L(x, \lambda, \mu) \leqslant f_0(x)$ as $h_j(x) = 0$,

$f_i(x) \leqslant 0$ and $\lambda \geqslant 0$. So if p^* is the minimum value of the main problem, we have $\inf_x(L(x,\lambda,\mu)) \leqslant p^*$. It turns out that for some problems, $g(\lambda,\mu) = \inf_x(L(x,\lambda,\mu))$ has an explicit analytic form. Hence our interest in:

Definition:

The **Lagrange dual function** associated to a general optimization problem is $g(\lambda,\mu) = \inf_x(L(x,\lambda,\mu))$.

Since $g(\lambda,\mu) \leqslant p^*$ for all μ and $\lambda \geqslant 0$, we may seek the best lower bound this gives to the main problem by maximising $g(\lambda,\mu)$ st. $\lambda \geqslant 0$. An interesting remark can then be made: $g(\lambda,\mu)$ is concave ($-g(\lambda,\mu)$ is convex), regardless of the nature of the f_i and h_i, so that we're solving a convex minimisation problem, but one for which g may not be differentiable, or may even only evaluated by solving an unconstrained optimisation problem $\inf_x L(x,\lambda,\mu)$ as routine.
Indeed, note that $L(x, t\lambda+(1-t)\lambda', t\mu+(1-t)\mu') = tL(x,\lambda,\mu)+(1-t)L(x,\lambda',\mu')$ (split $f_0 = tf_0+(1-t)f_0$), so $L(x, t\lambda+(1-t)\lambda', t\mu+(1-t)\mu') \geqslant tg(\lambda,\mu)+(1-t)g(\lambda',\mu')$ and finally $g(t\lambda+(1-t)\lambda', t\mu+(1-t)\mu') \geqslant tg(\lambda,\mu) + (1-t)g(\lambda',\mu')$.

One can try to describe when strong duality holds:

Slater's condition:

For the standard convex optimization problem, if A has full rank and there exists a feasible interior point x of the solution space, in the sense that $f_i(x) < 0$ for $i = 1, ..., n$, then $\max_{\lambda\geqslant 0,\mu} g(\lambda,\mu) = p^*$.

Proof: The proof is based on a convex separation theorem that we won't prove. We could focus on a sort of graph of the solutions, where we relax the equality constraints $Ax = b$. This would be $G = \{(f_1(x), .., f_n(x), Ax-b, f_0(x)) : f_i(x) \leqslant 0, i = 1, ..., n\}$, which isn't convex, but who's epigaph $E = \{e \geqslant g : g \in G\}$ is convex. Indeed, for e_1 and e_2 in E, $te_1 + (1-t)e_2 \geqslant t(f_1(x_1), .., f_n(x_1), Ax_1 - b, f_0(x_1)) + (1-t)(f_1(x_2), .., f_n(x_2), Ax_2 - b, f_0(x_2))$, and with convexity of the involved functions $te_1 + (1-t)e_2 \geqslant (f_1(tx_1+(1-t)x_2), .., f_n(f_1(tx_1+(1-t)x_2)), A(f_1(tx_1+(1-t)x_2)) - b, f_0(f_1(tx_1+(1-t)x_2)))$. Since $f_i(tx_1 + (1-t)x_2) \leqslant 0$ by convexity for $i = 1, ..., n$, this means $te_1 + (1-t)e_2 \in E$.
This epigraph will never intersect the open ray $0 \times 0\times]-\infty, p^*[$, since this would mean that there is an x which is feasible and has lower objective value then the minimum. The convex separation theorem we mentioned now provides a hyperplane directed by $(\lambda,\mu,v) \neq 0$ such that $vt \leqslant \alpha$ for $(0,0,t)$ in the ray $0 \times 0\times]-\infty, p^*[$, and $\sum \lambda_i f_i(x) + \mu^t(Ax-b) + vf_0(x) \geqslant \alpha$ for x such that $f_i(x) \leqslant 0, i = 1, ..., n$, since the graph belongs to the epigraph and is also separated. Note that the last expression resembles that of the Lagrangian, which is where the idea of using hyperplane separation may have come from.
By choosing a sequence of points in the ray converging to $(0,0,p^*)$, we see that in the limit $vp^* \leqslant \alpha$. This leads to an upper bound on p^* by an expression that looks like a Lagrangian, which is just what we want, so we'll see that assumptions from Slater's condition are only there to make this proof finish well.

We note that $\lambda, \mu, v \geqslant 0$: otherwise, since we have an interior point x, we seen that all positive scaled standard basis vectors of the space of E are in E, so that if some corresponding hyperplane coefficient was negative, we could contradict $\geqslant \alpha$ by evaluating at large standard basis vectors.
Now, if $v > 0$, then $\sum \lambda_i f_i(x) + \mu^t(Ax-b) + vf_0(x) \geqslant \alpha \geqslant vp^*$ becomes $L\left(x, \frac{\lambda}{v}, \frac{\mu}{v}\right) \geqslant p^*$ and we get $\max_{\lambda\geqslant 0,\mu} g(\lambda,\mu) \geqslant p^*$ and therefore $\max_{\lambda\geqslant 0,\mu} g(\lambda,\mu) = p^*$, as desired.

Otherwise $v = 0$ and we'll see how the assumptions of Slater's conditions make this case impossible. By evaluating at the feasible interior point x, we then have $\sum \lambda_i f_i(x) \geqslant \alpha \geqslant 0$ and the fact that $f_i(x) < 0$ for $i = 1, ..., n$ and $\lambda \geqslant 0$ implies $\lambda = 0$. Since $(\lambda, \mu, v) \neq 0$ as it directs a hyperplane, we're left with $\mu \neq 0$. For all x such that $f_i(x) \leqslant 0, i = 1, ..., n$, the expression of the halfspace has now been reduced to $\mu^t(Ax - b) \geqslant 0$. Since for the interior point $\mu^t(Ax - b) = 0$ and we can find some r, such that if $\|x - y\| \leqslant r$, then $f_i(y) < 0$ (continuity), there is such a y for which $\mu^t(Ay - b) < 0$, unless $\mu^t A = 0$, as $\mu^t(Ax - b) = 0$ isn't a full dimensional space. But both $\mu^t A = 0$ and $\mu^t(Ax - b) < 0$ are excluded, the latter by the hyperplane condition $\mu^t(Ax - b) \geqslant 0$ and the former by assuming that A has full rank.

We now look at relations between the primal and the dual solution:

Complementary slackness:

If $g(\lambda, \mu) = p^* = f_0(x)$ for feasible x and (feasible :=) $\lambda \geqslant 0, \mu$, then for all $i = 1, ..., n$, at least one of $\lambda_i = 0$ or $f_i(x) = 0$ holds, aka. $\lambda_i f_i(x) = 0$.

Proof: We have $f_0(x) = g(\lambda, \mu) \leqslant L(x, \lambda, \mu) \leqslant f_0(x)$, so that $L(x, \lambda, \mu) = f_0(x)$ and therefore $\sum \lambda_i f_i(x) = 0$ since x is feasible so $Ax = b$. Since $f_i(x) \leqslant 0$ and $\lambda \geqslant 0$ for $i = 1, ..., n$, this negative sum is only zero when all terms are, so $\lambda_i f_i(x) = 0$.

16.3 Karush-Kuhn-Tucker conditions

Karush-Kuhn-Tucker conditions:

If a standard convex minimization problem with differentiable f_i has a finite local minimum and strong duality holds (for example if it satisfies the Slater's conditions), then there is a solution to the following system, known as **KKT conditions**:

$$\begin{cases} f_i(x) \leqslant 0, i = 1, ..., n \\ Ax = b \\ \lambda \geqslant 0 \\ \lambda_i f_i(x) = 0, i = 1, ..., n \\ \sum \lambda_i \nabla f_i(x) + A^t \mu + \nabla f_0(x) = 0 \end{cases}$$

Conversely, if this system has a solution, then the corresponding x is a minimum.

The reason convex optimization is an active area of research is that this system may be hard to solve, so that optimization algorithms have to be developed.

Proof: The first 3 conditions represent feasibility, the 4th is complementary slackness, and the last is the first order condition of the unconstrained problem $\inf_x L(x, \lambda, \mu)$. If strong duality holds, the solutions attaining the optima solve the system. Conversely, we note that for a solution to the KKT system, $L(x, \lambda, \mu) = f_0(x)$ (feasibility and complementary slackness) and x is a critical point of the concave L, which has a single local and hence global optimal value, so that $L(x, \lambda, \mu) = g(\lambda, \mu)$. Now since $p^* \geqslant g(\lambda, \mu) = f_0(x) \geqslant p^*$, x attains the minimum.

We now give a more direct and geometric derivation of the KKT conditions.
We consider the general problem $\min f_0(x)$ st. $f_i(x) \leqslant 0, i = 1, ..., n$ for C^1 functions f_i.
If x is a local optimum, then we consider the active constraints $i \in A$ such that $f_i(x) = 0$. For any feasible y (sufficiently close to x), we have $f_i(y) \leqslant f_i(x)$ for $i \in \{0\} \cup A$.
We then get that

TO FINISH: Ruszczyński chap 3.4.

16.4 Solutions

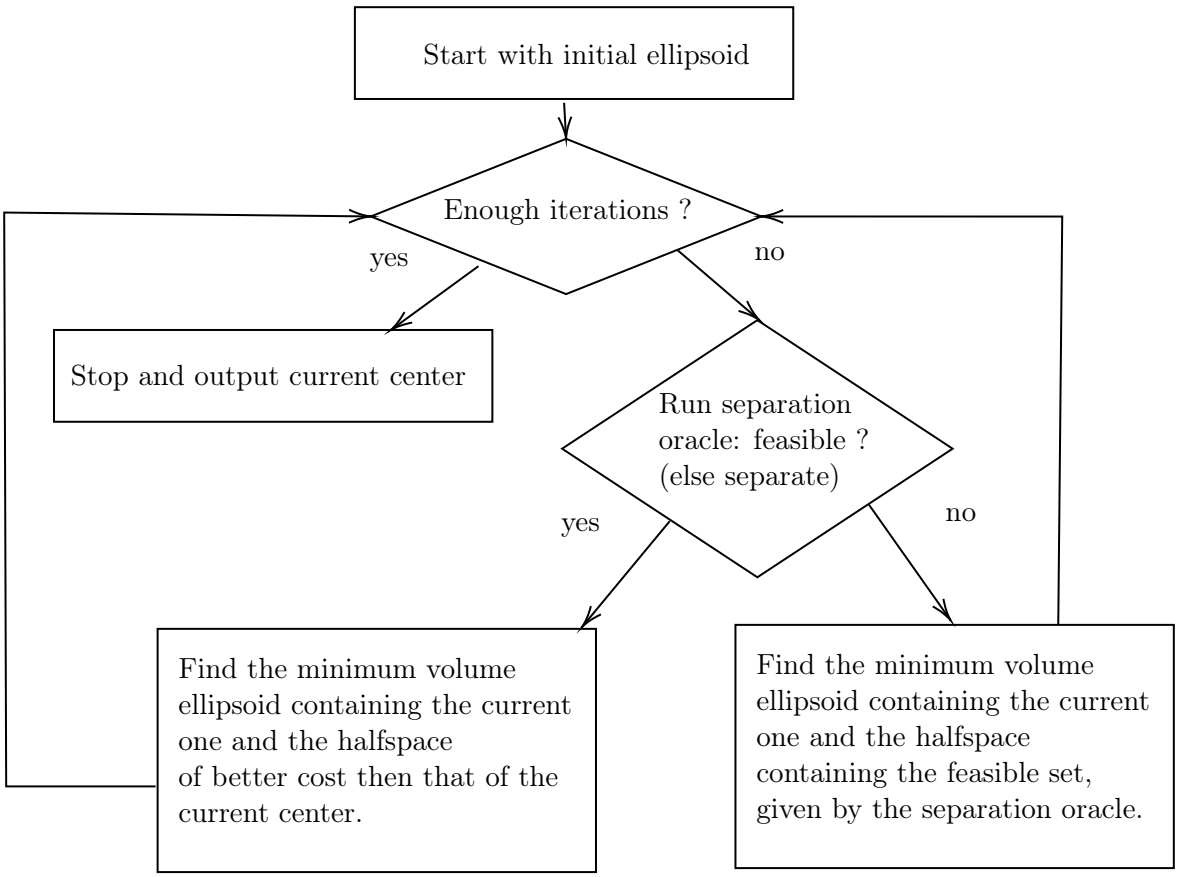
17 The ellipsoid method(s)

17.1 The ellipsoid method method

The ellipsoid method solves an optimization problem by a kind of binary search, with a separation oracle as subroutine. It also requires the knowledge of two balls B_i and B_c contained and containing the feasible set respectively. Starting from B_c, we check if the iteration steps ellipsoids center is feasible. If it isn't, we use the separation oracle to receive a halfspace (with the current ellipsoids center on its boundary) containing the feasible set, and if it is, we consider the halfspace of (linear) cost better then that of the current ellipsoids feasible center. In both cases, we proceed by building a new ellipsoid containing the intersection of the previous one with the halfspace at hand and of minimal volume for the latter property (a Löwner-John ellipsoid.). One can show that after finitely many iteration, the center of that steps ellipsoid will be feasible and (up to arbitrary additive error) optimal. The ellipsoid method requires taking square roots, so it's numerically inaccurate.

Algorithm:
Input: A convex set P of feasible solutions to the optimization problem, together with balls $B(x_0, r) \subseteq P \subseteq B(x_0, R)$ (in particular we need a feasible solution x_0 and the feasible sets have to be full-dimensional), and a separation oracle.
Task: Find a feasible $y \in P$ so that its cost $c \cdot y \geq \sup(c \cdot x : x \in P) - \varepsilon$ for a given ε.

Procedure:
Start with $y := x_0$ and $E_0 := B(x_0, R)$.
For $N(\varepsilon, r, R, d)$ iterations (over k), do:
$\rightarrow$ Run the separation oracle for y and P
$\rightarrow$ If $x_k \in P$, set $y := x_k$ and then define E_{k+1} to be the minimum volume ellipsoid containing the intersection of E_k and the halfspace $c \cdot x \geq c \cdot y$
$\rightarrow$ Else, let $a \cdot x = a \cdot x_k$ be the returned separating hyperplane passing through x_k, and define E_{k+1} to be the minimum volume ellipsoid containing the intersection of E_k and the halfspace $a \cdot x \leq a \cdot x_k$.



Before justifying the procedure, we want to point out that the input of the algorithm isn't an obviously attainable one. In the cases we'll present as applications, the input must be found by an analysis specific to the problem.
As a simpler example, we can consider an LP. Here the separation oracle checks if a candidate x is in P (given in H-description) by computing and checking $Ax \leq b$. If an inequality is violated, it's corresponding hyperplane serves as separating hyperplane. Finding the balls is less obvious. If we assume that the origin is feasible, we can look cubes $a[0,1]^d \subseteq P \subseteq b[0,1]^d$ and use balls $B(\frac{a}{2}\mathbf{1}, \frac{a}{2}) \subseteq P \subseteq B(\frac{a}{2}\mathbf{1}, b\sqrt{d})$. This is for example the case for capacitated transportation/matching problems.

We'll see that $vol(E_k) \leqslant e^{-\frac{k}{2d+2}} vol(B_d(0,1))$ for our construction of ellipsoids.
We now show a first loop invariant: each E_k contains $x \in P : c \cdot x \geq c \cdot y$. This is true by assumption for E_0 as it contains all of P. If at an iteration, $x_k \in P$, then E_{k+1} contains $x \in P : c \cdot x \geq c \cdot y$ as we updated $y := x_k$ and defined E_{k+1} that way. Otherwise, we note that by induction E_k contained $x \in P : c \cdot x \geq c \cdot y$, which is a feasible set and is therefore included in the halfspace $a \cdot x \leq a \cdot x_k$, so that it's contained in E_{k+1} in this case too.
A second invariant invariant is that y is always feasible. It is at the beginning ($y := x_0$) by assumption and is only ever replaced by feasible points.

If P is closed, it's also compact, due to being bounded by $B(x_0, R)$. So we may assume that the optimum wrt. $c \cdot x$ is attained in z. To estimate the distance $c \cdot z - c \cdot y$ ($\geqslant 0$ due to assuming a maximisation problem) of our solution candidate y at an iteration to the actual optimum, we'll use volumetric arguments. We'll find a set included in E_k whose volume depends on $c \cdot z - c \cdot y$, so that the volume bound on E_k will also turn into a bound on $c \cdot z - c \cdot y$.

This set will be a scaling of the ice-cream cone ("cornetto") $U = conv(z, (B(x_0, r) \cap \{x : c\cdot x = c\cdot x_0\}))$. Note that $U \subset P$ by convexity and the fact that $B(x_0, r) \subseteq P$ and U is full dimensional as $z \notin B(x_0, r) \cap \{x : c\cdot x = c\cdot x_0\}$ (otherwise, since $B(x_0, r) \subseteq P$, the solution $x + \frac{r}{\|c\|}c$ is feasible and has sup-maximal value).

We then consider $U' = U \cap \{x : c\cdot x \geqslant c\cdot y\}$. This is U, scaled with scaling center z and with scaling ratio $\frac{c\cdot z - c\cdot y}{c\cdot z - c\cdot x_0}$. Since $U \cap \{x : c\cdot x \geqslant c\cdot y\} \subseteq \{x \in P : c\cdot x \geqslant c\cdot y\}$ as $U \subset P$, we get $U' \subseteq E_k$ for all y smaller then its value in the kth iteration. In particular, this is true for the last iteration N and its corresponding y.

We have therefore $vol(U') \leqslant vol(E_N) \leqslant e^{-\frac{N}{2d+2}} vol(B_d(0,1))$. To compute $vol(U')$, we use the scaling construction to get $vol(U') = \left(\frac{c\cdot z - c\cdot y}{c\cdot z - c\cdot x_0}\right)^d vol(U)$. Next, $vol(U)$ depends entirely on the fixed $c\cdot z - c\cdot x_0$ which we can bound by $c\cdot(z - x_0) \leqslant \|c\|\|z - x_0\| \leqslant \|c\|2R$ (Cauchy-Schwartz, $P \subseteq B(x_0, R)$) to get rid of the purely theoretical z.

Finally, $c\cdot z - c\cdot y \leqslant \|c\|2Rvol(U)\sqrt[d]{e^{-\frac{N}{2d+2}} vol(B_d(0,1))}$, so that for N large enough (but still logarithmic), the right side becomes smaller the ε and therefore $c\cdot z - c\cdot y \leqslant \varepsilon \Rightarrow c\cdot z - \varepsilon \leqslant c\cdot y$, as desired.

We now explain how to construct the ellipsoids. We start with ellipsoid $E_k = T_k \times B_d(0,1) + c_k$ given by an inveritble linear transformation T_k of the unit ball and translated to its center c, and a separating hyperplane $c_k + v_k^\perp$ passing through c_k with normal v_k, so that $K \subset \left\{x \in E_k : v_k^t(x - c_k) \geqslant 0\right\}$. To construct E_{k+1}, we simplify the situation with a coordinate change. We'll build an ellipsoid E that contains $\left\{y \in B_d(0,1) : e_d^t y \geqslant 0\right\}$, so that for any for an orthonormal transformation M_k sending e_d to $T_k^{-1}(v)$, the ellipsoid $T_k \times M_k \times E + c$ contains $\left\{x \in E_k : v_k^t(x - c_k) \geqslant 0\right\}$ and has volume $|\det(T_k)||\det(M_k)|vol(E) = \frac{vol(E_k)}{vol(B_d(0,1))} \times 1 \times vol(E)$.
Indeed, $\left\{x \in E_k : v_k^t(x - c_k) \geqslant 0\right\} \subseteq T_k \times M_k \times \left\{y \in B_d(0,1) : e_d^t y \geqslant 0\right\} + c_k$ as for $x \in E_k$ with $v_k^t(x - c_k) \geqslant 0$, we have $x = c_k + z$ with $v_k^t z \geqslant 0$ and $z \in T_k \times B_d(0,1)$, so that $T_k^{-1}(v)\cdot T_k^{-1}(z) \geqslant 0$ (sdp) and finally $e_d\cdot M_k^{-1}T_k^{-1}(z) = M_k^{-1}T_k^{-1}(v)\cdot M_k^{-1}T_k^{-1}(z) \geqslant 0$ (orthonormal), so that $y = M_k^{-1}T_k^{-1}(z) \in B_d(0,1)$ and $e_d^t y \geqslant 0$.

To build E, we write it as $a\sum_{i=1}^{d-1} x_i^2 + b(x_d - t)^2 \leqslant 1$ and set the following conditions. First, $t > 0$. At $x_d = 0$, we want E to touch the unit circle in that hyperplane, which is part of $\left\{y \in B_d(0,1) : e_d^t y \geqslant 0\right\}$, so that $a\sum_{i=1}^{d-1} x_i^2 \leqslant 1 - bt^2 \Leftrightarrow \sum_{i=1}^{d-1} x_i^2 \leqslant 1$: this can be done by setting $a = 1 - bt^2$ and requiring $a = 1 - bt^2 > 0$. Next, we want the ellipsoid to contain e_d on its boundary, so that $b(1-t)^2 = 1$ so that we can fix a and b with a choice of $t < \frac{1}{2}$. Finally, to get a hold on the volume, we note that $a\sum_{i=1}^{d-1} x_i^2 + b(x_d - t)^2 \leqslant 1$ is the preimage of $B_d(-te_d, 1)$ under $diag\left(\frac{1}{\sqrt{a}}, \frac{1}{\sqrt{b}}\right)$, so that E has volume $\sqrt{a}^{d-1}.\sqrt{b}.vol(B_d(0,1))$. We can bound the volume to be

$$\sqrt{1 - \left(\frac{t}{1-t}\right)^2}^{d-1} \cdot \sqrt{\frac{1}{(1-t)^2}}.vol(B_d(0,1)) \leqslant e^{\frac{1}{2}\left(\frac{1}{(1-t)^2} - (d-1)\left(\frac{t}{1-t}\right)^2\right)} vol(B_d(0,1))$$

with $1 + x \leqslant e^x$.

It turns out that $t = \dfrac{1}{d+1}$ is a good choice: it satisfies the conditions and provides a volume bound of $e^{-\frac{1}{2d+2}} vol(B_d(0,1))$. Also, it has the property we tried to get intuitively: for $x \in \left\{y \in B_d(0,1) : e_d^t y \geqslant 0\right\}$ we have $\sum_{i=1}^{d} x_i^2 \leqslant 1$, so that $a\sum_{i=1}^{d-1} x_i^2 + (x_d - t)^2 \leqslant 1 + bt^2 + \left(bt^2 + b - 1\right) x_d^2 - 2btx_d$ and for $t = \dfrac{1}{d+1}$ the right side becomes $1 + \dfrac{2d+2}{d^2}\left(x_d^2 - x_d\right)$ which is less then 1 because $x_d \geqslant 0$.

Ex.ElliComp: If the current ellipsoids center is x and the matrix defining its quadratic form is A and the normal of the separating hyperplane is a, find the new center of the next ellipsoid in the ellipsoid method explicitly in terms of this data.

17.2 The affine scaling algorithm

In this section we discuss an algorithm that solves LPs and technically qualifies as an interior point method, but one that relies of ellipsoids.
The idea of the affine scaling algorithm is that, knowing that linear optimization on ellipsoids have an explicitly expressible (analytic ?) solution, one can successively inscribe ellipsoids in the polyhedron, by letting the centers of the ellipsoids by the successive optima, thereby moving towards an optimum.

So we start with a standard form polyhedron $\begin{cases} Ax = b \\ x \geqslant 0 \end{cases}$ where A has full row rank. To get a first ellipsoid, we need it's center y, which should be in the (relative) interior of the polyhedron (if we want the inscribed ellipsoid to have the dimension of the polyhedron). This is the case for $\begin{cases} Ay = b \\ y > 0 \end{cases}$, which contains the ball $y + \left(\ker A \cap B\left(0, \frac{\|y\|_\infty}{2}\right)\right)$.

What kind of ellipsoid should we use ? To ease things, we'll consider ellipsoids of the form $E \cap (Ax = b)$ where E is an ellipsoid of the ambient space with center y. We want the optimum of the ellipsoid to be attained in an interior point, so that we can proceed similarly in each iteration. This is achieved if $E \subset (x > 0)$, which is itself achieved if the axes of the ellipsoid are parallel to the grid, so that it's described by an expression of form $\sum_{i=1}^{n} s_i(x_i - y_i)^2 \leqslant 1$, and the endpoints of the axes are in the strict positive orthant $x > 0$. The latter condition is that for $z_i = \min(x_i : x \in E)$, $z_i > 0$, or equivalently $(z_i - y_i)^2 < y_i^2$ for all i. If we then choose an $s \in]0, 1[$, we see that $s_i = \frac{1}{s y_i^2}$ does the job.

So E has form $\sum_{i=1}^{n} \frac{(x_i - y_i)^2}{y_i^2} \leqslant s$.
We then check that if $x \in E$, we have $x > 0$: otherwise, if there was an x and an index i form which $x_i \leqslant 0$, then $x_i - y_i \leqslant (-y_i) < 0$ (as $y > 0$), so that $\frac{(x_i - y_i)^2}{y_i^2} \geqslant 1 > s$ and we can't have $\sum_{i=1}^{n} \frac{(x_i - y_i)^2}{y_i^2} \leqslant s$.

We will solve $\min(c \cdot x)$ st. $\begin{cases} Ax = b \\ x \geqslant 0 \\ \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{y_i^2} \leqslant s \end{cases}$ as a routine, since we can solve such problems explicitly. Note that the third constraint now makes the second redundant. The reason for which such a problem is easily solved, is that ellipsoids are linear transformation of balls, and linear optimization on balls is easy.
If we let $Y = diag(y_i)$, then E is described by $\left\|Y^{-1}(x - y)\right\| \leqslant s$. By changing variables to $d = Y^{-1}(x - y)$, $\tilde{A} = AY$ and $\tilde{c}^t = c^t Y$, the optimisation problem reformulates to $\min(\tilde{c} \cdot d)$ st. $\begin{cases} \tilde{A}d = 0 \\ d \in B(0, s) \end{cases}$.

The only component of the cost that is relevant to us is the one on the space $\tilde{A}d = 0$. So by completing the columns of $\tilde{A}^t$to a basis along it's orthogonal $\ker \tilde{A}$, we can decompose the cost $\tilde{c} = c_p + c_n$ where $c_p \in \ker \tilde{A}$ and $c_n \in Im\tilde{A}^t$. So we can find writ $c_n = \tilde{A}^t w$ for some vector w and $c_p = \tilde{c} - \tilde{A}^t w = Y\left(c - A^t w\right)$ as $Y^t = Y$. We can determine w by the requirement that $\tilde{A}c_p = 0 \Leftrightarrow AY^2\left(c - A^t w\right) = 0 \Leftrightarrow w = \left(AY^2A^t\right)^{-1} AY^2 c$ where $\left(AY^2A^t\right)$ is invertible as the kernel of YA^t is 0, for $YA^t z = 0 \Rightarrow A^t z = 0 \Leftrightarrow z = 0$, as Y is invertible (recall $y > 0$) and A has full row rank. If we have $\left(AY^2A^t\right)z = 0$, then $z^t\left(AY^2A^t\right)z = 0 \Rightarrow \left\|YA^t z\right\|^2 = 0$, so that $z = 0$, and $\ker\left(AY^2A^t\right) = 0$.

Intuitively $\min(\tilde{c}\cdot d)$ over $\begin{cases}\tilde{A}d=0\\ d\in B(0,s)\end{cases}$ is achieved in $-\frac{s}{\|c_p\|}c_p \in \ker\tilde{A}\cap d \in B(0,s)$. Formally, we split $\tilde{c}\cdot d = c_p\cdot d + c_n\cdot d$ and simplify $\tilde{c}\cdot d = c_p\cdot d$, as $d\in\ker\tilde{A}$ and $c_n\in Im\tilde{A}^t$. Cauchy-Schwartz provides $\tilde{c}\cdot d > -\|c_p\|.\|d\| \geqslant -\|c_p\|.s = c_p\cdot\left(-\frac{s}{\|c_p\|}c_p\right) = \tilde{c}\cdot\left(-\frac{s}{\|c_p\|}c_p\right)$.

TO CONTINUE.

17.3 Solutions

Ex.ElliComp:
The ellipsoid is of form $T(B_d(0,1)) + x$ and has equation $(y-x)^t A(y-x) \leqslant 1$ where A is symmetric positive definite. We relate the two representations by noting that for $y \in T(B_d(0,1)) + x$, we have $T^{-1}(y-x) \in B_d(0,1)$ so that $(y-x)^t \left(T^{-1}\right)^t T^{-1}(y-x)$ and we see that $\left(T^{-1}\right)^t T^{-1} = A$ is the positive definite matrix we want. Conversely, if we're given A, the by diagonalising it into $A = P^t D P$ where D is diagonal with positive entries, then by setting $T^{-1} = \sqrt{D}P$ does the job.
For the next ellipsoid in the ellipsoid method, we apply to the ellipsoid containing $B_d(0,1) \cap \{x \cdot e_d \geqslant 0\}$ a rotation bringing e_d to $\dfrac{T^{-1}(a)}{\|T^{-1}(a)\|}$, and then we apply T and the translation by x. $\left(0_{d-1}, \dfrac{1}{d+1}\right)$ will get mapped to the new ellipsoids center. It first gets mapped to $\dfrac{1}{d+1}\dfrac{T^{-1}(a)}{\|T^{-1}(a)\|}$ and then $\dfrac{1}{d+1}\dfrac{a}{\|T^{-1}(a)\|}$. Noticing that $\left\|T^{-1}(a)\right\| = \sqrt{a^t \left(T^{-1}\right)^t T^{-1} a} = \sqrt{a^t A a}$, we get the desired result.

18 Interior point methods

Following Boyd (Convex optimization) and Bertsekas (Convex optimization algorithms).

18.1 Gradient methods

The gradient method:
We develop the gradient method for an unconstrained convex minimisation problem with objective $f \in C^2(\mathbb{R}^n)$ with minimum value $p^* > -\infty$. The idea of gradient descent is based on the fact that the opposite gradient $-\nabla f(x)$ of f at x points in the steepest descending direction of f at x. We can therefore construct a sequence given by $x_{k+1} = x_k - \nabla f(x_k)$ to greedily minimise f. It turns out that we face an immediate flaw: $-\nabla f(x)$ points us in the steepest descending direction, but it doesn't inform us how far we should go in this direction. Consider for example a decent for function $x \mapsto x^2$ at point $x_0 = 1$. If we used update $x_1 = x_0 - 2x_0 = -1$, followed by update $x_2 = x_1 - 2x_1 = 1$, we see that this overestimation leads the procedure to oscillate, instead of converging to the minimum 0.

So we modify the sequence to $x_{k+1} = x_k - t_k \nabla f(x_k)$ for scalars $t_k > 0$ that we will choose to regulate the step size. Convexity of f alone isn't much help in proving the decrease of objective as it translates to $f(y) \geqslant f(x) + \nabla f(x)^t (y - x)$, so that $f(x_{k+1}) \geqslant f(x_k) - t_k \|\nabla f(x_k)\|^2$, which we can draw no good conclusion from. We are looking for $f(x_{k+1}) - f(x_k) \leqslant -t_k \|\nabla f(x_k)\|^2 \leqslant 0$, which we hope to achieve with a clever choice of t_k.

The sign of $f(x_{k+1}) - f(x_k)$, which we seek, can be studied with the 2nd order truncated Taylor development of f: $f(x_{k+1}) - f(x_k) = -t_k \|\nabla f(x_k)\|^2 + \frac{1}{2} t_k^2 \nabla f(x_k)^t \nabla^2 f(z_k) \nabla f(x_k)$ for some $z_k \in [x_k, x_{k+1}]$, depending on t_k. To avoid dealing with z_k's dependence on t_k, we can add an assumption to f: we assume the condition that ∇f is M-Lipschitz on $\mathbb{R}^d$, so that $(y-x)^t \nabla^2 f(z)(y-x) \leqslant M \|y - x\|^2$ for all x, y and z on the segment $[x, y]$ (by finite difference and Cauchy-Schwartz).
With this assumption, we can bound $-t_k \|\nabla f(x_k)\|^2 + t_k^2 \nabla f(x_k)^t \nabla^2 f(z_k) \nabla f(x_k) \leqslant \left(-t_k + \frac{M}{2} t_k^2 \right) \|\nabla f(x_k)\|^2$ and finally with $\min_{t_k} \left(-t_k + \frac{M}{2} t_k^2 \right) = -\frac{1}{2M} \leqslant 0$ achieved in $t_k = \frac{1}{M}$, we can set $t_k = \frac{1}{M}$ for all steps, to ensure that $f(x_{k+1}) - f(x_k) \leqslant 0$.

However, before investigating convergence to the minimum of this descent, we remark that the setting of $t_k = \frac{1}{M}$ requires us to know M explicitly. The problem is that finding M requires a specific analysis of f or the solving of another optimisation problem (find the maximum of $d^t \nabla^2 f(z) d$ for $d \in S^1$ and $z \in \mathbb{R}^d$).

By a first order Taylor approximation, we can see that for small $t_k > 0$, we have $f(x_{k+1}) \leqslant f(x_k) - t_k \|\nabla f(x_k)\|^2$, which ensures descent. Instead of finding such t_k analytically, we can try to find them algorithmically, in a procedure called **backtracking line search**.
Indeed, we can choose a $b \in]0, 1[$ and check for $f(x_{k+1}) \leqslant f(x_k) - b^q \|\nabla f(x_k)\|^2$ for increasing $q = 0, 1, ...$, where t_k is the first b^q so that $f(x_k - b^q \nabla f(x_k)) \leqslant f(x_k) - b^q \|\nabla f(x_k)\|^2$ holds. This will eventually happen for $b^q \leqslant \frac{1}{M}$, as when $t \leqslant \frac{1}{M}$, we have $\left(-t + \frac{M}{2} t^2 \right) \leqslant -\frac{t}{2}$, so that $f(x_{k+1}) - f(x_k) + b^q \|\nabla f(x_k)\|^2 \leqslant -\frac{b^q}{2} \|\nabla f(x_k)\|^2 \leqslant 0$.

The difference here is that M only ensure that this algorithm terminates, but we do not need to compute M to find the step size. M only informs us on the runtime of the procedure (at this stage), but it isn't required to be known to run the procedure. We can actually drop the assumption that ∇f is M-Lipschitz

on $\mathbb{R}^d$ and still prove termination of backtracking line search, but having a bound on t_k, which we would have by dropping the assumption, will prove useful in the convergence analysis.
In Boyd's book, it's argued that since the sequence (x_k) is in the compact the sublevel set $f(x) \leqslant f(x_0)$, ∇f is naturally Lipschitz on that compact set, so the assumption isn't needed. However, we needed M to prove that the sequence is decreasing and hence in the sublevel set, so this is a circular reasoning.

We could perform backtracking line search at the start of the descent to find a good constant step size. The advantage of recomputing the step size at each step with backtracking line search is that we can hope $f(x_k - b^q \nabla f(x_k)) \leqslant f(x_k) - b^q \|\nabla f(x_k)\|^2$ to be achieved for earlier q, depending on the current x_k, which would lead to greater steps.

We now turn to the question of whether the sequence (x_k) constructed in this fashion will converge to the global minimum of f. Note that the sequence become constant precisely when $\nabla f(x_k) = 0$, which is where the global minimum is achieved.

To ensure convergence, we'll have to add an assumption on f and b. First, our goal is to bound $f(x_k) - p^*$, which we can do with $f(x_{k+1}) - p^* \leqslant f(x_k) - p^* - t_k \|\nabla f(x_k)\|^2$. The problem is that this "arithmetic" bound leads nowhere, as we don't know the behavior of $\sum_k t_k \|\nabla f(x_k)\|^2$. A first step would be to note that $t_k = b^q \leqslant \frac{1}{M} \leqslant b^{q-1}$, so that $t_k \geqslant \frac{b}{M}$ and $-t_k \|\nabla f(x_k)\|^2 \leqslant -\frac{b}{M} \|\nabla f(x_k)\|^2$.

Next, we can notice in the Taylor expansion $f(y) - f(x) - \nabla f(x)^t (y - x) = (y - x)^t \nabla^2 f(z)(y - x)$, we can get a lower bound on the left by assuming that f is **strongly convex**, in the sense that $(y - x)^t \nabla^2 f(z)(y - x) \geqslant m \|y - x\|^2$ for some constant $m > 0$. This is the case if, for example, $\nabla^2 f(z)$ is positive definite on the compact sublevel set $f(z) \leqslant f(x_0)$.
In the newly obtained $f(y) \geqslant f(x) + \nabla f(x)^t (y - x) + m\|y - x\|^2$, we can get a further lower bound by minimising the quadratic right side for y. By convexity, it's minimum when its gradient wrt. y is zero, which means $\nabla f(x) + m(y - x) = 0 \Leftrightarrow y = x - \frac{1}{m} \nabla f(x)$. So we have $f(y) \geqslant f(x) - \frac{1}{2m} \|\nabla f(x)\|^2$. We can substitute y for for an antecedent of the achieved minimum of f to get $p^* \geqslant f(x) - \frac{1}{2m} \|\nabla f(x)\|^2 \Leftrightarrow \frac{1}{2m} \|\nabla f(x)\|^2 \geqslant f(x) - p^*$ for all x.

Finally, we can get a "geometric bound" by substitution of the previous bounds in $f(x_{k+1}) - p^* \leqslant f(x_k) - p^* - t_k \|\nabla f(x_k)\|^2$ to get $f(x_{k+1}) - p^* \leqslant \left(1 - \frac{2bm}{M}\right) (f(x_k) - p^*)$.
We have $M \geqslant m$ by their definition and if we add the condition $b < \frac{1}{2}$, then $c := \left(1 - \frac{2bm}{M}\right) < 1$ and the bound $f(x_k) - p^* \leqslant c^k \, (f(x_0) - p^*)$ proves convergence.

We can also combine $t_k \|\nabla f(x_k)\|^2 \leqslant f(x_k) - f(x_{k+1})$ with $t_k \geqslant \frac{b}{M}$ and $\frac{1}{2m} \|\nabla f(x_k)\|^2 \geqslant f(x_k) - p^*$ to get $f(x_k) - p^* \leqslant \frac{M}{2bm} (f(x_k) - f(x_{k+1}))$, which gives us a concrete criterion for stopping. We can iterate until $(f(x_k) - f(x_{k+1})) \leqslant \varepsilon$, a condition we cn easily check, and know that $f(x_k) - p^* \leqslant \frac{M}{2bm} \varepsilon$. However, if we want an explicit quality bound, we have to compute M and m.

SUM UP: f in C2 convex, minimum achieved, strongly convex and gradient lipschitz.

The subgradient method:
The subgradient method is close to the of gradient decent, but works in a non-differentiable setting. If we want to minimise the function $f(x)$ without constraints in a differentiable setting, we have the intuition that the gradient of f points in the steepest increasing direction. So intuition tells us that we can follow the opposite of the gradient, we're locally minimising the function. So our minimising sequence should have form $x_{k+1} = x_k - a_k g_k$ for scalars $a_k > 0$ and (imitations of) gradients g_k at x_k.
If f is a convex function which has an attained minimum in x^*, then this is the unique minimum of the function, and we'll expect the sequence of x_ks to converge to x^*.
In this context, we'll try to prove convergence by bounding $\|x_k - x^*\|$ by values converging to 0.
More can be said about $\|x_k - x^*\|^2$, as we can distribute the dotproduct and applying the recursive definition: $\|x_{k+1} - x^*\|^2 = \|x_k - a_k g_k - x^*\|^2 = \|x_k - x^*\|^2 - 2\,(x_k - x^*)\cdot a_k g_k + \|a_k g_k\|^2$.
Induction yields $\|x_{k+1} - x^*\|^2 = \|x_1 - x^*\|^2 - 2\sum_{i=1}^{k}(x_i - x^*)\cdot a_k g_k + \sum_{i=1}^{k}\|a_k g_k\|^2$ and positivity provides $2\sum_{i=1}^{k}(x_i - x^*)\cdot a_k g_k \leqslant \|x_1 - x^*\|^2 + \sum_{i=1}^{k}\|a_k g_k\|^2$.
Now we use the notion of subgradient: a subgradient of f at x is a vector g that verifies $f(y) - f(x) \geqslant g\cdot(y - x)$ for all y (it underestimates the differential). If g_k is a subgradient of f at x_i, then we can bound $2\sum_{i=1}^{k}(x_i - x^*)\cdot a_k g_k \geqslant 2\sum_{i=1}^{k} a_k\,(f(x_i) - f\,(x^*))$.
At this stage, we make a change in proof objective: we'll try to show that $(f(x_k) - f\,(x^*))$ converges to 0, as we have an upper bound on those values. More precisely, we have the bound $\sum_{i=1}^{k} a_k\,(f(x_i) - f\,(x^*)) \geqslant \sum_{i=1}^{k} a_k\left(\min_{i\leqslant k} f(x_i) - f\,(x^*)\right)$ so that we actually have to keep track of the minimum value encountered so far $\min_{i\leqslant k} f(x_i)$.
We have $\left(\min_{i\leqslant k} f(x_i) - f\,(x^*)\right) \leqslant \dfrac{\|x_1 - x^*\|^2 + \sum_{i=1}^{k} a_k^2\|g_k\|^2}{2\sum_{i=1}^{k} a_k}$. By making introducing the following conditions, we get convergence:

- The subgradients g_k are bounded (for example, f is a Lipschitz function)
- The sequence $\sum_{i=1}^{k} a_k^2$ is bounded and $\sum_{i=1}^{k} a_k$ diverges to infinity (for example $a_k = \dfrac{1}{k}$).

In that case, $\dfrac{\|x_1 - x^*\|^2 + \sum_{i=1}^{k} a_k^2\|g_k\|^2}{2\sum_{i=1}^{k} a_k} \xrightarrow[\infty]{r} 0$.
ADD STOP CRITERION AND QUALITY BOUND ???

Ex.?: We wich to use the subgradient method to minimise $f(x) = \max_{i\leqslant m}(a_i\cdot x - b_i)$. Show that f is convex, find a subgradient for f at a given x and show that this family of subgradients is bounded, so that we may apply the subgradient method.
As an example, in the context of statistical regression, we try to find an affine function $f(x) = w\cdot x + w_0$ that estimates a real-valued parameter as a function of other parameters in $\mathbb{R}^n$, given a set of observed data $(x_i, y_i)_{i\leqslant m}$. The goal is to find f so that $f(x_i) \approx y_i$ for all data points. We can get a good f by minimising the total error $E(w, w_0) = \sum_{i=1}^{m} |w\cdot x_i + w_0 - y_i|$. Prove that $E(w, w_0) = \max_{i\leqslant m}(a_i\cdot(w, w_0) - b_i)$ for a certain choice of a and b.

Finally, explain how one can use the subgradient method on this particular type of function as a routine to solve LPs approximately under certain conditions.

18.2 Newton method

Unconstrained case:
The Newton method is a descent method similar to the gradient method. We construct a sequence (x_k) that will converge to a point x^* achieving the minimum p^* of an objective f by a step of form $x_{k+1} = x_k + d_k$. In gradient descent, we chose $d_k = -t_k \nabla f(x_k)$ based on he intuition that the gradient points to the steepest ascent direction. This intuition comes from the fact that for $d \in S^1$ and $t > 0$ small enough, $f(x+td) - f(x)$ has the sign (and relative value wrt. t) of $\nabla f(x)^t d$, for which Cauchy-Schwartz $\nabla f(x)^t d \leqslant \|\nabla f(x)\|.\|d\|$ implies that this quantity is maximum in $d = \dfrac{1}{\|\nabla f(x)\|}\nabla f(x)$. What if we used higher order approximations of $f(x+td) - f(x)$ to determine the descent direction ?

For small $\|d\| > 0$, we have $f(x+d) - f(x) \approx \nabla f(x)^t d + \dfrac{1}{2} d^t \nabla^2 f(x) d$. We seek the d minimising this second part, which is a convex function if $\nabla^2 f(x)$ is positive definite, which we assume, so that the first order condition $\nabla f(x) + \nabla^2 f(x) d = 0 \Leftrightarrow d = \left(\nabla^2 f(x)\right)^{-1} (-\nabla f(x))$, where invertibility is due to positive definiteness. In this direction, we can estimate the improvement made to the objective by $f\left(x - \left(\nabla^2 f(x)\right)^{-1} \nabla f(x)\right) - f(x) \approx -\dfrac{1}{2} \nabla f(x)^t \left(\nabla^2 f(x)\right)^{-1} \nabla f(x)$, by assuming $f \in C^2$ so that $\nabla^2 f(x)$ is symmetric.

Therefore, the Newton method performs updates of form $x_{k+1} = x_k - t\left(\nabla^2 f(x)\right)^{-1} \nabla f(x)$.
We'll find the step size t with backtracking line search again, and use an exit condition of form $\dfrac{1}{2} \nabla f(x)^t \nabla^2 f(x)^{-1} \nabla f(x) \leqslant \varepsilon$, meaning that we stop when we expect the decrease in objective value to be to insignificant.

Line search in this context consists in searching $q = 0, 1, ...$ for a $b \in]0,1[$ such that $f(x_{k+1}) \leqslant f(x_k) - b^q \dfrac{1}{2} \nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k)$. This is because, by using a truncated Taylor development, we have $f(x_{k+1}) = f(x_k) - t\nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k) + \dfrac{1}{2} t^2 \nabla f(x_k)^t \left(\nabla^2 f(x_k)^{-1}\right)^t \nabla^2 f(z_k) \nabla^2 f(x_k)^{-1} \nabla f(x_k)$, and assuming again that ∇f is M-Lipschitz on $\mathbb{R}^d$ (as in gradient descent), we have bound $f(x_{k+1}) - f(x_k) \leqslant -t \nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k) + \dfrac{1}{2} t^2 M \left\|\nabla^2 f(x_k)^{-1} \nabla f(x_k)\right\|^2$.

Next, by assuming that f is strongly convex on $\mathbb{R}^d$ (as in gradient descent), we can use the fact that $\nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k) = \left(\nabla f(x_k)^t \left(\nabla^2 f(x_k)^{-1}\right)^t\right) \nabla^2 f(x_k) \left(\nabla^2 f(x_k)^{-1} \nabla f(x_k)\right) \geqslant m \left\|\nabla^2 f(x_k)^{-1} \nabla f(x_k)\right\|^2$, to get bound $f(x_{k+1}) - f(x_k) \leqslant \left(\dfrac{M}{2m} t^2 - t\right) \left(\nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k)\right)$. Note that this also juistifies our stopping criterion, as $\left(\nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k)\right)$ really does bound the objective decrement.

Following the arguments of gradient descent, for $t \leqslant \dfrac{m}{M}$ we have $\left(\dfrac{M}{2m} t^2 - t\right) \leqslant -\dfrac{t}{2}$, so that finally $f(x_{k+1}) - f(x_k) \leqslant -\dfrac{t}{2} \nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k)$, which is the line search stopping criterion.

Note that when line search terminates, that is for q such that $b^q > \dfrac{bm}{M}$, we can bound the decrement by $f(x_{k+1}) - f(x_k) \leqslant -\dfrac{bm}{2M} \nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k)$.
Note also that by the previously used $\nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k) \geqslant m \left\|\nabla^2 f(x_k)^{-1} \nabla f(x_k)\right\|^2 \geqslant 0$, we have $f(x_{k+1}) - f(x_k) \leqslant -\dfrac{bm}{2M} \nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k) \leqslant 0$, so this truly is a descent method.

To analyse the algorithm, we consider the size of $\nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k)$, which we expect to decrease. If $\nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k) \geqslant \mu$ for a total of n consecutive iterations, then $f(x_{k+1}) - f(x_k) \leqslant -\frac{bm}{2M}\mu$ and by telescopic sums and the bound $f(x) \geqslant p^*$, we have $p^* - f(x_0) \leqslant -n\frac{bm}{2M}\mu$, which reformultaes to $\frac{2M\left(f(x_0) - p^*\right)}{bm\mu} \geqslant n$. So for any μ, the case $\nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k) \geqslant \mu$ can occur only a finite number of consecutive times.
There must then occur an iteration with $\nabla f(x_k)^t \nabla^2 f(x_k)^{-1} \nabla f(x_k) < \mu$.
In particular, for $\mu = 2\varepsilon$, we know that the stopping criterion of our algorithm is reached, and that in $\frac{M\left(f(x_0) - p^*\right)}{bm\varepsilon}$ iterations.

TO DO: quality bound and quadratic convergence etc...
ASSUMPTIONS: f achieves minimum, is convex and in C^2, with positive definite hessian everywhere, grad lipschitz and strong convex

Linear equation constraint:

When dealing with a general optimisation problem $\min f_0(x)$ st. $\begin{cases} g_i(x) = 0 \\ h_i(x) \leqslant 0 \end{cases}$, we can get simplify the problem if there is a smooth parametrization of the solution space available. The problem is that such a parametrization may not be available. But for problems of form $\min f_0(x)$ st. $Ax = b$, we can describe the solutions to $Ax = b$ with a smooth parametrization. Indeed, the solutions to $Ax = b$ can be obtained by finding an specific solution x_0 to it via Gaussian elimination, and determining a basis of $\ker(A)$, again with Gaussian elimination. If B is a matrix who's columns for a basis of $\ker(A)$, then all solution to $Ax = b$ are of the form $x = By + x_0$, where $y \in \mathbb{R}^{rank(A)}$.
Then, the problem $\min f_0(x)$ st. $Ax = b$ is equivalent to $\min f_0(By + x_0)$ over $\mathbb{R}^{rank(A)}$.
To use the Newton descent on this problem, we must assume that f_0 atteins a minimum on the solutions to $Ax = b$, a condition that can't be derived from f_0. Indeed, consider the example of the convex $f_0(x, y) = x$, which doesn't attain a minimum on the plane, to be minimised under linear constraints $x = 0$, where its minimum everywhere. Convexity of f_0 implies that of $f_0(By + x_0)$ and so does smoothness.
The gradient is $\left(\nabla f(By + x_0)^t B\right)^t$ and the hessian is $B^t \nabla^2 f(By + x_0) B$. If the gradient of f is Lipschitz, so is $\left(\nabla f(By + x_0)^t B\right)^t$, as $\left\|B^t \nabla f(By + x_0) - B^t \nabla f(By' + x_0)\right\| \leqslant \left\|B^t\right\| \left\|\nabla f(By + x_0) - B^t \nabla f(By' + x_0)\right\| \leqslant \left\|B^t\right\| L \|B(y - y')\| \leqslant \left\|B^t\right\| L \|B\| \|y - y'\|$ by using matrix norms. Similarly, note that $x^t B^t \nabla^2 f(By + x_0) Bx = (Bx)^{t t} \nabla^2 f(By + x_0)(Bx)$ so that positive definiteness is conserved (as B is a basis), and we so is strong convexity, by taking m to be the minimum of the strict positive $x \mapsto (Bx)^{t t} \nabla^2 f(By + x_0)(Bx)$ over the unit sphere.

18.3 Barrier methods

Barrier methods allow to solve problems of form $\min(f_0(x))$ st. $f_i(x) \leqslant 0$ for convex functions in $\{0, ..., m\}$. They do so by an idea similar to Lagrange relaxation: including the constraints into the objective. Our goal is to do this in a way that allows us to use the Newton method. One way of penalizing the violation of constraints is by modifying the objective to $f_0(x) - \sum_{i\in[m]} \ln(-f_i(x))$, where $\phi(x) = -\sum_{i\in[m]} \ln(-f_i(x))$ is called a **barrier function**.

Multiple problems arise from this modification: first, the constraints haven't been taken care of, as $\sum_{i\in[m]} \ln(-f_i(x))$ is only defined for points verifying the constraints, second, the minimum might not be close to one of the original function.
The second problem could lessened by introducing a large $t > 0$ and switching to objective $tf_0(x) + \sum_{i\in[m]} \ln(-f_i(x))$ so that the main function to minimise is f_0.
For us to apply the Newton method to $tf_0(x) + \sum_{i\in[m]} \ln(-f_i(x))$, certain assumptions must be made. First, we will assume that the constraints $f_i(x) \leqslant 0$ for $i \in [m]$ form a bounded region: this will ensure that $tf_0(x) + \sum_{i\in[m]} \ln(-f_i(x))$ has a minimum p^*, attained in the interior ($f_i(x) < 0$ for $i \in [m]$).
Note that $\phi(x) = -\sum_{i\in[m]} \ln(-f_i(x))$ is convex, as a sum of compositions of the convex decreasing $-\ln$ and the concave $-f_i$. So that $tf_0(x) + \sum_{i\in[m]} \ln(-f_i(x))$ is convex as well. It's also C^2 if all f_i are.
The gradient and hessian of $\phi(x)$ are $\nabla\phi(x) = -\sum_{i\in[m]} \frac{1}{f_i(x)}\nabla f_i(x)$ and $\nabla^2\phi(x) = \sum_{i\in[m]} \frac{1}{f_i(x)^2}\nabla f_i(x)\nabla f_i(x)^t - \sum_{i\in[m]} \frac{1}{f_i(x)}\nabla^2 f_i(x)$. Since $\nabla f_i(x)\nabla f_i(x)^t$ is positive definite unless $\nabla f_i(x) = 0$, and $f_i(x) < 0$ in the interior of the feasible solution space, by assuming that each f_i has a positive definite hessian and no critical points in the interior of the feasible solution set (the latter not being required of f_0), then the objective $tf_0(x) + \sum_{i\in[m]} \ln(-f_i(x))$ has a positive definite hessian, as a sum.

PROBLEM: gradient not necessarily Lipschitz and no strong convexity...

We now deal with the problem that the introduction of the barrier function didn't really take care of the constraints, as they are now implicit in the domain of the objective. If we can find a solution x in the interior of the feasible space, that is to say $f_i(x) < 0$ for $i \in [m]$, that we can investigate if the Newton method can be adapted so as to never consider points outside of the domain. We can then try to find a initial interior solution by solving a auxiliary problem, like we did for the simplex method.

We can modify the Newton method at the line search step as follows. If $x_{k+1}(t)$ is outside the domain (the feasible set, which one can check via function evaluation), we set $f_0(x_{k+1}(t)) = +\infty$: this will lead line search to reject points outside the domain, as we compare $f_0(x_{k+1}(t)) = +\infty$ to a finite quantity. The question is whether there is a q such that $x_{k+1}(b^q)$ is in the domain, so that after a few values of q , we're back to normal line search. The answer is yes, since x_k is an interior point, so that for small enough t, therefore also for large enough q, we have $x_{k+1}(t) = x_k - t\left(\nabla^2 f(x)\right)^{-1}\nabla f(x)$ will be an interior point (satisfy $f_i(x) < 0$ for $i \in [m]$, by continuity). It may then be that the first q for which the point is in the

domain is not the first q for which the exit condition of line search is met. However, since the first q to meet both criteria is greater then those meeting each criteria individually, we have $b^q > \frac{b^{s_k} m}{M}$ for some $s_k \geqslant 1$.
PROBLEM: s depends on k, but we need a bound independent of k to proceed as in the proof of the original Newton method.

One may wonder why one hasn't used this modification for the regular Newton method to handle feasibility with respect to constraints, without the barrier function. The point is that the barrier function will prevent the gradient to point towards the "outside" of the feasible set, so that we expect line search to reach feasible points quite quickly.

The immediate question is how the optimal solution of the barrier method differs from that of the originial problem. If $x^*(t)$ denotes the solution outputted by the Newton method, then since Newton method stops when the gradient is small (technically a quadratic form evaluated in the gradient, and strong convexity implies the gadient is small as well), we have that the gradient of the barrier objective at $x^*(t)$, $\left| \nabla f_0(x^*(t)) - \sum_{i \in [m]} \frac{1}{f_i(x^*(t))} \nabla f_i(x^*(t)) \right| \leqslant \varepsilon$.
PROBLEM: is any part of this from Boyds book actually true ???

18.4 Solutions

Ex.?:
f is a maximum over affine functions, so it's convex (use linearity and bound).
For j maximising $f(x) = \max_{i \leqslant m}(a_i \cdot x - b_i)$, $f(y) - f(x) \geqslant (a_j \cdot y - b_j) - (a_j \cdot x - b_j)$ since $f(y) = \max_{i \leqslant m}(a_i \cdot y - b_i) \geqslant (a_j \cdot y - b_j)$. So $f(y) - f(x) \geqslant a_j \cdot (y - x)$, so that a_j is a subgradient for f at x. This sequence is finite, so bounded.

For all choices of sign $s_i \in \{+1, -1\}$, we have $\sum_{i=1}^{m} |w \cdot x_i + w_0 - y_i| \geqslant \sum_{i=1}^{m} s_i(w \cdot x_i + w_0 - y_i)$, with inequality being tight for at least on set of choices for s_i. So we have:

$$\sum_{i=1}^{m} |w \cdot x_i + w_0 - y_i| = \max_{\substack{i \leqslant m \\ s_i \in \{+1,-1\}}} \left(\sum_{i=1}^{m} s_i(w \cdot x_i + w_0 - y_i) \right)$$

Note that there are 2^m functions to check, so finding a subgradient takes exponential time in m.
We can use the subgradient method to solve a feasibility problem: for a system $Ax \leqslant b$, if the subgradient method for $f(x) = \max_i(A_{i*}x - b_i)$, finishes with a negative optimal value attained in x^*, then $Ax^* - b \leqslant 0$ and x^* is a feasible point. We can then use binary search to solve the LP approximately if we know that the objective values are in te range $[l, u]$. For a minimisation problem we solve $\begin{cases} Ax \leqslant b \\ c \cdot x \leqslant \frac{l+u}{2} \end{cases}$ repeatedly, setting $u \leftarrow \dfrac{l+u}{2}$ when the system has a solution and $l \leftarrow \dfrac{l+u}{2}$ if the subgradient method didn't certify feasibility (this might not imply actual infeasibility of the system), iterating till the distance $u - l$ has reached a arbitrarily set error tolerance.

19 More convex optimization methods

19.1 Polyhedral approximation

Outer linearisation:
We recall the geometric version of convexity of a function, for the differentiable case. If $f \in C^1$ is convex, the $f(y) \geqslant f(x) + \nabla f(x)^t(y-x)$. Now if we consider the epigraph of f, given by $C_f = \{(x,z) : z \geqslant f(x)\}$, and the graph of f, given by $G_f = \{(x,z) : z = f(x)\}$, then we can interpret the plane $f(x) + \nabla f(x)^t(y-x) = 0$ as the tangent plane at x of f, and convexity means that the graph and epigraph are in the halfspace above this plane. Hence, the intersection of many such halfspaces contains the graph and epigraph, and for such tangent halfspaces at a grid of points, we expect this polyhedron to approximate the epigraph. We are then interested in the idea of minimising hight z on this polyhedron to approximate minimising f.

We will deal with problems of form $\min f(x)$ st. $Ax \leqslant b$, where $f \in C^1$ is convex and the polyhedron defined by $Ax \leqslant b$ is bounded. We will solve approximative LPs of the form $\min z$ st. $z \geqslant f(x_i) + \nabla f(x_i)^t(x - x_i)$ of $i \in [k]$ and $Ax \leqslant b$. These LPs are bounded as $Ax \leqslant b$ is, and will therefore always have a finite minimum. We note that if for the minimum we have $z^* = f(x^*)$, where in general we have $z^* \leqslant f(x^*)$, because $f(x^*) \geqslant f(x_i) + \nabla f(x_i)^t(x^* - x_i)$ for all i by convexity, so that $(x^*, f(x^*))$ is feasible for the LP, then x^* minimises f, as for any y such that $Ay \leqslant b$, $(y, f(y))$ is feasible for the LP by convexity of f and hence $f(y) \geqslant z^* = f(x^*)$. If however we have $z^* < f(x^*)$, we get a finer approximation of the epigraph by adding constraint $z \geqslant f(x_{i+1}) + \nabla f(x_{i+1})^t(x - x_{i+1})$ where $x_{i+1} = x^*$. This constraint cuts off the previous LP-minimser (x^*, z^*) from the feasible set. Note that we can solve the next LP with a simple post-optimization step.

We start with a first LP for some x_0 in the contraint polytope, and denote by (x_k, z_k) the sucessive values of the kth LP solved, if $z^* = f(x^*)$ didin't occure by chance until then. Note that z_k will be an increasing sequence, since (x_{k+1}, z_{k+1}) satisfies the constraints of the kth LP among two new ones, so that $z_k \leqslant z_{k+1}$.

Convergence:

The sequence x_k has a subsequence that converges to a minimiser of f on $Ax \leqslant b$.

Proof: Since the polytope $Ax \leqslant b$ is compact, Bolzano-Weistrass guarantees a convergent subsequence with limit deonted x^*, and we denote it with x_k also, wlog. At iteration k, we have for all $i < k$, by feasibility $f(x_i) + \nabla f(x_i)^t(x_k - x_i) \leqslant z_k$. Now, for any x of the constraint polytope, we still have $z_k \leqslant f(x)$: $(x, f(x))$ is feasible in the kth LP as its in the epigraph, hence p is grater then the minimum of that LP. This is true for a minimiser x^- of f on $Ax \leqslant b$, and for the limit x^*, which is also in the polytope. So on the one hand, $z_k \leqslant f(x^*)$ so that $\lim_{k\to\infty} z_k \leqslant f(x^*)$, which exists as z_k is increasing an upper-bounded, and on the other hand $\lim_{k\to\infty} \left(f(x_{k-1}) + \nabla f(x_{k-1})^t(x_k - x_{k-1})\right) \leqslant \lim_{k\to\infty} z_k$, where by convergence $\lim_{k\to\infty}(x_k - x_{k-1}) = 0$, and by boundedness of the continuous ∇f on the compact polytope, $\lim_{k\to\infty} \left(f(x_{k-1}) + \nabla f(x_{k-1})^t(x_k - x_{k-1})\right) = \lim_{k\to\infty}(f(x_{k-1}) + 0) = f(x^*)$, also by continuity of f. Therefore $f(x^*) \leqslant \lim_{k\to\infty} z_k \leqslant f(x^*)$ so that $\lim_{k\to\infty} z_k = f(x^*)$. However, we also have $z_k \leqslant f\left(x^-\right)$ and therefore $f(x^*) = \lim_{k\to\infty} z_k \leqslant f\left(x^-\right)$. Since x^- was a minimiser, this implies that the limit x^* is too.

Inner linearisation:
Inner linearization isn't exactly what we can expect from its name. Whereas in outer linearisation, we exploited the fact that the graph of a convex function is above its tangent planes, we could think that inner linearisation, uses the fact that the graph of a convex function is below its chords (simplices with vertices on the graph).
However, inner linearisation uses sequential LPs to deal with a complex solution set, for an easy objective function. The strength of inner linearisation is best illustrated by an example.

We consider an unconstrained multi-source convex flow problem with quadratic costs, and in a peculiar form. Indeed, our variables will be indexed by paths! We're given a graph G and a set of k pairs of nodes s_i, t_i, the source and target of commodity $i \in [k]$. The goal is to ship r_i of this commodity along the graph. If a total of c units of commodities, all commodities confused, use edge e to be shipped, the a cost $q_e c^2$ has to be paid, with $q_e \geqslant 0$. This objective will have the effect of seeking "spread out" flows: for example for two disjoint paths with same length l and $q = 1$, the cost of using one path only is lr^2 while that of splitting the commodities along the path is $2l\left(\frac{r}{2}\right)^2 = \frac{1}{2}lr^2$, which is better.
We model the problem with the set of s_i-t_i-paths P_i of the graph, by using variable x_p to denote the number of commodities to be shipped along the path $p \in P_i$. The set P_i may be exponential in the size of the graph, but we'll see that we never have to access it in its integrality in what follows. For now, we remark that we have constraints $x \geqslant 0$ and $\sum_{p \in P_i} x_p = r_i$, and objective $\sum_{i \in [k]} \sum_{e \in E} q_e \left(\sum_{p \in \cup P_i : e \in p} x_p \right)^2$.
More generally inner linerisation solves $\min f(x)$ st. $Ax \leqslant b$ for a convex $f \in C^1$ and a bounded polyhedron $Ax \leqslant b$. The algorithm starts with a feasible vertex x_0 of $Ax \leqslant b$. In the case of our flow problem, we can use any tuple of s_i-t_i-paths to ship the all the stock of the commodities along, that we find with DFS for example. This is a vertex since all but one $x_{P_i} \geqslant 0$ is tight. At each iteration, we'll produce an iterate x_j that will converge to a minimiser, as well as a set of vertices $x'_0, ..., x'_j$ that will approximate the solution space for the next iteration. We start with $x'_0 = x_0$.
We consider the last iterate x_j, and start by approximating the objective by its gradient at that point $\nabla f(x_j)^t$ and solving the LP $\min \nabla f(x_j)^t (x - x_j)$ st. $Ax \leqslant b$ to get vertex x'_{j+1} of $Ax \leqslant b$ (which we assume bounded, so that a vertex solution exists). In the case of our flow problem, we don't actually have to solve an LP. Indeed, we see that the variables of different P_i for different i have no constraint in common. The point is that now the objective is linear, so that the optimization problem is fully decoupled, as the solution of $\min \left(c^t x + d^t y\right)$ on $\begin{cases} Ax \leqslant a \\ By \leqslant b \end{cases}$ is the sum of the solutions of $\min c^t x$ on $Ax \leqslant a$ and $\min d^t y$ on $By \leqslant b$. So for each commodity, we solve a linear objective over a simplex $x_{P_i} \geqslant 0$ and $1^t x_{P_i} = r_i$, who's solution is at a vertex $\begin{cases} x_p = r_i \\ x_{P_i \setminus p} = 0 \end{cases}$ for some path $p \in P_i$ of lowest cost. Since here $\nabla f\left(x^j\right) = \sum_{p \in \cup P_i} \left(\sum_{e \in p} 2q_e \left(\sum_{p' : e \in p'} x^j_{p'} \right) \right) e_p$ (iterates are now in power notation), we see that the cost of paths are the same as additive costs in a graph with edge-weights $w_e = 2q_e \left(\sum_{p' : e \in p'} x^j_{p'} \right)$. So we're actually looking for a shortest path for each commodity, and we can use Dijkstra instead of running the LP on an exponential number of constraints.

The next phase, in the general case, is to check if the iterate x_{j+1} of the problem for the linearised objective happens to be optimal. We do this by checking the sign of $\nabla f(x_j)^t (x'_{j+1} - x_j)$. Indeed, if

$\nabla f(x_j)^t(x'_{j+1} - x_j) \geqslant 0$, then since x'_{j+1} is a minimum of the LP $\nabla f(x_j)^t(x - x_j) \geqslant 0$ for all feasible x, which is equivalent to the point x_j being a minimiser of f (recall that the contrapositive holds: if there was an x with strictly smaller objective, then for any ε, since $f\left(\frac{\varepsilon}{2}x + \left(1 - \frac{\varepsilon}{2}\right)x_j\right) < f(x_j)$ by convexity, and $f\left(\frac{\varepsilon}{2}x + \left(1 - \frac{\varepsilon}{2}\right)x_j\right) - f(x_j) = \frac{\varepsilon}{2}\nabla f(x_j)^t(x'_j - x_j) + o(\varepsilon)$, so that $\nabla f(x_j)^t(x - x_j) < 0$ in the limit). We postpone explaining how this step is done for the flow problem.
If however $\nabla f(x_j)^t(x'_{j+1} - x_j) < 0$, then we compute a next iterate x_{j+1} by solving $\min f(x)$ on $x \in conv(x'_0, ..., x'_{j+1})$, aka. $\min f\left(x'^t\lambda\right)$ for λ in the $(j+1)$-diemensional standard simplex. We hop that the structure of the problem allows for this step to be easy. Otherwise, we have to use interior point methods, where the only advantage gained is that we decreased the dimension considerably, to $j+1$. Note that for the flow problem, the x' have support of size k, so that $x'^t\lambda$ is not computationally prohibitive. Also x_{j+1} is in that convex hull, so it has support of size at most $(j+1)k$. We can now see that computing $\nabla f(x_j)^t(x'_{j+1} - x_j)$ isn't computationally prohibitive as $(x'_{j+1} - x_j)$ has support of size at most $(j+2)k$. We can keep track of the support when implementing the algorithm.

We remark that since $\nabla f(x_j)^t(x'_{j+1} - x_j) < 0$, the point x'_{j+1} could not have been in $conv(x'_0, ..., x'_j)$, since x_j minimises f on that set, so that by convexity $\nabla f(x_j)^t(x - x_j) \geqslant 0$ for all x in it, so not x'_{j+1} in this case. Hence, the convex hulls $conv(x'_0, ..., x'_j)$ get larger an larger, as we add extreme points of $Ax \leqslant b$ at each step that didn't produce an optimum. This provides termination, as we can't add vertices to get bigger hulls indefinitely: we assumed $Ax \leqslant b$ to be bounded, so that this polytope has finitely many vertices. However, with this argument we must expect possible exponential runtime.
As a final remark, note that x_j only has to approximately minimise f on $conv(x'_0, ..., x'_j)$, so that iterior point algorithms may be used, since $\nabla f(x_j)^t(x'_{j+1} - x_j) < 0$ is a strict inequality. The error of interior point method must however be smaller that the variation of $\nabla f(x)^t(x'_{j+1} - x)$.

Duality:

19.2 Proximal algorithms

Proximal algorithms are the equivalent to regularisation or preconditioning from numerics.

19.3 Solutions

20 Lattice geometry methods for ILP

20.1 Geometric integer programming

Recall the problem of solving the IP $\max c\cdot x$ st. $Ax \leqslant b, x \in \mathbb{Z}^n$. As was done in the first versions of the ellipsoid method, on can (approximately) solve this optimization problem by applying binary search and solving feasibility problems as subroutine. For estimated bounds $l \leqslant \min(c\cdot x : Ax \leqslant b, x \in \mathbb{Z}^n) \leqslant \max(c\cdot x : Ax \leqslant b, x \in \mathbb{Z}^n) \leqslant u$, we can apply binary search on $[l, u]$ by asking at each step if we can find, for current midpoint-value m, a point such $Ax \leqslant b, x \in \mathbb{Z}^n$ and $c\cdot x \geqslant m$.

The algorithm we present to solve the feasibility problem, known as Lenstra's algorithm, is based on multiple observations. All full-dimensional bounded polyhedra contain a maximum volume ellipsoid. If we have a way of finding this ellipsoid and a way of finding an integer point in an an ellipsoid, then we could compute this ellipsoid for the polytope and check if it has an integer point. If it doesn't, the brilliant idea is that by slicing up the polytope by hyperplanes so that all integer points of the polytope are on one of these hyperplanes, we can perform the same operation recursively on the polytopes that are the intersections of the initial polytope and the hyperplanes slicing it (which we consider in their affine hull to get full-dimensionality). If at some stage, an integer point is found, then it's an integer point of the initial polytope and otherwise, the recursion proceeds until the hyperplanes have dimension zero (they're points). In such a worst case, we've reached the integer points of the polytope, and if none of them are returned, it must be the case that the polytope had no integer points in the first place.
Finally, if we have a clever way of slicing the polytope, then not too many subproblems are created at each step, and the runtime of the algorithm (despite being exponential) will be relatively good.
In the algorithm we've described, we can replace the search for a maximum volume ellipsoid by that of a maximum volume ball. This simplifies the subroutine, as we know that finding a maximum volume ellipsoid requires solving a convex program, while finding a maximum volume ball requires solving a linear one. But in that case, we'll loose the bound on the number of sub-problems created when slicing.

The algorithm exploits the following sub-routine, provided by the proof of Khinchine's Flatness Theorem (or a variant of it...):

Algorithm: (Khinchine-algorithm)
Input: a polytope in H-description $Ax \leqslant b$ (with integer entries).

Output: a solution $Ax \leqslant b, x \in \mathbb{Z}^n$ **or** a direction $c \in \mathbb{Z}^n$ so that all integer points of the polytope $Ax \leqslant b$ are on the union of the hyperplanes $c\cdot x = d$ where $d \in [\min(c\cdot x : Ax \leqslant b), \max(c\cdot x : Ax \leqslant b)] \cap \mathbb{Z}^n$, and d takes at most $1 + n.2^{O(n^2)}$ values.

With this algorithm we can describe the main one that solves feasibility:

Algorithm:
Input: a polytope in H-description $Ax \leqslant b$ (with integer entries).

Procedure: As the algorithm is recursive, we'll denote it by $Alg(A, b)$. $Alg(A, b)$ starts by applying Khinchine's algorithm on $Ax \leqslant b$. If a solution verifying $Ax \leqslant b, x \in \mathbb{Z}^n$ is found, we return it and stop. Otherwise, we run the algorithm recursively on the slices of the polyhedron given by c, by using $Alg\left(\begin{pmatrix} A \\ c \\ -c \end{pmatrix}, \begin{pmatrix} b \\ d \\ -d \end{pmatrix}\right)$. If one of the recursive calls returns a solution, then we return this solu-

tion, as it's also one for $Ax \leqslant b$. Otherwise, we return the message that the polytope has no integer points.

Output: a solution $Ax \leqslant b, x \in \mathbb{Z}^n$ or the message that the polytope has no integer points.

As we've mentioned for Khinchine's algorithm, all integer points of the polytope must be in one of the hyperplanes given by c. So if the algorithm works for one dimension less then the current dimension, and none of the sub-applications of Khinchine's algorithm return integer points, then none of the hyperplanes given by c contain integer points, so that the polytope had no integer points in the first place.

If $T(n)$ is the worst case runtime of the algorithm in dimension n, then in the recursive call we get runtime $T(n-1)$ for each slice, of which Khinchine's algorithm guaranteed there be at most $1 + n2^{O\left(n^2\right)}$, So $T(n) \leqslant T(n-1)n2^{O\left(n^2\right)}$ and solving this formula we get $T(n) \in O\left(\prod_{i=1}^{n} i2^{O\left(i^2\right)}\right) = O\left(2^{O\left(n^3\right)}\right)$ as $\sum_{i=1}^{n} i^2 = O\left(n^3\right)$ and $n! \leqslant (2^n)^n = 2^{O\left(n^3\right)}$.

20.2 Lattices

EXPLAIN LATTICES... (unimodular transformations and the, Hermite normal form)

Definition:
A (full-rank) lattice is $A\mathbb{Z}^n$ for some invertible real matrix A, or more precisely the integer combinations of the vectors A_{*i}.

The vectors of A are said to for a **basis** of the lattice $A\mathbb{Z}^n$. A lattice can have multiple basis. For example, the lattice spanned by $\begin{pmatrix}1\\4\end{pmatrix}$ and $\begin{pmatrix}-1\\1\end{pmatrix}$ is also spanned by $\begin{pmatrix}0\\5\end{pmatrix}$ and $\begin{pmatrix}-1\\6\end{pmatrix}$, because we can express an integer combination $a\begin{pmatrix}1\\4\end{pmatrix}+b\begin{pmatrix}-1\\1\end{pmatrix}=(2a-b)\begin{pmatrix}0\\5\end{pmatrix}+(b-a)\begin{pmatrix}-1\\6\end{pmatrix}$ and conversely $a\begin{pmatrix}0\\5\end{pmatrix}+b\begin{pmatrix}-1\\6\end{pmatrix}=(a+b)\begin{pmatrix}1\\4\end{pmatrix}+(a+2b)\begin{pmatrix}-1\\1\end{pmatrix}$, so that integer combinations are in a one-to-one correspondence.

A lattice can theoretically be spanned by more vectors then a basis in given dimension allows. For example $\begin{pmatrix}1\\0\end{pmatrix}$, $\begin{pmatrix}0\\1\end{pmatrix}$ and $\begin{pmatrix}0\\\sqrt{2}\end{pmatrix}$ is a basis of the lattice it spanns, as the only integer combination providing 0 is the one with all 0 scalars, as $\sqrt{2}$ is irrational.

Definition:
For any full rank lattice $A\mathbb{Z}^d$ (in the sense that A represents a basis), it's dual lattice is $(A^{-1})^t\mathbb{Z}^d$.
We also have $\left(A^{-1}\right)^t\mathbb{Z}^d=\left\{x\in\mathbb{R}^d : x\cdot y\in\mathbb{Z},\forall y\in A\mathbb{Z}^d\right\}$

The name is well chosen: the dual of the dual lattice is the initial lattice, as transposition and inversion commute. To see that $\left(A^{-1}\right)^t\mathbb{Z}^d\subseteq\left\{x\in\mathbb{R}^d : x\cdot y\in\mathbb{Z},\forall y\in A\mathbb{Z}^d\right\}$, write $x=\left(A^{-1}\right)^t z$ and $y=Aw$ for integer vectors z and w and compute $x^ty=z^tA^{-1}Aw=z^tw\in\mathbb{Z}$.
Conversely, to see $\left(A^{-1}\right)^t\mathbb{Z}^d\supseteq\left\{x\in\mathbb{R}^d : x\cdot y\in\mathbb{Z},\forall y\in A\mathbb{Z}^d\right\}$, take x in the latter set and write it in the basis that $\left(A^{-1}\right)^t$ defines as $x=\left(A^{-1}\right)^t z$. Our goal is to prove that z is integer. So by taking $y=Ae_i$, the identity $x\cdot y\in\mathbb{Z}$ becomes $z^te_i=z_i\in\mathbb{Z}$, which gets us to the goal.

INCLUDE: Lattices with space definition, existence of bases, ref Barvinok

20.3 Khinchine's flatness theorem and algorithm

INTERNAL NOTE: case that polytope is not full dimensional, but $dim-1$ dimensional. Find affine hull, find integer points in hull via Hermite form, solve for different polytope as in Rothvoss p.37. ALSO, my khinchine includes the maximum volume ellipsoid stuff inside it.

We first reduce the dimension by placing the polytope in its affine hull and parameterize the latter.
As we'll be interested in the integer points of the polytope and the affine hull of the polytope is described by $t + \ker(M)$, we look for the integer point solutions to $Mx = Mt$. As in all cases we run the algorithm, the entries of the H-description of the polytope are integer (note that the c we'll slice the polytope along has integer entries), and as we can assume that t is integer under the condition that $\ker(M)$ contains an integer point, we con solve $Mx = Mt$ by computing the Herminte normal form of M. The solutions will then by a lattice spanned by basis B, translated by t.
So we're looking for the the integer points of $A(By + t) \leqslant b \Leftrightarrow (AB)y \leqslant (b - At)$ where y is in a lower dimensional vector space.

So we can assume that the polytope is full dimensional and we'll keep denoting it by the system $Ax \leqslant b$ (instead of the more cumbersome $(AB)y \leqslant (b - At)$).
The next reduction is a less obvious one: we'll prove the flatness theorem for ellipsoids instead of polytopes, and handle the polytope case with inscribed and circumcribed ellipsoids to the polytope.
So let's first deal with:

Algorithm: (Khinchine-algorithm for ellipsoids)
Input: an ellipsoid $a + T(B(0,1))$ where T is invertible.

Output: an integer point of the ellipsoid **or** a direction $c \in \mathbb{Z}^n$ so that all integer points of the ellipsoid are on the union of the hyperplanes $c \cdot x = d$ where
$d \in [\min(c \cdot x : x \in a + T(B(0,1))), \max(c \cdot x : x \in a + T(B(0,1))] \cap \mathbb{Z}^n$, and d takes at most $2^{O(n^2)}$ values.

Proof: We're looking for an integer point x in $a + T(B(0,1))$, or equivalently, an lattice point of $T^{-1}\mathbb{Z}^n$ in the ball $B\left(T^{-1}a, 1\right)$. Rewriting $T^{-1} = B$ and $T^{-1}a = d$, a basic idea would be to solve $Bx = d$ and consider the candidate lattice point $B\lfloor x \rfloor$, which we expect to be in the ball as it's close to the center. So we look at $\|B\lfloor x \rfloor - d\| = \|B(\lfloor x \rfloor - x)\|$ and in order to be able to give a condition for which this is $\leqslant 1$, we bound it by $\sum_{i=1}^{n} |\lfloor x \rfloor - x| . \|B_{*i}\| \leqslant n . \|B_{*i}\|_\infty$ (product, triangular inequality, fractional parts, maximum).
The condition $\|B_{*i}\|_\infty \leqslant \frac{1}{n}$ therefore implies that $B\lfloor x \rfloor$ is the lattice point we were looking for, so that $\lfloor x \rfloor \in a + T(B(0,1))$.
TO FINISH

20.4 The LLL-algorithm

For a lattice spanned by B, we're interested in finding the non-zero lattice vector of smallest euclidean norm.

Gram-Schmidt procedure:
The Gram-Schmidt procedure takes as input base B of $\mathbb{R}^n$ and returns an orthogonal base B^* as output, which is based off of B. The idea is that of successively deleting the components of basis vectors in previous basis vectors.
We start with $b_1^* := b_1$, and define $b_2^* := b_2 - s_{1,2}b_1^*$ with $s_{1,2}$ chosen so that $b_2^* \cdot b_1^* = 0$, which is achieved by $s_{1,2} = \dfrac{b_2 \cdot b_1^*}{\|b_1^*\|^2}$ (use combine the two equations and solve for $s_{1,2}$). We can iterate this by setting $b_j^* := b_j - \sum_{i<j} s_{i,j} b_i^*$, choosing $s_{i,j} = \dfrac{b_j \cdot b_i^*}{\|b_i^*\|^2}$ so as to get $b_j^* \cdot b_i^* = 0$ for all $i < j$ (recalling that we inductively have $b_k^* \cdot b_i^* = 0$ for all $i, k < j$).

We now turn to some interesting facts about the Gram-Schmidt procedures outcome. We first point out that $\det(B) = \det(B^*)$, as we can obtain B^* from B by operations on the columns (recall multilinearity of the determinant). Next, we can compute $\det(B^*)$ quite easily with using $(B^*)^t B^* = diag\left(\|b_i^*\|^2\right)$ (the dot-product matrix and orthogonality), so that on the one hand $\det\left((B^*)^t B^*\right) = \left(\prod_{i=1}^n \|b_i^*\|\right)^2$ and on the other $\det\left((B^*)^t\right)\det(B^*) = \det(B^*)^2$. We therefore have $\det(B^*) = \pm\prod_{i=1}^n \|b_i^*\|$.
Orthogonality of B^* carries many other interesting properties with it. Here's a crucial one for what's to come:

Proposition:
For all lattice vectors $x \in B\mathbb{Z}^n \backslash \{0\}$, $\|x\| \geqslant \min_i(\|b_i^*\|)$.

Proof: We write $x \neq 0$ in terms of B^* by substituting $b_j = b_j^* + \sum_{i<j} s_{i,j} b_i^*$. An interesting fact is that the multiple of b_n in x is the same as that of b_n^*, since b_n^* is introduced in the last substitution. In general, for $x = \sum_{i=1}^n y_i b_i$ and $x = \sum_{i=1}^n z_i b_i^*$ after substitution, for $j = \max(i : y_i \neq 0)$ (there is such an index as as $x \neq 0$), b_j^* appears only one in a substitution, so that $z_j = y_j$ for that index, and non of the b_i^* appear in the expression of for $i > j$.
Now by Pythagoras for $x = \sum_{i=1}^j z_i b_i$, we have $\|x\|^2 = \sum_{i=1}^j |z_i| \|b_i^*\|^2$ (compute $x \cdot x$ and use orthogonality), so that in particular, $\|x\|^2 \geqslant |z_j| \left\|b_j^*\right\|^2$ and since $z_j = y_j \in \mathbb{Z}\backslash\{0\}$, this means $\|x\| \geqslant \left\|b_j^*\right\| \geqslant \min_i(\|b_i^*\|)$.

This proposition sais that for $OPT = \min(\|x\| : x \in B\mathbb{Z}^n \backslash \{0\})$, we have $OPT \geqslant \min_i(\|b_i^*\|)$. As we know form the section on approximation algorithms, bounds like this one are useful in showing the nature of approximation algorithms. If we combine this with remark of great importance, which is that $b_1^* = 1.b_1$ is a lattice vector, then if we have some way of finding a basis of the lattice and on ordering of it (as the Gram-Schmidt procedure depends on an ordering of the basis) for which $\|b_1^*\| \leqslant \alpha \|b_i^*\|$, then we have $\alpha OPT \geqslant \|b_1^*\|$ and returning b_1 provides an α-approximation algorithm. This is what we now investigate:

The LLL-algorithm:

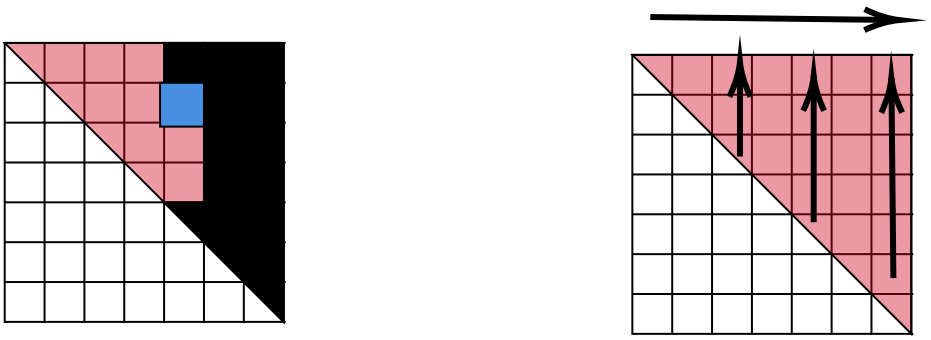
The first thing we'll investigate is how permuting the basis vectors affects the the Gram-Schmidt orthogonalisation. In fact, it turns out that we can do a lot with the study of the exchange of b_i and b_{i+1}.

How do we find a coefficient-reduced basis of the lattice ?
Starting from a basis B, we which to find a basis $\tilde{B}$ that spans the same lattice and who's Gram-Schmidt orthogonalisation is the same as Bs, so that $B^* = \left(\tilde{B}\right)^*$.
The idea is that the Gram-Schmidt orthogonalisation shouldn't be affected if we replace some basis vectors of B by linear combinations of previous basis vectors, as these will be deleted when deleting the component along the previous vectors during the procedure. In order for the coefficients of the Gram-Schmidt orthogonalisation to be small, since they depend on dot-products, an idea would be to use the previously described linear combinations to get a basis that is "as orthogonal as possible", so that the dot-products are small.

Let's investigate the simplest such operation: from B, we'll construct $\tilde{B} = update(B, l, k, q)$ so that $\widetilde{b_i} = \begin{cases} b_k + qb_l \ if \ i = k \\ b_i \ else \end{cases}$, for $l < k$ and $q \in \mathbb{Z}$. We see that $\tilde{B}$ is a basis of the same lattice, since we can re-express $\sum_{i=1}^{n} c_i b_i = (c_l - c_k q)b_l + c_k(b_k + qb_l) + \sum_{i \neq l,k} c_i b_i$ (and coefficients remain integral) and $\sum_{i=1}^{n} c_i \widetilde{b_i} = 0 \Leftrightarrow (c_l + c_k q)b_l + \sum_{i \neq l} c_i b_i = 0$ and by linear independence of basis B, first $c_i = 0$ for $i \neq l$, so that $c_k = 0$ and finally $c_l c_k q = 0 \Rightarrow c_l = 0$.

Now, we investigate the effect on the Gram-Schmidt orthogonalisation. For $i < k$, we produce the same vectors, as only previous vectors are involved in each step, so $\widetilde{b_i}^* = b_i^*$ for $i < k$. To compare the different orthogonalisations at k, we write $\widetilde{b_k} = b_k + qb_l \ = b_k^* + \sum_{i<l}(s_{ik} + qs_{il})b_i^* + (s_{lk} + q)b_l^* + \sum_{i>l}^{k-1} s_{ik} b_i^*$ by substituting the Gram-Schmidt orthogonalisation of B. Subtracting $\widetilde{b_k} = \widetilde{b_k}^* + \sum_{1}^{k-1} \widetilde{s_{ik}}\widetilde{b_i}^*$, we get $\widetilde{b_k}^* - b_k^* = \sum_{i<l} (s_{ik} + qs_{il} - \widetilde{s_{ik}})\, b_i^* + (s_{lk} + q - \widetilde{s_{lk}})\, b_l^* + \sum_{i>l}^{k-1} (s_{ik} - \widetilde{s_{ik}})\, b_i^*$, as $\widetilde{b_i}^* = b_i^*$ for $i < k$. By orthogonality of the Gram-Schmidt basis and $\widetilde{b_i}^* = b_i^*$, we deduce that all coefficients of the right side must be 0 (take a dot-product with $\widetilde{b_i}^* = b_i^*$) and finally that $\widetilde{b_k}^* = b_k^*$.
Then $\widetilde{b_i}^* = b_i^*$ for $i > k$, since previous vectors are involved in each step of Gram-Schmidt orthogonalisation and all previous vectors were the same for both basis. So the basis produce the same Gram-Schmidt orthogonalisation.

An improtant remark is that for this operation $s_{ik} = \widetilde{s_{ik}}$ for $i > l$. For the same reasons that we had $\widetilde{b_i}^* = b_i^*$ for $i < k$, we also have $s_{ij} = \widetilde{s_{ij}}$ for $i < j < k$. In the figure below, we've represented the situation: we represent the double indices by a square grid, where the red, blue and back squares represent the indices used of the s_{ij}. On the left, we see the the operation has changed the blue and black squares, whereas the red ones remain unchanged.



In fact the blue square was changed to $\widetilde{s_{lk}} = s_{lk} + q$. By applying this $update(B, l, k, q)$ iteratively to the basis with values of k and l changing as described on the right of the figure, we make sure that the value s_{lk} is translated by q and won't be affected ever again in the process. So to get small values of s_{lk}, we let $q = -\lfloor s_{lk} \rfloor$ or $q = -\lfloor s_{lk} \rfloor - 1$, depending on which is closer to s_{lk}, in that iteration. This will be the final value of s_{lk}. It has the important property that $|s_{lk}| \leqslant \frac{1}{2}$, as we can't simultaneously have $s_{lk} - \lfloor s_{lk} \rfloor > \frac{1}{2}$ and $\lfloor s_{lk} \rfloor + 1 - s_{lk} > \frac{1}{2}$ (else their sum is $1 > 1$). So the final basis we obtain after $\binom{n}{2}$ iterations of $update(B, l, k, q)$ is coefficient reduced.

20.5 Solutions

21 Semidefinte programming and applications in combinatorial optimization

21.1 Max-cut and the Goemans-Williamson algorithm

Recall the maximum cut problem: we're given a graph (V, E) and edge weights w and seek a bipartition of vertices into $A \subseteq V$ and $V \backslash A$ so that the total weight of edges with endpoints in either of the partition sets $\sum_{e \in \delta(A)} w_e$ is maximum.
In a previous exercise, we modeled this problem (for positive weights) as an IP and found a randomised $\frac{1}{2}$-approximation algorithms for the max-cut problem.
As a brief remark, we treat the case in which the weights are allowed to be negative. For positive weights on edge $\{u, v\}$, we keep constraints $\begin{cases} x_{uv} \leqslant y_u + y_v \\ x_{uv} \leqslant 2 - (y_u + y_v) \end{cases}$, but for negative weights, we use constraints $x_{uv} \geqslant |y_v - y_u| \Leftrightarrow \begin{cases} x_{uv} \geqslant y_u - y_v \\ x_{uv} \geqslant y_v - y_u \end{cases}$. This is because $|y_v - y_u|$ indicates if the edge $\{u, v\}$ is in the cut and because for the maximum solution, $x_{uv} = |y_v - y_u|$, for we could decrease it otherwise, as it appears in this constraint only, which would increase the objective (negative weight), contradicting maximality.

Alternatively, we can try to formulate the max-cut problem as a (non-linear) IP. We represent a bipartition of vertices by variables $y_v \in \{-1, 1\}$ for $v \in V$. Then, $\frac{1}{2}(1 - y_u y_v) = \begin{cases} 1 : \{u, v\} \in \delta(A) \\ 0 : \ else \end{cases}$ indicates whether edge $\{u, v\}$ is in the cut, so that the objective can be written as $W(A) = \sum_{e \in \delta(A)} w_e = \frac{1}{2} \sum_{(u,v) \in V^2} w_{(u,v)}(1 - y_u y_v)$. We can relax this to the following vector program (:=VP), which we'll see how to solve in a few paragraphs:

Problem: *Max-cut VP relaxation*

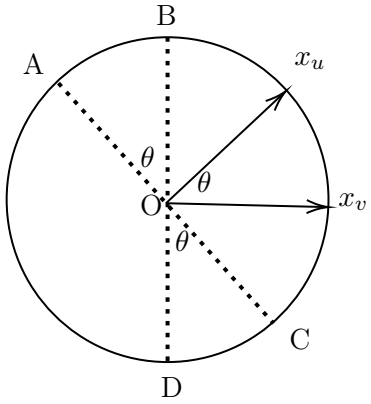
$$\begin{aligned} &\max \tfrac{1}{2} \textstyle\sum_{(u,v) \in E} w_{(u,v)}(1 - x_u \cdot x_v) \\ &\text{st. } x_v \in S^{|V|} = \left\{ x \in \mathbb{R}^{|V|} : \|x\|_2 = 1 \right\} \text{ for all } v \in V \end{aligned}$$

This is indeed a relaxation, as for all feasible solutions to the IP $y \in \{-1, 1\}^V$, we can consider the feasible solutions $x_v = (y_v, 0)$ of the VP, and remark that since $x_u \cdot x_v = y_u y_v$, the objective value of the VP is that of the IP. By denoting by z_{VP} the optimal value of this problem and z_{MC} that of max-cut, we therefore have $z_{VP} \geqslant z_{MC}$. Before discussing how VPs can be solved, we'll talk about the Goemans-Williamson algorithm.

The Goemans-Williamson algorithm produces a bipartition for the max-cut problem from the optimal solution x to the VP as follows: we get a random $r \in S^{|V|}$ with uniform distribution and partition the vertices into $A = \{u : x_u \cdot r \geq 0\}$ and $B = V \backslash A$. In visual terms: we partition the sphere along a random hyperplane and partition vertices according to the hemisphere their corresponding VP variable is in.
We now analyse the quality of such a bipartition. If W denotes the random variable that is the weight of such a bipartition, it's expectation is $E(W) = \sum_{(u,v) \in V^2} w_{uv} P(sign(x_u \cdot r) \neq sign(x_v \cdot r))$ where
$P(sign(x_u \cdot r) \neq sign(x_v \cdot r))$ is the probability that the dot-products $x_u \cdot r$ and $x_v \cdot r$ have the opposite sign

(so that $\{u, v\}$ is in the cut).

We can compute $P(sign(x_u \cdot r) \neq sign(x_v \cdot r)) = \frac{1}{\pi} arccos(x_u \cdot x_v)$ as follows. If x_u and x_v are colinear, since they're unit vectors, they're either equal or opposite and in both cases the equality holds (as we may assume that $x_v \cdot r \neq 0$ for all vertices, for the opposite event has probability 0). Otherwise, we can simplify the situation by considering the orthogonal projection of r on $Vect(x_u, x_v)$, normalised to $p_{uv}(r)$, which is uniformly distributed on unit circle of $Vect(x_u, x_v)$ (to see this, keep r multivariate Gaussian, use its rotational invariance and the fact that its marginals are Gaussian as well, then conclude by normalising). Indeed, $sign(x_v \cdot r) = sign(x_v \cdot p_{uv}(r))$ and similarly for u, so that we can picture the situation with our figure below, representing the plane $Vect(x_u, x_v)$.



The event who's probability we're computing then occurs precisely when $p_{uv}(r)$ is in the sectors of angles $\angle AOB$ and $\angle COD$, which have measure θ. By right angle rotations, we see that this is the angle between x_u and x_v, so that $x_u \cdot x_v = \cos\theta$, and since the event occurs with probability $\frac{2\theta}{2\pi}$ (uniform distribution), the result follows.

Now that we've established $E(W) = \sum_{(u,v)\in E} w_{uv} \frac{1}{\pi} arccos(x_u \cdot x_v)$, we turn to finding a bound α on the ratios $\frac{E(W)}{z_{MC}} \geqslant \frac{E(W)}{z_{VP}} \geqslant \alpha$ to judge the quality of the randomised approximation algorithm. For this, we need:

Lemma:

$$\frac{1}{\pi} arccos(y) \geqslant \alpha \frac{1}{2}(1-y)$$
$$\text{for } \alpha := \min_{0 \leqslant x \leqslant \pi} \left(\frac{2x}{\pi(1-\cos(x))} \right) > 0,87856$$

Proof: It's true for any α for $y = 1$, where both expressions are 0, so we can let $\alpha = \min \left(\frac{2arccos(y)}{\pi(1-y)} \right)$ over $[-1, 1[$. With the change of variable $y = \cos(x)$, this is $\alpha = \min \left(\frac{2x}{\pi(1-\cos(x))} \right)$. The function $f(x) = \frac{2x}{\pi(1-\cos(x))}$ has derivative $f'(x) = \frac{2}{\pi} \frac{(1-\cos(x)) - x\sin(x)}{(1-\cos(x))^2}$ and using some more calculus one can find the desired result. □

Now, for *positive* weights $w_{uv} \geq 0$ we can use the first inequality of the lemma to get $\sum_{(u,v)\in E} w_{uv}\frac{1}{\pi}arccos(x_u \cdot x_v) \geqslant \alpha \sum_{(u,v)\in E} w_{uv}\frac{1}{2}(1 - x_u \cdot x_v)$ So in this case $E(W) \geq \alpha z_{VP}$ and finally $1 \geqslant \frac{E(W)}{z_{MC}} \geqslant \alpha$.

Theorem: *Quality of the Goemans-Williamson algorithm*

For **positive** weights $z_{MC} \geqslant E(W) \geqslant \alpha z_{MC}$, where $\alpha := \min_{0\leqslant x\leqslant \pi}\left(\frac{2x}{\pi(1-\cos(x))}\right) > 0,87856$.

The Goemans-Williamson algorithm requires solving a vector/semidefinite programm, which we therefore now discuss.

Definition:
A **vector program** (VP) is an optimization problem of form:

$$\max \sum_{i,j} c_{i,j}(v_i \cdot v_j)$$
$$\text{st. } v_i \in \mathbb{R}^d \text{ and } \sum_{i,j,k} a_{i,j,k}(v_i \cdot v_j) = b_k \text{ all } i \in [d] \text{ and } k \text{ in a finite set.}$$

A **semidefinite program** (SDP) is an optimization problem of form:

$$\max \sum_{i,j} c_{i,j}(m_{i,j})$$
$$\text{st. } \sum_{i,j,k} a_{i,j,k}(m_{i,j}) = b_k \text{ for all } k \text{ in a finite set}$$
$$\text{and for } M = (m_{i,j}) \in \mathbb{R}^{d\times d},\ M^t = M \text{ and } Spec_{\mathbb{C}}(M) \subset \mathbb{R}_+$$
$$\Leftrightarrow M \text{ a symmetric positive semidefinite matrix.}$$

One can solve a VP by solving the SDP with identical coefficients by recalling that on can diagonalise $M = P^{-1}DP$ with $P^{-1} = P^t$ and all $D_{ii} \geqslant 0$, so that for $V = \sqrt{D}P$, we obtain $M = V^tV$, and we can then set $v_i = V_{*i}$ to get the corresponding solution to the VP. This factorisation and the fact that a product V^tV is always symmetric positive semidefinite imply that infeasiblity and unboundedness also translate between programs. Cholesky factorisation computes V in $O(d^3)$ time, but requires taking square roots, so it's numerically inaccurate.

SDPs can be solved in with the ellipsoid method.

TO COMPLETE: Better treatment in the gupta o'donnell script, max cut fron Korte Vygen, and semidef stuff from Grötschel Lovasz Schrijver, for separation oracle.

The SDP relaxation of max-cut is $\max \frac{1}{2}\sum_{\{u,v\}\in V^2} w_{\{u,v\}}(1 - y_{uv})$ for spd matrix $Y = (y_{uv})_{V^2}$, where we've set $w_{\{u,v\}} = 0$ if $\{u, v\} \notin E$, and constraints $y_{vv} = 1$ for all $v \in V$ and $|y_{uv}| \leqslant 1$ for all $\{u, v\} \in V^2$, due to the variables representing dot-products of unit vectors. If we order the vertices from 1 to $|V|$, we

see that $Y = I + \sum_{1 \leqslant i < j \leqslant |V|} y_{ij}(E_{ij} + E_{ji})$ for canonic basis matrices (E_{ij}), so that the affine hull is $\binom{|V|}{2}$ dimensional. Separating hyperplanes in $|V|^2$ dimensional space become separating hyperplanes in $\binom{|V|}{2}$ dimensional by inserting $y_{vv} = 1$ and $y_{uv} = y_{vu}$.

We know that the feasible set is contained in the ball $B\left(0, \sqrt{\binom{|V|}{2}}\right)$ (in the affine hull), as $Y = I$ is feasible and $|y_{uv}| \leqslant 1$. Showing that 0 is an interior point of the feasible set is a bit harder.

Positive definiteness translates to $z^t Y z \geqslant 0$ for all $z \in \mathbb{R}^{|V|}$, which for or space means $z^t Y z = \sum_{1 \leqslant i,j \leqslant |V|} z_i y_{ij} z_j = \sum_{1 \leqslant i \leqslant |V|} z_i^2 + 2 \sum_{1 \leqslant i < j \leqslant |V|} z_i y_{ij} z_j \geqslant 0$. We want to guarantee this for a small neighbourhood of 0. We'll try to exploit $0 \leqslant (z_i - z_j)^2 = z_i^2 + z_j^2 - 2 z_i z_j$ to get this conclusion. To get this expression in a sum over $1 \leqslant i < j \leqslant |V|$, we must have $(|V| - 1)$ times the term z_i^2 (the number of pairs terms $z_i z_j$ containing z_i). Our goal then being to bound $z^t Y z = \frac{1}{(|V|-1)} \sum_{1 \leqslant i < j \leqslant |V|} \left(z_i^2 + z_j^2\right) + 2 \sum_{1 \leqslant i < j \leqslant |V|} z_i y_{ij} z_j =$, we could use $z_i y_{ij} z_j \geqslant \frac{-1}{(|V|-1)} |z_i z_j|$, which would allow for $z^t Y z \geqslant \frac{1}{(|V|-1)} \sum_{1 \leqslant i < j \leqslant |V|} \left(z_i^2 + z_j^2 - 2|z_i z_j|\right) = \frac{1}{(|V|-1)} \sum_{1 \leqslant i < j \leqslant |V|} (|z_i| - |z_j|)^2 \geqslant 0$, the desired result. This condition can be obtained by using the intermediate $z_i y_{ij} z_j \geqslant -|z_i y_{ij} z_j| \geqslant \frac{-1}{(|V|-1)} |z_i z_j|$ to get the equivalent condition $|y_{uv}| \leqslant \frac{1}{(|V|-1)}$.

To conclude, if $y \in B\left(0, \frac{1}{(|V|-1)}\right)$, then $|y_{uv}| \leqslant \frac{1}{(|V|-1)}$ (as $\|.\|_2 \geqslant \|.\|_\infty$), so in particular $|y_{uv}| \leqslant 1$ and Y is sdp by the previous paragraph. So $B\left(0, \frac{1}{(|V|-1)}\right)$ is contained in the feasible set.

21.2 The Lovász theta function

Gupta O'Donnell script, among others, like Gärtner Matousek

21.3 Hazan's algorithm

We'll now discuss an interior point method for solving SDP's of form $\max C \circ X$ st. X spd, $tr(X) = 1$, and $A_i \circ X \leqslant b_i$ for $i \in [m]$, where the A_i and C are sdp and we use notation $C \circ X = \sum_{i,j \in [n]} c_{ij} x_{ij}$.

It performs binary search on the objective values, solving feasibility problems at each step. We'll soon see that SPDs in this form are always bounded, with upper bound $\lambda_{max}(C)$ the largest eignevalue of C, which we can compute, for example with the power method. So binary serach can be performed on $[\lambda_{min}(C), \lambda_{max}(C)]$.

To solve feasibility problems of finding an X that is spd such that $tr(X) = 1$ and $A_i \circ X \leqslant b_i$ for $i \in [m]$, we'll use a barrier method. We define penalty $f(X) = \frac{1}{K} \ln \left(\sum_{i \in [m]}^{m} e^{K(A_i \circ X - b_i)} \right)$ for some $K > 0$ that we'll choose later, a try to solve $\min f(X)$ st. X spd and $tr(X) = 1$.

This objective is convex, as the composition of the convex $g(y_1, ..., y_m) = \frac{1}{K} \ln \left(\sum_{i \in [m]}^{m} e^{y_i} \right)$ with an affine map. To see that g is convex, we compute the Hessian with $\partial_{ij} g(y) = \frac{-e^{y_i + y_j}}{K \left(\sum_{k \in [m]}^{m} e^{y_k} \right)^2}$ for $i \neq j$ and $\partial_{ii} g(y) = \frac{e^{y_i} \left(\sum_{k \in [m]}^{m} e^{y_k} \right) - e^{2y_i}}{K \left(\sum_{k \in [m]}^{m} e^{y_k} \right)^2}$, where we see that semidefinite positiveness depends on the nominator.

To show that $\sum_i z_i^2 \left(\sum_{k \in [m] \setminus i}^{m} e^{y_k + y_i} \right) - \sum_{i \neq j} z_i z_j e^{y_i + y_j} \geqslant 0$, we write it as $\sum_{i \neq j} \left(z_i^2 e^{y_j + y_i} - z_i z_j e^{y_i + y_j} \right) \geqslant 0$ which is due to $z_i^2 e^{y_j + y_i} + z_j^2 e^{y_i + y_j} - 2 z_i z_j e^{y_i + y_j} \geqslant 0$ for all $i > j$, as can be seen from $e^{y_i + y_j} \geqslant 0$ and $z_i^2 + z_j^2 - 2 z_i z_j = (z_i - z_j)^2 \geqslant 0$.

The objective satisfies $\max_i (A_i \circ X - b_i) \leqslant f(X) \leqslant \max_i (A_i \circ X - b_i) + \frac{\ln(m)}{K}$ since $e^{K \max_i (A_i \circ X - b_i)} \leqslant \sum_{i \in [m]}^{m} e^{K(A_i \circ X - b_i)} \leqslant m e^{K \max_i (A_i \circ X - b_i)}$ by positivity and increase of e and ln, and the properties of the logarithm. So if we fix an error ε and set $K = \frac{\ln(m)}{\varepsilon}$, we have $\max_i (A_i \circ X - b_i) \leqslant f(X) \leqslant \max_i (A_i \circ X - b_i) + \varepsilon$. If $\min f(X) \leqslant \varepsilon$, then we know that X is such that $A_i \circ X \leqslant b_i + \varepsilon$ for all $i \in [m]$. If $\min f(X) > \varepsilon$, then for all spd X with $tr(X) = 1$, $f(X) > \varepsilon$, so that $\max_i (A_i \circ X - b_i) > 0$ so that in particular there is an index for which $A_i \circ X > b_i$, and we see that the initial SDP is infeasible. Thus, solving $\min f(X)$ st. X spd and $tr(X) = 1$ for our choice of K provides an almost feasible X or provides the certitude that the SDP is infeasible.

We will now explain how to solve $\min f(X)$ st. X spd and $tr(X) = 1$ with a conditional gradient or Frank-Wolfe type approach. Recall that these approaches compute a sequence of feasible solutions X_k, starting from $X_0 = E_{11}$, by approximating the objective by it's first-order approximation, which is $S \mapsto f(X_k) + \nabla f(X)^t (S - X_k)$ at each step, finding spd S_k with $tr(S_k) = 1$ that maximises it, and updating with $X_{k+1} = X_k + s_k(S_k - X_k)$, where $s_k \in [0, 1]$ is chosen in some way. The fact that the relation defining X_{k+1} is a convex combination on the convex spectahedron means that we do indeed get feasible points.

The first task in performing this algorithm is to compute $\nabla f(X)$.
We compute $\partial_{(ij)} f(X) = \frac{\sum_{k\in[m]}^{m} (A_k)_{ij} e^{K(A_k \circ X - b_k)}}{\sum_{k\in[m]}^{m} e^{K(A_k \circ X - b_k)}}$, which can be expressed as $\nabla f(X) = \sum_{k\in[m]}^{m} \frac{e^{K(A_k \circ X - b_k)}}{\sum_{k\in[m]}^{m} e^{K(A_k \circ X - b_k)}} A_i$ (by thinking of the gradient as a matrix, like for X). Note that this is a convex combination.

Next, we're faced with solving the subroutine of form $\max_{X\in Sp}(G \circ X)$, where we denote the spectahedron $Sp = \{X : X \ psd, tr(X) = 1\}$ and recall that G is psd, since $\nabla f(X)$ is, as a convex combination of the psd A_k. We'll show that this is equivalent to solving an eigenvalue problem. Indeed, a psd X can be diagonalized into $X = PDP^t$ by spectral decomposition, with $D \geqslant 0$ and P orthonormal. By decomposing $D = \sum_{i=1}^{n} D_{ii} E_{ii}$ and computing the matrix producs along the lines of $P \begin{pmatrix} 0 & & \\ & D_{ii} & \\ & & 0 \end{pmatrix} P^t = P \begin{pmatrix} & 0 & \\ & D_{ii}(P_{*i})^t & \\ & 0 & \end{pmatrix} = D_{ii} P_{*i} (P_{*i})^t$, we get $X = \sum_{i=1}^{n} D_{ii} P_{*i} (P_{*i})^t$. The condition $tr(X) = 1$, the trance properties, and $tr\left(P_{*i}(P_{*i})^t\right) = \|P_{*i}\|^2 = 1$ imply that $\sum_{i=1}^{n} D_{ii} = 1$. In parallel, rewriting the objective as $G \circ X = tr\left(G^t X\right)$, and using spectral decomposition $G = Q\Delta Q^t$, we have $tr\left(G^t X\right) = tr\left(Q\Delta^t Q^t P D P^t\right) = tr\left(\left(Q^t P\right)(\Delta D)\left(Q^t P\right)^t\right)$ by the trace properties. Now $\left(Q^t P\right)$ is also orthonormal and (ΔD) diagonal, so that by the same arguments as for X, we can write $G \circ X = \sum_{i=1}^{n} \Delta_{ii} D_{ii}$.

Thus, the problem reduces to finding the maximum $\sum_{i=1}^{n} \Delta_{ii} D_{ii}$ where the D_{ii} form points of the standard simplex. We therefore know the maximum to be $\max_i(\Delta_{ii})$. Recall that the Δ_{ii} where the eigenvalues of the psd G, so that $\max_i(\Delta_{ii}) = \lambda_{\max}(G)$. The maximum is attained by $X = xx^t$, for a unit eigenvector x of eigenvalue $\lambda_{\max}(G)$ of G, since $G \circ X = tr\left(G^t x x^t\right) = tr\left(x^t G^t x\right) = tr\left(\lambda_{\max}(G) x^t x\right) = \lambda_{\max}(G)$.

So the subroutine we have to carry out is finding the largest eigenvalue of a psd matrix. In this particular case, the eigenvalue problem can be solved approximately by the power method.
Note that it's this argument that allowed us to determine the bound of binary search.

All that is left to clarify is how to set the $s_k \in [0, 1]$. This is linked to the analysis of Hazan's algorithm, which we'll now start.
The first inequality need is estimating how good the linear approximation is: we'll show the bound on the iterations decrease in objective value $f(X_{k+1}) - f(X_k) \leqslant \nabla f(X_k)(s_k(S_k - X_k)) + s_k^2 K$.
The bound is shown by giving a Taylor series approximation of second order, with remainder, the remainder having form $\frac{1}{2} s_k^2 (S_k - X_k)^t \nabla^2 f(Y)(S_k - X_k)$ (where we identify the matrices to vectors), where Y is a convex combination of X_k and X_{k+1} an is therefore also in the spectahedron. this can be bounded by $\frac{1}{2} s_k^2 \lambda_{max}\left(\nabla^2 f(Y)\right) diam(Sp)^2$, where $diam(Sp)$ is the diameter of the spectahedron for the Frobenious/L^2-norm. We'll show that $diam(Sp) = \sqrt{2}$. We have for $A, B \in Sp$, $\|A - B\|_2^2 = tr\left((A - B)^t (A - B)\right) = tr\left(A^t A\right) - tr\left(A^t B\right) - tr\left(B^t A\right) + tr\left(B^t B\right)$. With the previous discussion and $tr\left(A^t A\right) \leqslant \max_{X\in Sp}(A \circ X)$, for example, we find that $\|A - B\|_2^2 \leqslant \lambda_{max}(A) - \lambda_{min}(B) - \lambda_{min}(A) + \lambda_{max}(B)$. Since eigenvalues of psd matrices are positive, and those of psd matrices of trance 1 must sum to 1 by spectral decomposition and trace invariance, we have $\|A - B\|_2^2 \leqslant 2$, so that applying the square root yields the bound. The bound is tight and therefore the diameter, as can be seen when A and B are different elementary matrices of

diagonal entry.

Hence the bound for our inequailty to be shown is now $s_k^2 \lambda_{max}\left(\nabla^2 f(Y)\right)$. To deal with $\lambda_{max}\left(\nabla^2 f(Y)\right)$, we'll first use bound $\lambda_{max}\left(\nabla^2 f(Y)\right) \leqslant tr\left(\nabla^2 f(Y)\right)$, which holds as $\nabla^2 f(Y)$ is psd by convexity of f. We'll now handle $\nabla^2 f(Y)$. Recall that $\partial_{(ij)} f(Y) = \dfrac{\sum_{k\in[m]}^m (A_k)_{ij} e^{K(A_k \circ Y - b_k)}}{\sum_{k\in[m]}^m e^{K(A_k \circ Y - b_k)}}$, so that to find $\partial_{(ij)(ij)} f(Y)$, we can look at $\partial_{(ij)(ij)} \left(\dfrac{e^{K(A_k \circ Y - b_k)}}{\sum_{k\in[m]}^m e^{K(A_k \circ Y - b_k)}} \right)(Y)$ only, by linearity. We then get $\partial_{(ij)(ij)} \left(\dfrac{e^{K(A_k \circ Y - b_k)}}{\sum_{k\in[m]}^m e^{K(A_k \circ Y - b_k)}} \right)(Y) = \dfrac{K(A_k)_{ij} e^{K(A_k \circ Y - b_k)} \left(\sum_{q\in[m]}^m e^{K(A_q \circ Y - b_q)}\right) - e^{K(A_k \circ Y - b_k)} \left(\sum_{q\in[m]}^m K(A_q)_{ij} e^{K(A_q \circ Y - b_q)}\right)}{\left(\sum_{k\in[m]}^m e^{K(A_k \circ Y - b_k)}\right)^2}$. We then see that when computing $tr\left(\nabla^2 f(Y)\right)$, we get sum $\dfrac{K \sum_{k\in[m]}^m \sum_{q\in[m]}^m (A_k)_{ij} ((A_k)_{ij} - (A_q)_{ij}) e^{K(A_k \circ Y - b_k + A_q \circ Y - b_q)}}{\sum_{k\in[m]}^m \sum_{q\in[m]}^m e^{K(A_k \circ Y - b_k + A_q \circ Y - b_q)}}$. This is a convex combination of numbers that can be bounded by $K 2 \max_{kij}(|(A_k)_{ij}|)$. Since we can scale the inequalities of the SDP so that $\max_{kij}(|(A_k)_{ij}|) \leqslant \dfrac{1}{2}$, we have bound $\lambda_{max}\left(\nabla^2 f(Y)\right) \leqslant K$.

We can finally complete the anaylsis. By denoting the optimum with X^*, and using convexity to get inequality $\nabla f(X_k)\left(X^* - X_k\right) \leqslant f\left(X^*\right) - f(X_k)$, as well as using $\nabla f(X_k)(s_k(S_k - X_k)) \leqslant s_k\left(\nabla f(X_k)\left(X^* - X_k\right) + \delta\right)$ where δ is the approximation error of the eigenvalue computation used to since S_k, we get $\nabla f(X_k)(s_k(S_k - X_k)) \leqslant s_k\left(f\left(X^*\right) - f(X_k)\right) + s_k\delta$. Finally with $f(X_{k+1}) - f(X_k) \leqslant \nabla f(X_k)(s_k(S_k - X_k)) + s_k^2 K$, we get $f(X_{k+1}) - f(X_k) \leqslant s_k\left(f\left(X^*\right) - f(X_k)\right) + s_k\delta + s_k^2 K$, rewritten as $f(X_{k+1}) \leqslant s_k f\left(X^*\right) + (1 - s_k) f(X_k) + s_k\delta + s_k^2 K$.

Many choice for s_k are now possible. We'll investigate $s_k = \min\left(1, \dfrac{2}{k}\right)$, for which the recursion becomes $f(X_{k+1}) \leqslant \dfrac{2}{k} f\left(X^*\right) + \left(1 - \dfrac{2}{k}\right) f(X_k) + \dfrac{2}{k}\delta + \dfrac{4}{k^2} K$ for $k \geqslant 2$. We'll show by induciton that $f(X_k) \leqslant f\left(X^*\right) + \delta + \dfrac{4K}{k}$. For $k = 1$, this is just $f(X_{k+1}) \leqslant s_k f\left(X^*\right) + (1 - s_k) f(X_k) + s_k\delta + s_k^2 K$ and $1 \leqslant 4$. For the step, note that $f(X_{k+1}) \leqslant \dfrac{2}{k} f\left(X^*\right) + \left(1 - \dfrac{2}{k}\right)\left(f\left(X^*\right) + \delta + \dfrac{4K}{k}\right) + \dfrac{2}{k}\delta + \dfrac{4}{k^2} K = f\left(X^*\right) + \delta + 4K\dfrac{k-1}{k^2} \leqslant f\left(X^*\right) + \delta + \dfrac{4K}{k+1}$, since $\dfrac{(k+1)(k-1)}{k^2} = \dfrac{k^2 - 1}{k^2} < 1$.

21.4 Solutions

22 Quadratic programming

22.1 Definitions and examples

Quadratic programm:

A **quadratic programm (QP)** is an optimization problem of form $\min \frac{1}{2}x^tQx + c^tx$ st. $Ax \leqslant b$, where Q is symmetric (and non-zero).

In general, we consider the case where Q is positive (semi-/) definite, so that this is a convex minimization problem.

For example, finding the distance between polyhedra can be formulated as a quadratic program, as minimising $\|x-y\|$ for $Ax \leqslant b$ and $Cy \leqslant d$ is equivalent to minimising (x^t, y^t) $\begin{pmatrix} 1 & & & -1 & & \\ & \ddots & & & \ddots & \\ & & 1 & & & -1 \\ -1 & & & 1 & & \\ & \ddots & & & \ddots & \\ & & -1 & & & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ st. $\begin{pmatrix} A & 0 \\ 0 & C \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leqslant \begin{pmatrix} b \\ d \end{pmatrix}$.

Another application is in the field of finance. We need to determine the fractions of capital x_i to be invested in project i, which has as return the random variable X_i, that allows for high and safe returns. The return is $E\left(\sum x_iX_i\right) = \sum x_im_i$, assuming $m_i = E(X_i)$ is known, for $\sum x_i = 1$ and $x \geqslant 0$. We could at this stage solve an LP, but we also want to take risk into account. The risk is given by the covariances $\sum_{ij} Cov(x_iX_i, x_jX_j) = x^tQx$ where Q is the covariance matrix of X. To see that Q is positive semidefinite, we write it as $Q = E\left((X-m)(X-m)^t\right)$ so that $x^tQx = E\left(\left(x^t(X-m)\right)^2\right) \geqslant 0$. We then look for the minimum risk portfolio with an arbitrarily set minimum expected return r by solving $\min x^tQx$ st. $\begin{cases} \sum x_im_i \geqslant r \\ \sum x_i = 1 \\ x \geqslant 0 \end{cases}$. Alternatively, we can set $s > 0$ and $t < 0$ to model the focus on either risk or return, and solve $\min(s.x^tQx + t.m^tx)$ st. $\begin{cases} \sum x_im_i \geqslant 0 \\ \sum x_i = 1 \\ x \geqslant 0 \end{cases}$.

22.2 Active set method

We assume that Q is sdp, which guarantees that he QP has a unique solution, if the polyhedron is non-empty. Indeed, the QP can't be unbounded in this case, as $\frac{1}{2}x^tQx + c^tx = \left(x^tQx\right)\left(\frac{1}{2} + \frac{c^tx}{x^tQx}\right)$, where $\frac{c^tx}{x^tQx} \xrightarrow[\infty]{\min(|x_i|)} 0$ (as can be seen from spectral decomposition), so that outside a large enough cube, $\left(x^tQx\right)\left(\frac{1}{2} + \frac{c^tx}{x^tQx}\right) > 0$, and hence $\frac{1}{2}x^tQx + c^tx$ is bounded from below.

The active set method requires us to solve $\min x^tQx + c^tx$ st. $Ax = b$ as subroutines, and this can be done almost analytically, like for the Newton method. We start by finding a base B of $\ker(A)$ and a particular solution x_0 to transform the problem to the unconstrained $\min y^t\left(B^tQB\right)y + \left(2x_0^tQB + c^tB\right)y$. In case of a zero kernel, the optimisation problem is trivially solved by 0. For positive semidefinite Q, B^tQB is positive semidefinite as well, so that this is a convex minimisation problem, since B^tQB is the Hessian. The first order condition is $\left(B^tQB\right)y = -\left(2x_0^tQB + c^tB\right)^t$ and solving this linear system (efficiently, since B^tQB is spd) provides the minima.
We note that if Q is positive definite, there is a unique solution y, which is non-zero if $c + 2Qx_0$ is. Another way of solving this subroutine is presented in the next paragraph.

Optimality conditions:
The active set method will also use optimality testing as subroutine. We can check if x is an optimal solution to the QP by looking for the existance of solutions to the KKT system $\begin{cases}\lambda \geqslant 0 \\ (A_{T*})^t\lambda + c + Qx = 0\end{cases}$ for λ, where T are the indices of rows of $Ax \leqslant b$ that are tight for x, which handles the complementary slackness condition (extend to the actual KKT system by setting the other multipliers to zero). This is an LP-feasibility problem. We can actually solve a QP by solving LPs of this kind, for all possibilities of T (or at least those that correspond to faces). If we don't know the tight inequalities of the optimal solution, and tr to solve the KKT directly, we're dealing with a system of quadratic in-/equation, due to complmentary slackness conditions.

For the subroutine of solving the equality constrained QP, the KKT system is simply $\begin{cases}Ax = b \\ A^t\mu + c + Qx = 0\end{cases}$, which can be written as $\begin{pmatrix}Q & A^t \\ A & 0\end{pmatrix}\begin{pmatrix}x \\ \mu\end{pmatrix} = \begin{pmatrix}-c \\ b\end{pmatrix}$, with a symmetric invertible $\begin{pmatrix}Q & A^t \\ A & 0\end{pmatrix}$ when Q is sdp and A has full rank, so that this system has a solution that solves the QP in this case. Indeed, $\begin{cases}Ax = 0 \\ A^t\mu + Qx = 0\end{cases} \Rightarrow \begin{cases}x^tA^t = 0, A^t\mu + Qx = 0 \\ x^tA^t\mu + x^tQx = 0\end{cases} \Leftrightarrow x, \mu = 0.$

The active set algorithm (without degeneracy):
Contrarily to linear programming, QPs can attain their minimum anywhere on the polyhedron. If the minimum is attained on the interior of a face (including the polyhedron itself), then we can solve the problem on the affine hull of the face by solving a linear first order condition and finding the part of the solutions that are inside the face/polyhedron, via an LP. However, if we where to try this on all faces, we would have to determine all faces first, which is quite inefficient.
A better idea is to iteratively alternate between looking for a minimum on the face one is currently considering, and changing face (the "active sets") in a clever way, depending on a method for telling if doing so will yield better results.

We start the algorithm with a initial point x_0 of the polyhedron, which can be found by solving an LP-feasibility problem. We consider the first active set I_0 to be the indices of rows for which $Ax_0 \leqslant b$

is tight. We ask if x_0 is a minimiser on the interior of the face that it's on, a type of point we'll call **quasistationary**, and if not, in which direction the minimiser is. To check this we note that the points of the affine hull of the face can be written as $x_0 + v$ where $A_{I*}v = 0$. The objective can be written as $\frac{1}{2}(x_0+v)^tQ(x_0+v) + c^t(x_0+v)$, who's minimisation over $A_{I*}v = 0$ is equivalent to that of $\frac{1}{2}v^tQv + \left(x_0^tQ + c^t\right)v$ on $A_{I*}v = 0$ as constant terms don't matter in the objective. We solve this as described previously.

Two cases are of interest: the one where the optimum is $v = 0$, so that x_0 is quasistationary, and that where $v \neq 0$, that is that we can do better, even on that face. We will investigate the second case first. In this case, we disjoin two further cases: either $x_0 + v$ is in the interior of the face (check $A_{[m]\backslash I_0*}(x_0+v) < b_{[m]\backslash I_0}$), or it isn't. In the case that it is, the point $x_1 = x + v$ is quasistationary, and has strictly lower objective value, by strict convexity of the objective for Q sdp. If it isn't, then we will follow the direction of v from x_0 in a strate until we reach the boundary of the face: the point at which we stop may not be the optimum on the face, but it should be close. We do this by looking at $x_0 + tv$ for $t \in [0,1[$. The farthest we can go so that $A(x_0+tv) \leqslant b$ is $t_0 = \min\limits_{i\in[m]\backslash I_0}\left(\frac{b_i - A_{i*}x_0}{A_{i*}v} : A_{i*}v > 0\right)$. There has to be an index $i \in [m]\backslash I_0$ such that $A_{i*}v > 0$, because otherwise $x_0 + tv$ for all $t > 0$ and hence $t = 1$ is in the polyhedron, the case we've disjoined and excluded here. Note also that $t_0 > 0$ as $b_i - A_{i*}x_0$ for $i \in [m]\backslash I_0$. The point $x_0 + t_0v$ has therefore strictly lower objective value, by strict convexity of the objective for Q sdp. We note that for an index $j \in [m]\backslash I_0$ achieving $\min\limits_{i\in[m]\backslash I_0}\left(\frac{b_i - A_{i*}x_0}{A_{i*}v} : A_{i*}v > 0\right)$, we have $A_{j*}(x_0 + t_0v) = b_j$. Hence if we consider the set of active/tight inequalities on $x_0 + t_0v$, it contains I_0 and the new j. If we set $x_1 = x_0 + t_0v$ and we repeat the full process we started on x_0 from here, defining x_2 in case x_1, turns out to not be quasistationary on the face given by $I_0 \cup j$, in the same way, and so on, then the set of active inequalities will grow and grow, and we'll be considering faces of decreasing dimension, until we end up the a quasistationary x_i in iteration i, possibly after m iterations, for in that case the interior is the space.

Hence in at most m iterations of the case $v \neq 0$, we end up with a quasistationary point x_i. The case $v = 0$ meant that x_0 was quasistationary. So in all cases we end up at a quasistaionary point. We then test if this point is actually a minimum on all of the polyhedron with the KKT conditions, which are $\begin{cases}\lambda \geqslant 0\\ (A_{I_i*})^t\lambda + c + Qx_i = 0\end{cases}$, where I_i is the set of tight inequalities on x_i.
Instead of solving this LP-feasibillity problem we can observe the following. If x_i is a non-degenerate (the inequalities it's tight at are independent) quasistationary point, that is, it's the solution to $\min \frac{1}{2}x^tQx + c^tx$ st. $\begin{cases}A_{I_i*}x = b_I\\ A_{[m]\backslash I_i*}x \leqslant b_{[m]\backslash I_i}\end{cases}$ and verifies $A_{[m]\backslash I_i*}x < b_{[m]\backslash I_i}$, where A_{I_i*} has full rank (non-degenerate), then Slater's condition applies and we're guaranteed a solution to the KKT system, which here resumes to $(A_{I_i*})^t\mu + c + Qx_i = 0$. This is actually a direct consequence of A_{I_i*} having full rank. So a way of checking global optimaility of x_i is by solving $(A_{I_i*})^t\mu + c + Qx_i = 0$, and checking if the unique solution $\mu \geqslant 0$, which implies global optimality. If there is some j with $\mu_j < 0$, then by uniqueness of the solution μ, we know that the global KKT system can't have a feasible solution, hence x_i isn't optimal.
For a geometric viewpoint, recall that $c + Qx_i = 0$ is the gradient of the objective at x_i and $(A_{I_i*})^t\mu$ for $\mu \geqslant 0$ is the normal cone of the face.

The case of a quasistationary point that isn't a global optimum point can happen. For example, consider the problem of minimisng the distance to $(0,2)$ on the unit cube, and we start at $\left(\frac{1}{2}, -1\right)$. Since

we sarted on the bottom face, we look for the minimum on that face, which is the quasistationary $(0,-1)$. To get to the actual optimum $(0,1)$, we have to leave the bottom face.

The $\mu_j < 0$ of $(A_{I_i*})^t\mu + c + Qx_i = 0$ indicates that we should move in the opposite direction and leave the face by discarding the condition $A_{j*}(x_0 + v) = b_j$. To do this we seek a direction d that should lower the objective if we persue it a bit, which in terms of gradients means $\left(x_i^tQ + c^t\right)d < 0$. We have $c + Qx_i = -(A_{I_i*})^t\mu$. Since the $(A_{I_i*})^t$ are independent (non-degeneracy), those of $I_i\backslash j$ can't span all of space, and we can project $(A_{j*})^t$ onto $span\left((A_{I_i\backslash j*})^t\right)^\perp$ getting the vector $a \in \ker(A_{I_i\backslash j*})$ with $a^t(A_{j*})^t > 0$. Then $(c + Qx_i)^t a = -\mu^t A_{I_i*}a = -\mu_j A_{j*}a > 0$, so that $d = -a$ is a good candidate. We next look if this direction yields feasible points by investigating $A(x_i + td) \leqslant b$ for $t > 0$. On the rows of $I_i\backslash j$, since $d \in \ker(A_{I_i\backslash j*})$ and I_i are the tight rows for x_i, no problems arise. Since $A_{j*}d < 0$ and the j'th inequality is tight at x_i, we have $A_{j*}(x_i + td) \leqslant b_j$ for $t \geqslant 0$. Finally, for the rows of $[m]\backslash I_i$, we have $A_{[m]\backslash I_i*}x_i < b_{[m]\backslash I_i}$: so even if there is a $k \in [m]\backslash I_i$ with $A_{k*}d > 0$, we can find a small enough t such that $(A_{[m]\backslash I_i*}(x_i + td) \leqslant b_{[m]\backslash I_i}$ by a similar argument as we've seen for taking the direction of the quasistationary point on a face that doesn't contain it. So we can move a bit in the direction of d from x_i and stay in the polyhedron. We'll now formally prove that the direction we've chosen really does decrease the objective.
Indeed, for $f(x_i + td) = \frac{1}{2}(x_i + td)^tQ(x_i + td) + c^t(x_i + td) = t^2\frac{1}{2}d^tQd + t\left(x_0^tQ + c^t\right)d + f(x_i)$, which is decreasing for $t \in \left[0, \frac{-\left(x_0^tQ + c^t\right)d}{d^tQd}\right]$ (since $\left(x_i^tQ + c^t\right)d < 0$, as we conjectured). Hence, for a positive t in both ranges, we can move to $x_{i+1} = x_i + td$ to decrease the objective value strictly.

We just devellopped a method that, given a non optimal quasisationary point, contructs one of lower objective value. Once we've arrived at this point, we reiterate the algorithm, starting with that x_{i+1} as initial point. This concludes the description of the active set method.
To see that the algorithm terminates, note that after at most m successive iterations, we encounter a quasistationary point of strictly lower objective value. Hence quasistationary points may not repeatedly appear under consideration during the algorithm, as this would contradict the strict decreases. By strict convexity, a face may contain at most one quasistationary point per face, if any. So since there are finitely many faces, there are finitely many quasistactionary points, and the algorithm terminates. Since the algorithm terminates only when the quasisationary point is optimal, the algorithm does indeed produce the optimal solution.

Degeneracy:

22.3 Solutions

23 Multiplicative weights method

23.1 Multiplicative weights algorithms

The multiplicative weight algorithm originates from decision theory. We're in the following situation: we have to make a sequence of T yes/no decisions on a topic we know nothing about, and have access to the advice of n "experts" (which may more or less frequently give false advice). The problem consists in making decisions given the experts advice, such as to get the best results. On the course of the process, we can observe if our decisions are correct or not, and adapt our credibility of the experts at each step. Since we have no clue on the topics we decide on, our measure of how good our decisions were will be a comparison to the best experts number of incorrect decisions (the least number of incorrect decisions over all experts).

For example, we can introduce the following procedure. We attribute to each expert i a credibility weight $w_i(t)$, for decision t, that we initialise at $w_i(t) = 1$. To make decision t, we pick the yes/no advise that is in a weighted majority (with yes at ties) with the experts. For example, if $I \subset [n]$ are the experts advising "yes", then we play "yes" if $\sum_{i\in I} w_i(t) \geqslant \frac{1}{2}\sum_{i\in[n]} w_i(t)$. We then observe if the decision was correct or not and update the credibility of experts as follows: if i was correct, $w_i(t+1) = w_i(t)$ and otherwise $w_i(t+1) = \frac{w_i(t)}{1+\varepsilon}$, for a fixed parameter $\varepsilon > 0$.
We'll now analyse the procedure. We'll keep track of $\phi(t) = \sum_{i\in[n]} w_i(t)$ (sometimes called a potential). If I is the set of experts that was wrong for decision t, then $\phi(t+1) = \frac{1}{2}\sum_{i\in I} w_i(t) + \sum_{i\in[n]\setminus I} w_i(t)$. To relate this to $\phi(t)$, we note that $\phi(t+1) = \phi(t) - \frac{1}{2}\sum_{i\in I} w_i(t)$. In the case that we the weighted majority was wrong, so $\sum_{i\in I} w_i(t) \geqslant \frac{1}{2}\sum_{i\in[n]} w_i(t)$, we then get that $\phi(t+1) \leqslant \frac{3}{4}\phi(t)$. In general, we have $\phi(t+1) \leqslant \phi(t)$, as credibility can only decrease. So if M deontes the number of mistakes made with our procedure, then $\phi(T+1) \leqslant \left(\frac{3}{4}\right)^M \phi(1) = n\left(\frac{3}{4}\right)^M$. On the other hand, $\phi(t) \geqslant w_i(t)$ for all experts by positivity, and if we denote by m_i the number of misteks in T decisions made by expert i, then we have $w_i(t) = \left(\frac{1}{1+\varepsilon}\right)^{m_i}$. Combining the bounds yields $\left(\frac{1}{1+\varepsilon}\right)^{m_i} \leqslant n\left(\frac{3}{4}\right)^M$, which reformulates to $M \leqslant \frac{1}{\ln(4/3)}(\ln(n) + \ln(1+\varepsilon)m_i)$. This is the type of result we desire in decision theory, since in the bound i was arbitrary, so we may take it for the expert with the least mistakes. Thus, for n experts, a small ε and a number of decisions such that $T \geqslant \frac{\ln(n)}{\ln(4/3)}$, the average number of mistakes $\frac{M}{T}$ will at most one more a small fraction of that of the best expert.

Multiplicative weight algorithms:
The multiplicative weights algorithms solve the same type of problems, but slightly differently. Now, we consider expert advice to not be comparable. Thus, a first difference in the context is that following an expert i's advice, which is the typeof action we're allowed to take, will incure a cost $m_i \in [-1, 1]$, where negative costs are gains.
Our new strategy is the following. At decision t, we pick a probability distribution $p_i(t)$ over the ex-

perts i with $p_i(t) = \dfrac{w_i(t)}{\phi(t)}$, sample an expert with it and follow its advice. We'll update credibility with the rule $w_i(t+1) = w_i(t)(1-\varepsilon)^{m_i(t)}$ if $m_i(t) \geqslant 0$ (costs in total) and $w_i(t+1) = w_i(t)(1+\varepsilon)^{-m_i(t)}$ if $m_i(t) < 0$ (gains in total), where $m_i(t)$ is the cost of expert i, if we played according to i's advice at stage t.

To analyse the new strategy, we again track the potential, and hope to bound the avergae cost $p(t)^t m(t)$ at decision t. Now $\phi(t+1) = \sum_{m_i(t)\geqslant 0} w_i(t)(1-\varepsilon)^{m_i(t)} + \sum_{m_i(t)<0} w_i(t)(1+\varepsilon)^{-m_i(t)}$. The updates have the property that $(1\pm\varepsilon)^{\mp x} \leqslant (1-\varepsilon x)$ for $x \in [-1,1]$. To see this, for $x \geqslant 0$, we can use convexity of positive exponentials to get $(1-\varepsilon)^x \leqslant (1-\varepsilon)$ and $x \leqslant 1$ and $\varepsilon > 0$ to conclude, and for $x \leqslant 0$, we can use convexity of positive exponentials to get $(1+\varepsilon)^{-x} = \left(\dfrac{1}{1+\varepsilon}\right)^x \leqslant 1 \leqslant 1-\varepsilon x$. This allows us to get $\phi(t+1) \leqslant \sum_{i\in[n]} w_i(t)(1-m_i(t)\varepsilon))$ since weights remain positive. This in turn gives $\phi(t+1) \leqslant \phi(t)\left(1-\varepsilon p(t)^t m(t)\right)$, as $w_i(t) = p_i(t)\phi(t)$. With the good old inequality $e^x \geqslant 1+x$ we get $\phi(t+1) \leqslant \phi(t) e^{-\varepsilon p(t)^t m(t)}$. Therefore, in the end we have $\phi(T+1) \leqslant n e^{-\varepsilon \sum_{t\leqslant T} p(t)^t m(t)}$.

Imitating the previous algorithm, we note that $\phi(T+1) \geqslant w_i(T+1) = (1-\varepsilon)^{\sum_{m_i(t)\geqslant 0} m_i(t)}(1+\varepsilon)^{-\sum_{m_i(t)<0} m_i(t)}$, so that combining bounds rewrites to $\ln(n) - \varepsilon \sum_{t\leqslant T} p(t)^t m(t) \geqslant \sum_{m_i(t)\geqslant 0} m_i(t)\ln(1-\varepsilon) + \sum_{m_i(t)<0} -m_i(t)\ln(1+\varepsilon)$. With more clever inequalities, we can maque that inequality more digestible. With $\ln(1+x) \geqslant x - x^2$ on $\left[-\dfrac{1}{2}, \dfrac{1}{2}\right]$, we have for $\dfrac{1}{2} \geqslant \varepsilon > 0$ also $\ln(1-\varepsilon) \geqslant -\varepsilon - \varepsilon^2$. Then $\sum_{m_i(t)\geqslant 0} m_i(t)\ln(1-\varepsilon) + \sum_{m_i(t)<0} -m_i(t)\ln(1+\varepsilon) \geqslant -\varepsilon \sum_{t\leqslant T} m_i(t) - \varepsilon^2 \sum_{t\leqslant T} |m_i(t)|$.

Finally, we get the nice bound on the total expected cost $\sum_{t\leqslant T} p(t)^t m(t) \leqslant \sum_{t\leqslant T} m_i(t) + \varepsilon \sum_{t\leqslant T} |m_i(t)| + \dfrac{\ln(n)}{\varepsilon}$.

We can get even smoother by minimising the second part in ε: for n large enough so that $\varepsilon = \sqrt{\dfrac{\ln(n)}{n}} \leqslant \dfrac{1}{2}$, we have $\sum_{t\leqslant T} p(t)^t m(t) \leqslant \sum_{t\leqslant T} m_i(t) + (n\sqrt{\dfrac{\ln(n)}{n}} + \dfrac{\ln(n)}{\sqrt{\frac{\ln(n)}{n}}}$.

INCLUDE: Update rule from Gupta O'Donnell as exercise ?

23.2 Solving particular LPs and SDPs with multiplicative weights

LPs, like minimum set cover:
The context is that of determining if intersections of polyhedra are empty or not, where one of the polyhedra has a particular form that allows us to find point in it reasonably fast. If P is the "easy" polyhedron, we seek to find a point $x \in P$ such that $Ax \geqslant b$, or decide that none exists. We're actually in an even more restrained context, where we assume that we can easily decide if a plane $\left(p^tA\right)x < \left(p^tb\right)$ separates P from $Ax \geqslant b$. To be precise, we assume that an oracle tells us, given p, if $\left(p^tA\right)x < \left(p^tb\right)$ on all of P, or finds an element x in P such that $\left(p^tA\right)x \geqslant \left(p^tb\right)$.

This is the case for the LP-relaxation of the set cover problem. If we want to test if the k among m sets $S_i \subseteq [n]$ can cover the elements of $[n]$, then we want to find a binary vector x such that $1^tx = k$ and $\sum_{i:j\in S_i} x_i \geqslant 1$ for all $j \in [n]$. In the LP-relaxation where the binary condition becomes $x \geqslant 0$, we can let P be the simplex $1^tx = k$, $x \geqslant 0$ and $Ax \geqslant b$ be the constraints $\sum_{i:j\in S_i} x_i \geqslant 1$ for all $j \in [n]$. For p, we can decide if $\left(p^tA\right)x < \left(p^tb\right)$ on all of P directly by testing if $k.\max_j\left(\left(p^tA\right)_j\right) < \left(p^tb\right)$ (we solve an LP on the standard simplex): if not, then we output $x = ke_j$ for the index j attaining the previous maximum. This is our oracle, which only required a comparison and the computation of a maximum.

With such an oracle, how do we find a point $x \in P$ such that $Ax \geqslant b$ or decide that none exists ?
The relation to the multiplicative weights algorithm is that we can consider the constraints $\sum_{i:j\in S_i} x_i \geqslant 1$ for all $j \in [n]$ to be the experts, in the sense that their cost is $m_j = \frac{1}{k-1}\left(\sum_{i:j\in S_i} x_i - 1\right)$, where the $\frac{1}{k-1}$ is for normalisation, as for $k \geqslant 2$, $\frac{1}{k-1}\left(\sum_{i:j\in S_i} x_i - 1\right) \in [-1,1]$, for the x produced by the oracle for the set cover problem we just discussed. We then see that $p^t(Ax-b) = p^tm$.
If we use the multiplicative weights algorithm, where at each step t we compute $x(t)$ as a point in P such that $\left(p(t)^tA\right)x(t) \geqslant \left(p(t)^tb\right)$ (or find a p that separates the polyhedra), and define costs as we just decribed, then we have $0 \leqslant p(t)^tm(t)$, so that by the bound from multiplicative weights, we now have $0 \leqslant \sum_{t\leqslant T}\frac{1}{k-1}(Ax(t)-b)_i + \varepsilon\sum_{t\leqslant T}\frac{1}{k-1}|(Ax(t)-b)_i| + \frac{\ln(n)}{\varepsilon}$.
By arraging sums by introducing $\pm\sum_{t\leqslant T,(Ax(t)-b)_i<0}\frac{1}{k-1}(Ax(t)-b)_i$, we get $0 \leqslant (1+\varepsilon)\sum_{t\leqslant T}\frac{1}{k-1}(Ax(t)-b)_i + 2\varepsilon\sum_{t\leqslant T,(Ax(t)-b)_i<0}\frac{1}{k-1}|(Ax(t)-b)_i| + \frac{\ln(n)}{\varepsilon}$, and finally $0 \leqslant (1+\varepsilon)\sum_{t\leqslant T}\frac{1}{k-1}(Ax(t)-b)_i + 2\varepsilon T + \frac{\ln(n)}{\varepsilon}$.
We can rewrite this introducing $x^* = \frac{1}{T}\sum_{t\leqslant T}x(t)$ which is in P by convexity, to get $0 \leqslant (Ax^*-b)_i + \frac{(k-1)}{(1+\varepsilon)}\left(2\varepsilon + \frac{\ln(n)}{\varepsilon T}\right)$, so that x^* is approximately feasible, since $Ax \geqslant b-\delta$ for $\delta = \frac{(k-1)}{(1+\varepsilon)}\left(2\varepsilon + \frac{\ln(n)}{\varepsilon T}\right)\cdot 1$.
This term can be made small by making ε small and T large.

Returning to the case of set cover, the method we described allows us to find an almost feasible point for the LP-relaxation of the set cover feasibility problem for k sets, or determine that none exists. In this case, we can actually consider $\overline{x} = \frac{1}{1-\delta}x^*$ with $\delta < 1$, which is a feasible point for the minimum set cover

LP-relaxation, with objective value $\frac{k}{1-\delta}$, which is almost k for small δ.

SDPs, like maximum cut:
Recall the SDP-relaxation of the max cut problem, where we minimise a linear function on the coeficients of a psd X, subject to constraints $x_{ii} = 1$.

We make use of the fact that one can easily answer questions of type: is there a psd X of trace 1 that satisfies $\sum_{ij} c_{ij}x_{ij} \geqslant q$, for some q and a psd C ? We use this in our oracle, as we've done with LPs. It turns out that asking $\sum_{ij} c_{ij}x_{ij} \geqslant q$ is equivalent to finding the largest eigenvalue of C: with spectral decomposition, we have $C = P^tDP$ for a diagonal $D \geqslant 0$, who's diagonal entries are the eigenvalues of C, and by setting $Y = PXP^t$, and by writing $\sum_{ij} c_{ij}x_{ij} = tr\left(C^tX\right)$ to ease computations, we have $tr\left(C^tX\right) = \left(P^tDPP^tYP\right) = tr(DY)$. Since also $tr(Y) = tr(X) = 1$, we have $tr(DY) = \sum_{ii} d_{ii}y_{ii} \leqslant \max(d_{ii})$, by positivity and $\sum_{ii} y_{ii} = 1$. This inequality is tight: if v is the unit eigenvector to the maximum eigenvalue of C (aka $\max(d_{ii})$), then for $X = vv^t$, we have $tr\left(C^tX\right) = tr\left(Cvv^t\right) = \max(d_{ii})\|v\|^2 = \max(d_{ii})$.
So $\sum_{ij} c_{ij}x_{ij} \geqslant q$ precisely the case when $\max(d_{ii}) \geqslant q$: if it is the $X = vv^t$ is a candidate, otherwise, $tr\left(C^tX\right) \leqslant \max(d_{ii})$ implies impossibility of a solution.

INCLUDE: power method for computing largest eigenvalues...
FINISH: from LP SDP Gupta script

23.3 Solutions

24 Non-convex optimization

Non-convex optimization is a much more difficult task then convex optmization. This is because in convex optimization, local minima are by deffinition global ones, so that local improvements always lead to the best solution. This isn't the case in non-convex optimization, in which local information is nothing but local. here, local optima may not be global, and worse first order local approximations are unreliable, so that saddle points turn up.

24.1 Gradient methods

In non-convex optimization, it may be hard to do the best, but it's still easy to do better.
In a differentiable setting, we can still perform gradient descent with constant step size s, where we update with $x_{k+1} = x_k - s\nabla f(x_k)$. We still have, assuming L-smoothness, or equivalently an L-Lipschitz gradient, or a bounded Hessian, the bound $f(x+d) \leqslant f(x) + \nabla f(x)^t d + \frac{1}{2}L\|d\|^2$. So we have $f(x_{k+1}) \leqslant f(x_k) - s\|\nabla f(x_k)\|^2 + \frac{1}{2}Ls^2\|\nabla f(x_k)\|^2$, or equivalently $\left(s - \frac{1}{2}Ls^2\right)\|\nabla f(x_k)\|^2 \leqslant f(x_k) - f(x_{k+1})$. Assuming that $0 < s < \frac{2}{L}$, which in practice means that we seek small s as L is generally unknown, we have $\left(s - \frac{1}{2}Ls^2\right) > 0$. This implies that $0 \leqslant f(x_k) - f(x_{k+1})$, so that this is a descent method. We could also perfom this step with backtracking line search and a variable step length at each iteration s_k.
What is missing in the non-convex case is the strong convexity, aka. a lower bounded Hessian, which drove most of the key steps in the analysis in the convex case.
Instead, we can make the following observation: we have $\left(s - \frac{1}{2}Ls^2\right)\sum_{i=0}^{k-1}\|\nabla f(x_i)\|^2 \leqslant f(x_0) - f(x_k)$.
This can still be exploited with $\min_{i\leqslant k-1}\left(\|\nabla f(x_i)\|^2\right) \leqslant \frac{1}{k}\sum_{i=0}^{k-1}\|\nabla f(x_i)\|^2$, in the form of $\min_{i\leqslant k-1}\left(\|\nabla f(x_i)\|^2\right) \leqslant \frac{f(x_0) - f(x_k)}{k\left(s - \frac{1}{2}Ls^2\right)}$. In the case that f is lower boundable by some m, we get $\min_{i\leqslant k-1}\left(\|\nabla f(x_i)\|^2\right) \leqslant \frac{f(x_0) - m}{k\left(s - \frac{1}{2}Ls^2\right)}$ and we see that the smallest gradient encountered along the descent will eventually become smaller. So we'll get closer to critical points, which may be saddle points, instead of local minima.
We can actually get better results for adapting steps s_k, with a closer analysis. The starting point of this version of gradient descent is that in $\left(s_k - \frac{L}{2}s_k^2\right)\|\nabla f(x_k)\|^2 \leqslant f(x_k) - f(x_{k+1})$, for $0 < s_k < \frac{2}{L}$, we have $f(x_k) \geqslant f(x_{k+1})$, and so for m lower-bounding f and $\delta_k = f(x_k) - m$ we have $\delta_k \geqslant \delta_{k+1} \geqslant 0$, and by writing $\mu_k = \left(s_k - \frac{L}{2}s_k^2\right) > 0$, we have $\mu_k\|\nabla f(x_k)\|^2 \leqslant f(x_k) - f(x_{k+1}) = \delta_k - \delta_{k+1}$. Summing on the steps as before yields $\sum_{k=0}^{K-1}\|\nabla f(x_k)\|^2 \leqslant \sum_{k=0}^{K-1}\frac{1}{\mu_k}(\delta_k - \delta_{k+1}) = \frac{\delta_0}{\mu_0} + \sum_{k=1}^{K-1}\delta_k\left(\frac{1}{\mu_k} - \frac{1}{\mu_{k-1}}\right) - \frac{\delta_K}{\mu_K}$ by a split of the sum and an index change. Now by dropping the last term in the last bound, because it's negative, and using the decrease of positive δ_k, we get $\sum_{k=0}^{K-1}\|\nabla f(x_k)\|^2 \leqslant \left(\frac{1}{\mu_0} + \sum_{k=1}^{K-1}\left(\frac{1}{\mu_k} - \frac{1}{\mu_{k-1}}\right)\right)\delta_0 = \frac{\delta_0}{\mu_{K-1}}$, so a bound of $\frac{\delta_0}{\mu_{K-1}}$. However, we can get a better bound with a clever update. If we manage to set $\mu_{K-1} = \frac{C}{\sqrt{\sum_{k=0}^{K-1}\|\nabla f(x_k)\|^2}}$ for a choice of s_{K-1} and a constant C, then $\sum_{k=0}^{K-1}\|\nabla f(x_k)\|^2 \leqslant \frac{\delta_0}{\mu_{K-1}}$ becomes

$\sqrt{\sum_{k=0}^{K-1} \|\nabla f(x_k)\|^2} \leqslant \frac{\delta_0}{C}$. We then have $\min_{k \leqslant K-1}(\|\nabla f(x_k)\|) \leqslant \frac{\delta_0}{KC}$. This would be a better bound in the sense that it doesn't require use to know L. We can therefore iterate for $K = \left\lceil \frac{\delta_0}{\varepsilon C} \right\rceil$ so that we have $\min_{k \leqslant K-1}(\|\nabla f(x_k)\|) \leqslant \varepsilon$, which we can determine if we know m and C only.

CONCLUDE: use $\left(s_k - \frac{L}{2}s_k^2\right) \leqslant \frac{s_k}{2}$ on $0 < s_k < \frac{1}{L}$ to simplify setting μ

Stochastic gradient descent:

In the chapter on neural networks, we'll encounter optimization problems in which the objective is an increadibly large sum of similar objectives, such as $f(x) = \frac{1}{|F|}\sum_{i \in F} g_i(x)$ where F is large, for example F is the size of a data set. To handle this problem, an idea is to sequentially partially minimise one of the g_i, hoping that by similarity of the g_i, we'll get many of them, hence also f, to take smaller values.

In fact, we will proceed at follows. We choose a probability distribution on F, and at each step. We select am $i \in F$ according to this distribution, and update $x_{k+1} = x_k - s\nabla g_i(x_k)$, a gradient descent update for step s and gradient $\nabla g_i(x_k)$.

To analyse this approach, we consider $f(x_{k+1}) \leqslant f(x_k) - s\nabla f(x_k)^t \nabla g_i(x_k) + \frac{L}{2}s^2\|\nabla g_i(x_k)\|^2$, assuming f to be l-smooth, so that by linearity and monotony of expectation $Ef(x_{k+1}) \leqslant Ef(x_k) - s\nabla f(x_k)^t E(\nabla g_i(x_k)) + \frac{L}{2}s^2 E\|\nabla g_i(x_k)\|^2$. The similarity assumption on the sum that is the objective will be that $E(\nabla g_i(x_k)) = \nabla f(x_k)$, so that we expect the average gradient of a term to be that of the sum. To handle $E\|\nabla g_i(x_k)\|^2$, we will assume that we know a bound $\|\nabla g_i(x_k)\| \leqslant B$ for all $i \in F$. So we have $Ef(x_{k+1}) \leqslant Ef(x_k) - s\|\nabla f(x_k)\|^2 + \frac{L}{2}s^2B^2$. Rephrasing to $\|\nabla f(x_k)\|^2 \leqslant \frac{1}{s}(Ef(x_k) - Ef(x_{k+1})) + \frac{L}{2}sB^2$, we have with the same arguments as before $\min_{k \leqslant K-1}\left(\|\nabla f(x_k)\|^2\right) \leqslant \frac{1}{Ks}(f(x_0) - m) + \frac{L}{2}sB^2$. So if we do $K-1$ iterations with $s = \frac{1}{\sqrt{K}}$, we'll know that $\min_{k \leqslant K-1}\left(\|\nabla f(x_k)\|^2\right)$ is in $O\left(\frac{1}{\sqrt{K}}\right)$.

Some remarks are in order. $E(\nabla g_i(x_k)) = \nabla f(x_k)$ doesn't imply $E\|\nabla g_i(x_k)\|^2 = \|\nabla f(x_k)\|^2$: consider the case of uniform distribution on $\pm v$ for some $v \neq 0$, then $E(\nabla g_i(x_k)) = \frac{1}{2}(v - v) = 0$ but $E\|\nabla g_i(x_k)\|^2 = \|v\|^2 \neq 0$.

24.2 Indefinite quadratic programming

With "InteriorPtsMethodBook" by Ye, move to QP chapter ?

24.3 Sequential convex programming

24.4 Low rank matrix recovery

24.5 Solutions

25 Algebraic methods for interger optimization

25.1 More integer optimization problems:

Clustering (minimum variance):
We consider a set of n data-points on q criteria indexed as $p_i \in \mathbb{R}^q$. We which to partition the data-points into k clusters of equal size, so that for clusters $C_j \subseteq [n]$ for $j \in [k]$ and for a distance d, the total aggregate distance of the points to their cluster average is minimum. This means that we which to minimise $\sum_{j\in[k]} \sum_{i\in C_j} d\left(p_i, \frac{1}{|C_j|} \sum_{s\in C_j} p_s\right)$. We'll fix the size of the clusters as $\frac{n}{k}$, which we assume to be integer.
A first step is to model this as a MINLP (depending on d, as we'll see) with the use of indicator variables $x_{i,j} \in \{0,1\}$. We let $x_{i,j} = 1$ if $i \in C_j$ and $x_{i,j} = 0$ otherwise. For only a single attribution to be possible, we add constraint $\sum_{j\in[k]} x_{i,j} = 1$. We also add the constraint on the cluster size $\sum_{i\in[n]} x_{i,j} = \frac{n}{k}$. The objective becomes $\sum_{j\in[k]} \sum_{i\in[n]} x_{i,j} d\left(p_i, \frac{k}{n} \sum_{s\in[n]} x_{s,j} p_s\right)$.
We can treat two cases of d as MINLPs. For the norm $\|.\|_1$, we can introduce variables $m_{i,d} \in \mathbb{R}$ and constraints $m_{i,d} \geqslant p_{i,d} - \frac{k}{n} \sum_{s\in[n]} x_{s,j} p_{s,d}$ and $m_{i,d} \geqslant -p_{i,d} + \frac{k}{n} \sum_{s\in[n]} x_{s,j} p_{s,d}$ for all $i \in [n]$, $j \in [k]$ and $d \in [q]$. We note that $m_{i,d} \geqslant \left| p_{i,d} - \frac{k}{n} \sum_{s\in[n]} x_{s,j} p_{s,d} \right|$ and therefore $\sum_{d\in[q]} m_{i,d} \geqslant \left\| p_i - \frac{k}{n} \sum_{s\in[n]} x_{s,j} p_s \right\|_1$.
By setting the objective to $\sum_{j\in[k]} \sum_{i\in[n]} x_{i,j} \sum_{d\in[q]} m_{i,d}$, we note that at a minimum solution, we must have all $m_{i,d} = \left| p_{i,d} - \frac{k}{n} \sum_{s\in[n]} x_{s,j} p_{s,d} \right|$ so that the objective really does represent the aggregate distance: otherwise, one could decrease the variables $m_{i,d}$ until a constraint is tight, as they appear in a single constraint each, thereby decreasing the objective. Conversely, for a minimum clustering, we can set $m_{i,d} = \left| p_{i,d} - \frac{k}{n} \sum_{s\in[n]} x_{s,j} p_{s,d} \right|$ and get a feasible solution to the MINLP.
For the maximum norm $\|.\|_\infty$, we can introduce variables $m_i \in \mathbb{R}$ and constraints $m_i \geqslant p_{i,d} - \frac{k}{n} \sum_{s\in[n]} x_{s,j} p_{s,d}$ and $m_i \geqslant -p_{i,d} + \frac{k}{n} \sum_{s\in[n]} x_{s,j} p_{s,d}$ for all $d \in [q]$. Then $m_i \geqslant \left\| p_i - \frac{k}{n} \sum_{s\in[n]} x_{s,j} p_s \right\|_\infty$ and for reasons similar to the previous MIP, we can solve the problem by minimising objective $\sum_{j\in[k]} \sum_{i\in[n]} x_{i,j} m_i$.
In the case that the data is integer, we can set the m-variables to be integers, and we obtain INLP problems.

Clustering (minimum aggregate distance or cluster diameter):
In this first version of clustering, we seek to minimise the sum of the aggregate distance of the cluster

points. We'll have $y_{i,h,j} \in \{0,1\}$ indicate if points p_i and p_h are in the same cluster j by constraining it with $y_{i,h,j} \leqslant x_{i,j}$, $y_{i,h,j} \leqslant x_{h,j}$, $y_{i,h,j} \geqslant x_{h,j} + x_{i,j} - 1$.
Then the objective becomes $\sum_{j\in[k]} \sum_{i\neq h\in[n]^2} y_{i,h,j}\|p_i - p_h\|$.
Alternatively, one can ask for clusters of uniformly small diameter. To get this we introduce variable $d_j \in \mathbb{R}$ for cluster j and require that $d_j \geqslant y_{i,h,j}\|p_i - p_h\|$ for all $i \neq h \in [n]^2$, so that d_j is greater that the largest distance that two points of cluster j can be separated by. Finally, to get a uniform bound on these diameters, we introduce $m \in \mathbb{R}$ and constraints $m \geqslant d_j$ for all $j \in [k]$ and minimise objective m.

25.2 Gröbner bases

From "Ideals, varieties and algorithms" and Hoekstra thesis

The link between integer programming and the algebra we develop in this section is explained in the first paragraph of the next section, which we recommend reading before proceeding here.

25.3 Gröbner base methods for ILP

It turns out that IP-feasibility turns up in a particular type of chage of variables in algebraic geometry. Indeed, for a monomial $y_1^{n_1}...y_k^{n_k}$, we can substitue $y_i = x_1^{a_{i1}}...x_k^{a_{ik}}$ to get $y_1^{n_1}...y_k^{n_k} = x_1^{(a_{11}n_1+...+a_{k1}n_k)}...x_k^{(a_{1k}n_1+...+a_{kk}n_k)}$. We can then ask if there is a monomial $y_1^{n_1}...y_k^{n_k}$ for which this change of variable will lead to a monomial $x_1^{b_1}...x_k^{b_k}$, which is equivalent to asking IP-feasibility of $\begin{cases} A^t n = b \\ n \geqslant 0 \\ n \in \mathbb{Z}^k \end{cases}$. Note that we're restrained to IP-feasibility with $A \geqslant 0$ (to deal with polynomials). There are generalizations for arbitrary A, which we won't develop. We will therefore study algebraic methods for answering the change of variable problem.

For a monmial $f = x_1^{b_1}...x_k^{b_k} \in \mathbb{C}[x_1, ..., x_k]$, and for the $f_i = x_1^{a_{i1}}...x_k^{a_{ik}} \in \mathbb{C}[x_1, ..., x_k]$, we want to know if the re is a monomial $h = y_1^{n_1}...y_k^{n_k} \in \mathbb{C}[y_1, ..., y_k]$, since for for which $h(f_1, ..., f_k) = f$. With elimination theory, we're lead to consider the ideal $K = \langle y_1 - f_1, ..., y_k - f_k \rangle \in \mathbb{C}[x_1, ..., x_k, y_1, ..., y_k]$, since for $h = y_1^{n_1}...y_k^{n_k}$, we have:

Characterisation:

We have $h(f_1, ..., f_k) = f \Leftrightarrow f - h \in K$.

Proof: Indeed, if $f-h \in K$, then $f(x_1, ..., x_k)-h(y_1, ..., y_k) = \sum_{i\in[k]} q_i(y_i-f_i)$ for some $q_i \in \mathbb{C}[x_1, ..., x_k, y_1, ..., y_k]$, so that $f(x_1, ..., x_k) - h(f_1, ..., f_k) = \sum_{i\in[k]} q_i(f_i - f_i) = 0$ and hence $h(f_1, ..., f_k) = f$.
Conversely, if $h(f_1, ..., f_k) = f$, we'll investigate $f-h$, which is under this assumption $f-h = h(f_1, ..., f_k)-h(y_1, ..., y_k) = \sum_{\alpha\in F} c_\alpha \left(f_1^{\alpha_1}...f_k^{\alpha_k} - y_1^{\alpha_1}...y_k^{\alpha_k}\right)$ for some finite set $F \in \mathbb{N}^k$ where h has all its non-zero coefficients indexed in multi-index notation. We can prove that this is in K with a nice computational trick, using the identity $(X - Y) | (X^n - Y^n)$. We write $f_1^{\alpha_1}...f_k^{\alpha_k} - y_1^{\alpha_1}...y_k^{\alpha_k}$ as a telescopic sum:

$$\begin{array}{lll} y_1^{\alpha_1}...y_k^{\alpha_k} & -f_1^{\alpha_1}y_2^{\alpha_2}...y_k^{\alpha_k} & \\ & f_1^{\alpha_1}y_2^{\alpha_2}...y_k^{\alpha_k} & -f_1^{\alpha_1}f_2^{\alpha_2}y_3^{\alpha_3}...y_k^{\alpha_k} \\ & & f_1^{\alpha_1}f_2^{\alpha_2}y_3^{\alpha_3}...y_k^{\alpha_k} \\ & & \ddots \\ & & f_1^{\alpha_1}...f_{k-1}^{\alpha_{k-1}}y_k^{\alpha_k} \quad -f_1^{\alpha_1}...f_k^{\alpha_k} \end{array}$$

Where we see that the sums columns cancel out, exept for the outer ones. Each line can be factored in form $f_1^{\alpha_1}...f_{i-1}^{\alpha_{i-1}}y_{i+1}^{\alpha_{i+1}}...y_k^{\alpha_k}\,(y_i^{\alpha_i} - f_i^{\alpha_i})$, which is divided by $(y_i - f_i)$, using the identity we mentioned. Therefore, $f - h \in K$.

Gröbner IP-feasibility:

We compute a Gröbner basis G for $\langle \left(y_i - x_1^{a_{i1}}...x_k^{a_{ik}}\right)_i \rangle$ wrt an eliminiation order in which the x variables are greater then the y variables. We determine the remainder of $x_1^{b_1}...x_k^{b_k}$ in the division algorithm by G. The IP is feasible precisely when the remainder is a monomial of form $y_1^{n_1}...y_k^{n_k}$.

COMPLETE CONTINIUE: thesis Hoekstra, IP minimisation, non-positive case.

25.4 Graver bases

Conformity:

For vectors of $\mathbb{R}^d$, we say that x is **conformal** to y, written $x \sqsubseteq y$, if their coordinates have pairwise same sign ($x_i y_i \geqslant 0$ for $i \in [d]$), and $|x_i| \leqslant |y_i|$ for all $i \in [d]$. We call a sum $s = \sum_{j=1}^{k} v_j$ **conformal** if $v_j \sqsubseteq s$ for all $j \in [k]$.

This is a partial order as it's transitive.

Graver base:

For a lattice of form $L(A) = \left\{x \in \mathbb{Z}^d : Ax = 0\right\}$, we consider the set of non-zero minimal lattice vectors wrt $\sqsubseteq$ (the v st. there is no w with $w \sqsubseteq v$ unless $w = v$), which is a finite (possibly empty) set called the **Graver basis** of the lattice, written G_A.

Proof: It's non-trivial to show that the Graver basis is a finite set.
We show that any subset S of $\mathbb{Z}^d$ has a finite number of non-zero minimal vectors wrt$\sqsubseteq$. We show this by induction on d, where for $d = 1$, $\sqsubseteq$ is just $\leqslant$for same signed numbers, hence the Graver basis is the minimum strict positive and maximum strict negative numbers of the closed set, and has size at most 2. For general d, we can focus on the intersection with one quadrate/orthant, wlog the positive one, as showing finiteness for all of them shows finiteness for the whole set. Note the interesting remark that for the positive quadrant, we see that minimality wrt $\sqsubseteq$ is equivalent to being Pareto-minimal.
Unless, the intersection of S with the orthant is empty, in whih case an empty Graver base does the job, there is at least one Graver basis element in the orthant. Indeed, if x is a point of the orthant we can keep finding elements $y_n \sqsubseteq ... \sqsubseteq y_1 \sqsubseteq x$ with strict inequalities until y_n is in the Graver basis: at each step $\|.\|_1$ decreases, but it's a positive integer, so there comes a point that can't be dominated further, and is minimal. So we have a minimal vector x, and we know that the vectors with $y \geqslant x$ can't be minimal as they're dominated by x, and there aren't any $y \leqslant x$ in the intersection of S with the positive orthant (except for x), as x is minimal. Hence we know that the rest of the Graver basis vectors (of S in this orthant) are in the intersections of S with the slices defined by $0 \leqslant y_i \leqslant x_i$, for the dimensions $i \in [d]$. These slices have form $\mathbb{N}^{d-1} \times \{0, .., x_i\}$, and we know that a minimal vector of $S \cap \mathbb{R}^d_+$ is also one of $S \cap \left(\mathbb{N}^{d-1}\{0, .., x_i\}\right)$ because is $z \in S \cap \left(\mathbb{N}^{d-1} \times \{0, .., x_i\}\right)$ is dominated $y \leqslant z$, then this is true in $S \cap \mathbb{R}^d_+$ as well. So the Graver basis elements of $S \cap \mathbb{R}^d_+$ are also minimal in $\mathbb{N}^{d-1} \times \{0, .., x_i\}$.
By slicing $S \cap \left(\mathbb{N}^{d-1} \times \{0, .., x_i\}\right)$ further into $S \cap \mathbb{N}^{d-1} \times \{k\}$ for $k \in \{0, .., x_i\}$, we get that minimal elements of $\mathbb{N}^{d-1} \times \{0, .., x_i\}$ are also minimal in $S \cap \mathbb{N}^{d-1} \times \{k\}$ for $k \in \{0, .., x_i\}$.
We note that for a minimal element z of $S \cap \left(\mathbb{N}^{d-1} \times \{0, .., x_i\}\right)$ with $z_d = k$, $z_{[d-1]}$ is minimal in the projection of $S \cap \mathbb{N}^{d-1} \times \{k\}$ on the first $d-1$ coordinates: if it weren't, so that $y \leqslant z_{[d-1]}$, then $(y, k) \leqslant z$, where $(y, k) \in S \cap \mathbb{N}^{d-1} \times \{k\}$, contradicting the minimality of z. Now the projection of $S \cap \mathbb{N}^{d-1} \times \{k\}$ on the first $d-1$ coordinates has a finite Graver basis by induction, since we're in dimension $d-1$. We've established that the Graver basis elements other then x must project to Graver basis vectors of $S \cap \mathbb{N}^{d-1} \times \{k\}$, for some slice in k for some slice in dimension i, of which there are only finitely many, and since at most $x_i + 1$ of them can project to the same one in that respective slice, there are only finitely many Graver basis vectors in the slices, hence in $S \cap \mathbb{R}^d_+$.

Graver basis decomposition:

All non-zero elements $x \in L(A)$ can be written as $x = \sum_{g \in G_A} k_g g$ for $k_g \in \mathbb{N}$, possibly not uniquely.
In fact, there is a conformal sum with at most $2d-1$ Graver basis vectors.

Proof: All elements $x \in L(A)$ can be written as $x = \sum_{g \in G_A} k_g g$ for $k_g \in \mathbb{N}$, possibly not uniquely.
We show this by induction on $\|.\|_1$, where the zeor case is handled by zero coefficients in the sum. If an arbitrary x isn't already in the Graver bases element, we can keep finding elements $y_n \sqsubseteq ... \sqsubseteq y_1 \sqsubseteq x$ with strict inequalities until y_n is in the Graver basis: at each step $\|.\|_1$ decreases, but it's a positive integer. Then $x - y_n$ has strictly lower norm $\|.\|_1$ as we are in the same quadrant at both vectors. So we use the induction assumption to get $x - y_n = \sum_{g \in G_A} k_g g$: incorporating y_n in the sum with a positive coefficient concludes the step.
A example where decompositions aren't unique is for the lattice of integer points on the plane $x+y-2z = 0$. We consider the positive orthant, in which $(2,0,1)$, $(0,2,1)$ and $(1,1,1)$ are in the Graver basis, which one can check by noting that all integer vectors of the positive orthant with at least one smaller coordinate don't satisfy $x + y - 2z = 0$. Here, $(2,0,1) + (0,2,1) = 2.(1,1,1)$ are different conformal sums of Graver basis vectors summing to $(2,2,2)$.
To get the bound on the number of Graver basis in a sum, we'll consider $\max 1^t \lambda$ st. $x = \sum_{g \in G_A, g \sqsubseteq x} \lambda_g g$ and $\lambda \geqslant 0$, which we expect to use few g in great multiplicity. This LP is feasible ($\lambda = k$) and bounded, since for all dimensions $|x_i| = \sum_{g \in G_A, g \sqsubseteq x} \lambda_g |g_i|$ as the coordinates have same sign by conformity, hence $\lambda_g \leqslant \frac{|x_i|}{|g_i|}$.
For an optimal λ provided by the simplex method, only at most d of the λ_g corresponding to the optimal basis are non-zero.
We next analyse the fractional part $y = \sum_{g \in G_A, g \sqsubseteq x} (\lambda_g - \lfloor \lambda_g \rfloor) g = x - \sum_{g \in G_A, g \sqsubseteq x} \lfloor \lambda_g \rfloor g \in L(A)$ by integrality and linearity. If $y = 0$, then $x = \sum_{g \in G_A, g \sqsubseteq x} \lambda_g g$ is a sum of $d \leqslant 2d-1$ terms and we got our conclusion. Otherwise, since we can write $y = \sum_{g \in G_A, g \sqsubseteq x} q_g g$ for $q_g \in \mathbb{N}$, and since by optimality $\sum_{g \in G_A, g \sqsubseteq x} (q_g + \lfloor \lambda_g \rfloor) \leqslant \sum_{g \in G_A, g \sqsubseteq x} \lambda_g$, as $\sum_{g \in G_A, g \sqsubseteq x} (q_g + \lfloor \lambda_g \rfloor) g = x$ provides feasibility, we have $\sum_{g \in G_A, g \sqsubseteq x} q_g \leqslant \sum_{g \in G_A, g \sqsubseteq x} (\lambda_g - \lfloor \lambda_g \rfloor) < d$ (since λ is basic), so that at most $d-1$ integers q_g can be non-zero in that sum, we know that in the positive integral sum $\sum_{g \in G_A, g \sqsubseteq x} (q_g + \lfloor \lambda_g \rfloor) g = x$ has at most $d + d - 1 = 2d-1$ entries (non-zero coefficients).
This sum is also conformal, hence we're done

To compute Graver bases of lattices, we can relate them to Gröbner bases by characterising them with primitive binomials of a toric ideal, which we now do.
TO COMPLETE: toric ideal version and computation with Hoekstra and its references.

25.5 Graver base methods for INLP

We now discuss how Graver bases can be used to solve a particular type of non-linear convex integer optimization problem:

Integer convex-separable optimization:

The solution space of such an optimization problem is $\mathbb{Z}^d \cap \{x : Ax = b, l \leqslant x \leqslant u\}$. The objective is to minimise $\sum_{j\in[d]} f_j(x_j)$, where the f_j are univariate convex functions mapping integers to integers. The minimum exists as the solution space is finite (integer vectors in a rectangle).

More generally, we can optimize $\sum_{i\in[d]} f_i\left(\sum_{j\in[d]} w_{ij}x_j\right)$ for such functions on that solution space and for integral W, by reducing it to problems of the first type, by minimising $\sum_{i\in[d]} f_i(y_i)$ over $\mathbb{Z}^d \cap \left\{\begin{pmatrix} x \\ y \end{pmatrix} : Ax = b, l \leqslant x \leqslant u,\ Wx - Iy = 0\ , l' \leqslant y \leqslant u'\right\}$, as the zero function is convex, and where l' and u' are computed by solving LPs of form $\min(\pm y_i)$ st. $\begin{cases} y_i = W_{i*}x \\ Ax = b, l \leqslant x \leqslant u \end{cases}$, which have finite solutions ($l \leqslant x \leqslant u$ means we deal with a polytope), that we round to integers.

We'll need to develop a bit of theory for integer convex-separable optimization before giving the algorithm for solving such a problem, given a Graver basis of $L(A)$. The general idea of the algorithm is that we perform local search, by moving along Graver basis vectors in a magnitude determined by a form of exact line search by binary search.

Integer convex binary search:

We can find the minimum of a convex f on $\mathbb{Z} \cap [r, s]$ for integer r, s as follows:
If $r = s$, then this is the optimum. Else, we check if the midpoints $\left\lfloor \frac{r+s}{2} \right\rfloor$ and $\left\lfloor \frac{r+s}{2} \right\rfloor + 1$ are $=$, in which case we output this value as its optimal,$<$, in which case se keep searching on $\left[r, \left\lfloor \frac{r+s}{2} \right\rfloor\right]$, or $>$, in which case se keep searching on $\left[\left\lfloor \frac{r+s}{2} \right\rfloor + 1, s\right]$. The algorithm takes at most $\log_2(s-r)$ iterations.

In the $=$ case, we know that f can't have lower values on $\left[r, \left\lfloor \frac{r+s}{2} \right\rfloor\right]$ or $\left[\left\lfloor \frac{r+s}{2} \right\rfloor + 1, s\right]$, as in the first case, $\left\lfloor \frac{r+s}{2} \right\rfloor$'s value would contradict convexity between r and $\left\lfloor \frac{r+s}{2} \right\rfloor + 1$, and similarly in the other case. In the $<$case, the minimum can't be in $\left[\left\lfloor \frac{r+s}{2} \right\rfloor + 1, s\right]$, as convexity between it and $\left\lfloor \frac{r+s}{2} \right\rfloor$ would be contradicted by $\left\lfloor \frac{r+s}{2} \right\rfloor + 1$. Similar arguments handle $>$.

Line search:

For a convex $f : \mathbb{Z}^d \to \mathbb{Z}$ and $h(k) = f(x + kg)$ for $k \in \mathbb{N}$, we solve $\min h(k)$ st. $l \leqslant x + kg \leqslant u$ and $k \in \mathbb{N}$, assuming $l \leqslant x \leqslant u$ (feasibility) and $f(x + g) < f(x)$ (descent direction) for $g \neq 0$ as follows. We determine how large k can be so that $l \leqslant x + kg \leqslant u$, by noting $k \leqslant s = \min \left(\begin{cases} \frac{u_i - x_i}{g_i} : g_i > 0 \\ \frac{l_i - x_i}{g_i} : g_i < 0 \end{cases} \right) < \infty$, since $g \neq 0$. We then use the binary search as we just described on $[0, s]$ to find the minimum, as h is convex.

To see that h is convex, split $x = (1 - t)x + tx$.
Note that this part is why we assume a finite solution space. Otherwise, we can't use binary search as we might have $s = \infty$, in which case we max have unboundedness or a minimum the distance to which we have no information on.

A key point in the analysis of our algorithm for INLP is that for a sub-optimal point x and a point x^* that achieves the minimum we have $x^* - x \in L(A)$ so that we have a conformal sum $x^* - x = \sum_{g \in G_A} k_g g$ of at most $2d - 1$ terms. We will now use:

Superadditivity lemma:

For a conformal sum $\sum g$ (possibly with repretitions), we have $f\left(x + \sum g\right) - f(x) \geqslant \sum (f(x+g) - f(x))$.

Then $0 > f(x^*) - f(x) = f\left(x + \sum_{g \in G_A} k_g g\right) - f(x) \geqslant \sum_{g \in G_A} k_g (f(x + g) - f(x))$. The key point is that we then know that not all tems of the last sum can be positive, so that there is at least one $g \in G_A$ for which $f(x + g) < f(x)$, aka. g is a descent direction when x is non-optimal.
Proof of the lemma: Since f is convex-separable, the result follows if it's true for the f_j, by adding the inequalities over j. So all we have to show is $f_j\left(x_j + \sum_g g_j\right) - f_j(x_j) \geqslant \sum_g (f_j(x_j + g_j) - f(x_j))$, where conformity of the sum implies that the g_i have the same sign, or are all zero, in which case the inequality is true. If we write $h(g_j) = f_j(x_j + g_j) - f(x_j)$, we reformulate the inequality as $h\left(\sum_g g_j\right) \geqslant \sum_g h(g_j)$. This is follows from $h(tx) \leqslant th(x)$ for $t \in [0, 1]$, which is itself due to convexity: note that f_j is convex and so h is as well, and $h(0) = 0$, so $h(tx) \leqslant th(x) \Leftrightarrow h(tx + (1 - t)0) \leqslant th(x) + (1 - t)h(0)$. Indeed, $\sum_g h(g_j) = \sum_g h\left(t_g \sum_g g_j\right) \leqslant \sum_g t_g h\left(\sum_g g_j\right) = h\left(\sum_g g_j\right)$ for $t_g = \frac{g_j}{\sum_g g_j} \in [0, 1]$ as all g_j have the same sign and sum up to 1, and we handled the case in which they're all zero.

Graver basis integer convex-separable optimization algorithm:

We start with the Graver basis of $L(A)$ and an initial feasible solution x_0. At each step, we check if there is a Graver basis vector g such that $f(x_k + g) < f(x_k)$ and if so, we choose the steepest descent direction g' that minimises $f(x_k + g')$. If there isn't, then x_k will be optimal. Otherwise, we

use line search on $h(k) = f(x_k + kg')$ st. $l \leqslant x + kg' \leqslant u$ and $k \in \mathbb{N}$, the solution k^* of which provides $x_{k+1} = x_k + k^* g'$.

Proof of correctness: Note that by $g \in L(A) \Rightarrow Ag = 0$, the points will satisfy $Ax_k = Ax_0 = b$, and by line search they will also satisfy $l \leqslant x_k \leqslant u$, so that they're feasible. As we've seen previously, if x_k isn't optimal, then a g that is a descent direction is guaranteed to exist, so that the algorithm stops only once the point is optimal. To see that the algorithm does stop at some point, we will show that for an optimum x^*, we have $f(x_{k+1}) - f(x^*) \leqslant \dfrac{2d-2}{2d-1}(f(x_k) - f(x^*))$ at each iteration, so that $f(x_k) - f(x^*) \leqslant \left(\dfrac{2d-2}{2d-1}\right)^k (f(x_0) - f(x^*))$ and since f maps integers to integers, when x_k is non-optimal, $1 \leqslant f(x_k) - f(x^*) \leqslant \left(\dfrac{2d-2}{2d-1}\right)^k (f(x_0) - f(x^*))$ so that $k \leqslant \dfrac{\ln(f(x_0) - f(x^*))}{\ln\left(\frac{2d-1}{2d-2}\right)}$, so there can be only finitely many iterations in which x_k isn't optimal.

To see that $f(x_{k+1}) - f(x^*) \leqslant \dfrac{2d-2}{2d-1}(f(x_k) - f(x^*))$, we use the superadditivity lemma on $x^* - x_k = \sum_{g \in G_A} k_g g$, where the sum have at most $2d-1$ terms. We get $f(x^*) - f(x_k) \geqslant \sum_{g \in G_A} k_g (f(x_k + g) - f(x_k))$, to which we add $t(f(x_k) - f(x^*))$ where t is the number of terms (with multiplicity of the k_g) of the sum, to get $(t-1)(f(x_k) - f(x^*)) \geqslant \sum_{g \in G_A} k_g (f(x_k + g) - f(x^*))$. Lower-bounding the sum by the minimum of its terms, say $f(x_k + g') - f(x^*)$, we get with positivity $(t-1)(f(x_k) - f(x^*)) \geqslant t\left(f(x_k + g') - f(x^*)\right)$. Now we see the point of selecting the steepest descent direction g'' in the Graver basis, as we then get $\dfrac{(t-1)}{t}(f(x_k) - f(x^*)) \geqslant \left(f(x_k + g'') - f(x^*)\right)$. Since line search produces a lower value, we get $\dfrac{(t-1)}{t}(f(x_k) - f(x^*)) \geqslant (f(x_{k+1}) - f(x^*))$. Finally, we note that $\dfrac{(t-1)}{t}$ increases in t, which is at most $2d-1$, so that $\dfrac{2d-2}{2d-1}(f(x_k) - f(x^*)) \geqslant (f(x_{k+1}) - f(x^*))$ (with positivity), as we desired to show.

COMPLETE: phase 1 feasibility with Onn's notes

25.6 Solutions

26 Aspects ot combinatorial optimization

26.1 Fractional combinatorial optimization: the discrete Newton method

A railway company wants to decide on where to build tracks so as to connect a set of cities between each other. This corresponds to finding a tree T in a graph (V, E) who's vertices are the cities and edges represent the available building spots. Each building on an edge e incurs a cost $c(e)$ and will yield a income $m(e)$. In an unexpected plot-twist, we now seek a tree that maximises the ratio of the total income to the total cost (instead of their difference): $\max\left(\frac{\sum_{e\in T} m(e)}{\sum_{e\in T} c(e)}\right)$ over all trees T of the graph.
More generally, for a combinatorial problem who's feasible solutions can be represented by a set of incidence vectors $X \subseteq \{0,1\}^n$, we seek to solve:

The fractional combinatorial optimization problem:
$\max_{x\in X}\left(\frac{a^t x - k}{b^t x}\right)$ for criteria vectors a and b in $\mathbb{R}^n$, where we assume that $a^t x > k$ and $b^t x > 0$ over X.

In an attempted IP-modeling perspective, we would introduce a variable $\lambda \in \mathbb{R}$ and constraints $\frac{a^t x - k}{b^t x} \leqslant \lambda \Leftrightarrow a^t x - k - \lambda b^t x \leqslant 0$ for all $x \in X$, so that minimising λ would provide that desired maximum ratio. However, the constraints are actually is non-linear.
If we write $h(\lambda) = \max_{x\in X}\left(a^t x - k - \lambda b^t x\right)$, then constraints become $h(\lambda) \leqslant 0$. Next, if we observe that for a fixed x, $\lambda \mapsto a^t x - k - \lambda b^t x$ strictly decreases in λ (due to $b^t x > 0$ over X), then the maximum $h(\lambda)$ also strictly decreases in λ (otherwise, the $a^t x - k - \lambda b^t x$ for which the maximum is attained would have increased). Thus, if $h(\lambda) = 0$ at some value, then $h(\lambda) \leqslant 0$ from there onward. Since our goal is to minimise λ subject to $h(\lambda) \leqslant 0$, we're actually looking for the unique root of h.

One can use the Newton method to find roots of differentiable functions: it consists of building a sequence of points by intersecting the tangent of the functions graph at that point with the zero-line. Here, the function h is piecewise affine: it's convex so that when $x \in X$ is maximiser for λ_1 and λ_2, the epigraph must contain the segment between their images, which corresponds to $a^t x - k - \lambda b^t x$, which will be on the boundary of the epigraph between these points, for otherwise, we'd reach a contradiction. We can still perform a Newton method as follows:

- Start at $\lambda_0 = 0$
- At each step i, find the maximiser $x_i \in X$ of $a^t x - k - \lambda_i b^t x$.
  In our examples, this requires finding a maximum spanning tree for weights $a_e - \lambda_i b_e = m(e) - \lambda_i c(e)$.
- Solve for the tangent intersecting the zero-line as λ_{i+1}: $a^t x_i - k - \lambda_{i+1} b^t x_i = 0 \Leftrightarrow \lambda_{i+1} = \frac{a^t x_i - k}{b^t x_i}$.
- Proceed until encountering a λ_s for which $h_s = a^t x_s - k - \lambda_s b^t x_s = 0$.

The immediate question is whether we'll ever encounter a λ_s for which $h_s = a^t x_s - k - \lambda_s b^t x_s = 0$ after a finite (and hopefully not too large) number of k steps.
Could we ever encounter an $h_s < 0$? By definition of λ_s in our algorithm, $a^t x_{s-1} - k - \lambda_s b^t x_{s-1} = 0$, so that by maximality in the definition of h_s, $h_s \geqslant a^t x_{s-1} - k - \lambda_s b^t x_{s-1} = 0$. Next, to see that we make progress, we note that $\lambda_{i+1} - \lambda_i = \frac{a^t x_i - k}{b^t x_i} - \lambda_i = \frac{h_i}{b^t x_i} \geqslant 0$. But do these steps get to small to quickly ?

By remarking that by maximality $h_i \geqslant a^t x_{i+1} - k - \lambda_i b^t x_{i+1} = a^t x_{i+1} - k - \lambda_{i+1} b^t x_{i+1} + (\lambda_{i+1} - \lambda_i) b^t x_{i+1}$, introducing λ_{i+1} to make h_{i+1} appear, we have $h_i \geqslant h_{i+1} + \dfrac{h_i}{b^t x_i} b^t x_{i+1}$ and finally $1 \geqslant \dfrac{h_{i+1}}{h_i} + \dfrac{b^t x_{i+1}}{b^t x_i}$.

We can exploit this identity by seeing that it implies that at least one of $\dfrac{1}{2} \geqslant \dfrac{h_{i+1}}{h_i}$ or $\dfrac{1}{2} \geqslant \dfrac{b^t x_{i+1}}{b^t x_i}$ is true. It's possible to do a deep analysis from this starting point that shows that the algorithm will terminate in time polynomial in n. However, it's very computational, so we'll skip it here.

As a excuse for an answer, we'll remark that for integer valued $b \in \mathbb{Z}^n \backslash 0$, $b^t x$ is integer valued so that $\dfrac{1}{2} \geqslant \dfrac{b^t x_{i+1}}{b^t x_i}$ may happen at most $\log_2(\|b\|_1)$ times. From that point onward, only $\dfrac{1}{2} \geqslant \dfrac{h_{i+1}}{h_i}$ may occur so that $0 \leqslant h_i < \varepsilon$ after at most $\log_2 \left(\dfrac{\|a\|_1}{\varepsilon} \right)$ iterations since $h_0 = a^t x_0 - k$.

26.2 Multi-objective combinatorial optimization: Pareto fronts and efficient paths

Suppose we have n objects with d quantifiable attributes. We would like to find the objects that can't be improved on every aspect.

Pareto efficiency and fronts:

Recall that for vectors $v_i \in \mathbb{R}^d$, we have $v_i \leqslant v_j$ if this holds on all coordinates, and $v_i < v_j$ if $v_i \leqslant v_j$ and $v_i \neq v_j$, so that at least on inequality is strict. For n such vectors, we call v_i **Pareto efficient/maximal** if there is no v_j with $v_i < v_j$. The set of Pareto maximal vectors is called the Pareto front. We have similar notions for minimality.

For example among $\binom{2}{0}, \binom{2}{1}, \binom{1}{2}$ and $\binom{0}{4}$ only $\binom{2}{0}$ is not maximal, as $\binom{2}{0} < \binom{2}{1}$, and the other 3 vectors consitute the Pareto front. For a visual representation of pareto maximality, draw the vectors and visualise the definition ()and make a connection to convexity)

The problem of finding the Pareto front is somtimes called the maximum vector problem. We present a brute force and a numerical algorithm for the problem.
In brute force, we can loop on i of a candidate set C initialized as $[n]$ and then on $j > i$ of C to check whether $v_i < v_j$ or $v_i > v_j$, or none of both, with d comparisions each time. If $v_i < v_j$ doesn't occure, we can add v_i to the pareto front, and if $v_i > v_j$ occurs, then we can delete j from C, as it can't be maximal so there is no point in checking further. The algorithm works be because when i is considered, we didn't delete it from the candidate set in the previous iterations, so it's not dominated by lower index vectors, and we then add i to the front if it isn't dominated by the rest of the vectors with greater index. A crude bound on time is $O\left(n^2 d\right)$.
Our next algorithm is numerical and linear in n, but exponential in the dimension and the vector size. We assume our vectors to be integer valued and let $M = \max_i(\|v_i\|_\infty) + 1$, so that all vectors are in the interior of a square of length M. The idea is that once we have found one maximum vector v_i, we know that there are no vectors in the quadrant $\left(v_i + \mathbb{R}_+^d\right) \setminus v_i$ and that we can ignore those in the quadrant $\left(v_i + \mathbb{R}_-^d\right)$. This leaves $2^d - 2$ quadrants to consider for further invesitagtion. Since we deal with integer vectors in the interior of the square, we know that the other vectors of the remaining quadrants are in squares of size $M-1$ at most. We thus use a divide and conquer approach. To divide, we seek a particular maximum vector to cut the cube into quadrants. A particular type of maximum vector is a lexicographically maximum one, wrt. some lexicographic order: it can be found in $O(dn)$ time by repreated search of (all) maxima on the coordinates (bubble-sort, storing all the occurances of the same value). All maximum vectors will then by considered b the algorithm, as they always in the quadrants that we don't ignore.
If $T(M)$ is the worst case ruuntime of the algorithm, we have $T(M) \leqslant \left(2^d - 2\right) T(M-1) + dn$, where we assume that T is increasing since we can solve the same problem in larger squares. Therefore $T(M) \leqslant 2^{dM} + dn2^{M+2}$ and this is an $O\left(2^{dM} + dn2^{M+2}\right)$ algorithm suited for a large number of vectors, few criteria represented by a small integers.

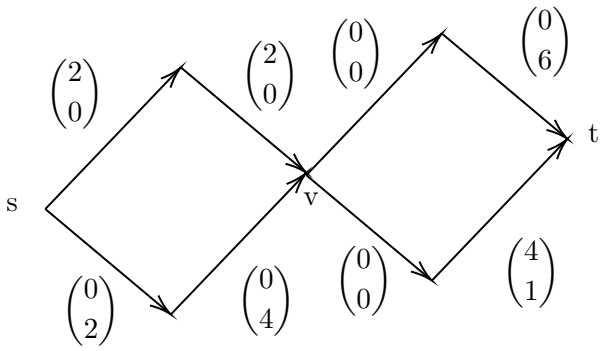
Multi-obcjective shortest path:
We'll now see that some algorithms can be adapted to a multi-objective setting, in which we seek Pareto efficient versions of an object. For the multi-objective shortest path problem, edge costs are now vectorial with d criteria, and the cost of a path is the sum of the vector costs of its edges. We seek all the Pareto minimal paths form a source s to a target t (or more precisely any other vertex).

In the following, we assume positive costs. This way, walks that contain cycles arn't efficient, as the

walk without the cycle is has less cost in every criterion, hence the walk is dominated and can't be minimal. This also holds if we assume to that there are no negative cycles for all criteria.

The key link between multi-objective shortest paths and the regular one is sub-path efficiency. If an s-t-path P is minimal then for any r on the path $P_{s\to r}$ must be minimal. This is because for vectors v, u, w we have equivalence between $v < u$ and $v + w < u + w$, so that replacing $P_{s\to r}$ by a pareto-better one from v to r would yield a better walk, hence a better path.

We'll actually restrain ourselves to finding one particular efficient s-t-path. This is because there may be a large amount of them: indeed, if we set all costs to be the zero vector, then all paths have cost zero, hence all are efficient, and we'd have to enumerate paths, of which there can be an amount exponential in the graph size (for example in a necklace of squares).
Still, this doesn't mean that we can discard efficient paths in the course of our algorithm, as the following example shows:



In this example, arrived at v from s, the two possible paths are incomparable with the top costing $\binom{4}{0}$ and the bottom costing $\binom{0}{6}$. If we discard the top one, then by taking bottom and then top or bottom, we get s-t-paths of costs $\binom{0}{12}$ and $\binom{4}{7}$ respectively. They are incomparable among each other, but the second one is actually not efficient ! Indeed, it's dominated by the top-top path of cost $\binom{4}{6}$. So unlike for the regular shortest paths, we have to keep track of all efficient paths, in a sense.

We will now adapt Dijkstra's to this multi-objective setting. We will use vertex labels l_P which will indicate the cost and the predecessor on an efficient s-v-path P. We collect them in a set label set L_v, as we'll keep track of multiple efficient paths. We adapt Dijkstra as follows: at each step, we consider all possible paths to vertices of the frontier and compute their Pareto front.
FIX/INCLUDE: there is no control on the number of paths we consider for this pareto front...
We then expand on the vertices of that front by considering their neighbours (one-by-one on th front). If a neighbour u isn't among the explored or frontier vertices, then we label it with the path that is the s-v-path followed by (v, u), marking v as predecessor, and add u to the frontier. Otherwise, if the neighbour u is in the frontier, we check if the s-v-path followed by (v, u) is dominated by a path from the labels of L_u, and if it isn't we add this path to the labels. In any case, we mark the v of the pareto front as explored. Once t is selected for this operation, we stop the algorithm.

The algorithm terminates since at each step one more vertex becomes explored, and the frontier is only empty once we've considered all the connected component of s. We now show that it works correctly, by showing the loop invariant that at each step, the path of the labels of a vertex that becomes explored in that iteration, has only explored vertices in it, and is in fact an efficient one for all paths of the graph. For

the first part, note that we add edges to paths that lead to unexplore vertices, or vertices in the frontier, who we expand upon only when they're selected to be explored.
Now, assume for contradiction that v has been selected to be explored, so that it has a label in L_v that is not dominated in the frontier at that iteration, but that the path P of the label isn't efficient in the graph. We consider an s-v-path P' that dominates it, in the sense that $c(P') < c(P)$. We consider the iteration where v becomes explored at its start, so that v is still considered to be in the frontier. We consider the first vertex u on P' that isn't explored. Since in general for all iterations, the neighbours of explored vertices are either explored or in the frontier by the end of the iteration, u must be in the frontier at the beginning of the iteration. By sub-path optimality, we have $c(P'_{s\to u}) \leqslant c(P')$, so that $c(P'_{s\to u}) < c(P)$. But $P'_{s\to u}$ is a path among the labels of L_u: this contradicts that $c(P)$ was in the Pareto front of the costs of all paths from all labels of vertices of the frontier, as its dominated by a path leading to u in the frontier.

FIX/COMPLETE: with Borndörfers paper or searching for Martin's algorithm...

26.3 Combinatorial optimization under uncertainty: frugal algorithms

Thesis Sahil

26.4 Online combinatorial optimization

Online optimization problems are problems in which the problem data is given sequentially, where at each step, an online algorithm has satisfy a certain requirement given a set of possible actions.
We'll look at two examples: online bipartite maximum matching and the online k-server problem.

In the online bipartite maximum matching problem. We're given a set of nodes V. Sequentially, we're given vertices of a set U, the size of which we don't know. We're given edges connecting the new vertex u to vertices of V, which we didn't know at a previous point of the algorithm. At each step, we must match u to a vertex of V, if this is possible, and we can't modify the previously selected edges.

An algorithm that solves this problem consists of matching the u to any unmatched v, at each iteration. This will produce a maximal, but not necessarily maximum matching. To see maximuality, assume there is an edge $\{u, v\}$ that we could at to the matching after termination: then u wasn't matched in the iteration when it showed up, despite v being available, which contradicts our algorithm.

We know from the matching section, that for a maximal matching m and a Maximum matching M, we have $|M| \leqslant 2|m|$. So the matching obtained with our algorithm is larger then half of the maximum possible value. Here, we say that our algorithm has **competitivity ratio** $\frac{1}{2}$.
In fact, no online algorithm can have a better ratio for this problem. Indeed, consider the following instances on nodes v_1, v_2 and u_1, u_2, where u_1 is connected to both v_1 and v_2. The instances differ in the fact that in one, u_2 is connected to v_1, while in the other it is connected to v_2. An arbitrary online algorithm recieving u_1 must match it to v_1 or v_2. In the first case, the first instance yields a final matching with one edge only, and in the second, it's the second instance that does. Yet, in both instances the maximum matching has size 2. So the competitivity ratio of this algorithm must be at least $\frac{1}{2}$.

COMPLETE: k Server lower bound, not all approx are online (ex: knapsack density)

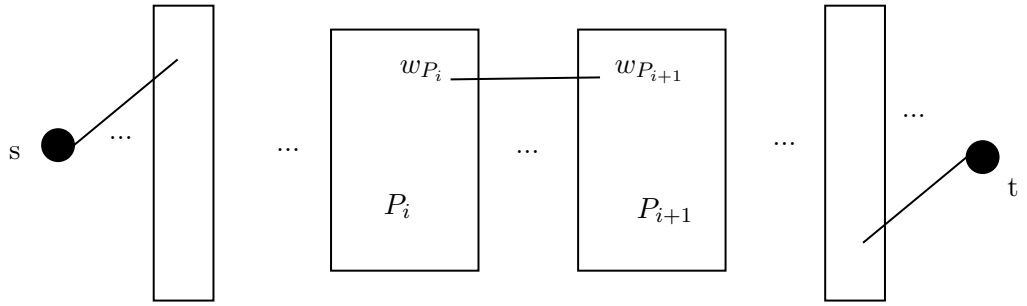
In the k-server problem, we're given a connected graph G of size $|V| > k$, with positive edge weights, so that we can define a metric d on V given by the legths of the shortest paths between vertices. We have k servers, that we initially place on vertices of the graph (we may place multiple on the same vertex). For an unknown number n or sequential requests, we're given a vertex u at each request. We must satisfy the request by moving a server to that vertex, if there isn't a server on the vertex already. In fact, we may move as many servers as we like in one iteration. The cost of that iteration will be the total distance travelled by the servers in that round (0 if no moves were made).

We'll investigate the strategy of moving the closest server to the vertex u_i at which there is a request in iteration i. We'll give a simple analysis for the instances in which at least $k+1$ different vertices of V recieve requests (so in particular $n \geqslant k+1$). This will force any algorithm (even offline, in the sense that it knows the sequence of requests before their appearance) to move at least one server at some point. This gives lower bound $d_{\min}$ on the cost of these instances, where $d_{\min} > 0$ is the minimum distance between pairs of vertices in G. We'll next use a very crude upper bound on the cost incurred by our algorithm: each move costs at most $d_{\max}$, the maximum distance between pairs of vertices in G, so that the cost of our algorithm is at most $nd_{\max}$. If alg is the cost of our algorithm, and opt is the minimum cost for the instance, then we have $alg \leqslant nd_{\max}$ and $d_{\min} \leqslant opt$, from which we can get a crude competitivity ratio with $alg \leqslant n\frac{d_{\max}}{d_{\min}} opt$.

We'll mention how to solve the offline version of the k-server problem, for the fun of it.
Similarly to cops and robbers, we'll reduce the number of servers as follows. We consider a graph P on ver-

tices V^k that represent the list of vertices where the servers are at. Its edges are $\{(v_1, ..., v_k), (u_1, ..., u_k)\}$, where $\{v_1, u_1\}$, ..., $\{v_k, u_k\}$ are either edges of the graph G, or have $v_i = u_i$, so that moving along that edge corresponds to moving (or keeping in place some of) the k servers along these edges in G. We exclude loops though. The cost of such an edge is the sum of the weights of edges $\{v_1, u_1\}$, ..., $\{v_k, u_k\}$ (think as 0 if $v_i = u_i$), so that the cost corresponds to moving alll k servers. They are also positive.

We will now reduce the problem to a shortest path problem. We create n copies of the graph P, that we connect in a line as in the figure below, where w_{P_i} and $w_{P_{i+1}}$ correspond to the same vertex in P. We also add a source s and a target t:



We give edges incident to the source cost 0. In the layers between copies of P, we give edges cost $k(d_{\max}+1)$, except for some particular ones. It will be the edge corresponding to (copies) vertices of P, that have form $w = (v_1, ..., v_k)$, where one of the v_i is the request of stage i, which we give cost 0. Essentially, we penalize not having a server at the requested vertex. We finish by doing the same for edges incident to t from the last copy of P.

Now, a shortest s-t-path must correspond to servers satisfying all requests. Indeed, it's always better to move servers so that at least one satisfies the request, which corresponds to moving in the copy of P, to then move to the next copy of P with an edge with weight 0, then to do something else, resulting in using an inter-copy-edge of weight $k(d_{\max} + 1)$ which is mode then any cost of any movement in the copy of P.
So the cost of this shortest path, which must only edges of weight 0 on the inter-copy-edges, is also the cost of the k-server instance. Since any k-server solution can be represented as a path in this model, the shortest path also represents the minimum k-server solution.
Note that the gadget we use is polynomial in the size of the problem, and shortest path can be solved in polynomial time, to that offline k-server is in P.

26.5 Solutions

27 Submodular optimization

27.1 Basics, examples, extensions

Submodular functions:

Given a finite set U and its power set 2^U, a **submodular function** is a function $f : 2^U \to \mathbb{R}$ that satisfies $f(A) + f(B) \geqslant f(A \cup B) + f(A \cap B)$.

An example of a submodular function and a corresponding optimization problem can be found in the **maximum cover problem**. In it, we're given n subsets A_i of a main set M, as well as positive weights for the elements of M in the form of $w : M \to \mathbb{R}_+$, where we denote $w(A_i) = \sum_{a \in A_i} w(a)$. The goal is to find $k \leqslant n$ indices in $[n]$ in the form of $S \subseteq [n]$ so a to maximise $f(S) = w(\cup_{i \in S} A_i)$.

To see that f is submodular, we start by writing;
$f(S) + f(R) = w(\cup_{i \in S} A_i) + w(\cup_{i \in R} A_i) = \sum_{a \in \cup_{i \in S} A_i} w(a) + \sum_{a \in \cup_{i \in R} A_i} w(a)$
$f(S \cup R) + f(S \cap R) = w(\cup_{i \in S \cup R} A_i) + w(\cup_{i \in S \cap R} A_i) = \sum_{a \in \cup_{i \in S \cup R} A_i} w(a) + \sum_{a \in \cup_{i \in S \cap R} A_i} w(a)$
In the sum $\sum_{a \in \cup_{i \in S} A_i} w(a) + \sum_{a \in \cup_{i \in R} A_i} w(a)$, an element a is accounted for precisely once, if either there is an A_i with $i \in S$ such that $a \in A_i$, but there is no $j \in R$ such that $a \in A_j$, which implies that $i \notin R$, so that $i \in S \backslash R$, or by similar arguments, $i \in R \backslash S$. An element a is accounted for precisely twice when there are $i \in S$ and $j \in R$ so that $a \in A_i$ and $a \in A_j$, with possibly $i \neq j$.

In the sum $\sum_{a \in \cup_{i \in S \cup R} A_i} w(a) + \sum_{a \in \cup_{i \in S \cap R} A_i} w(a)$, an element a is accounted for precisely once, if there is an A_i with $i \in S \cup R$ such that $a \in A_i$, but $i \notin S \cap R$, on which case it's accounted for twice. This is equivalent to $i \in S \backslash R$ or $i \in R \backslash S$. As mentioned, an element a is accounted for precisely twice, if there is an A_i with $i \in S \cap R \subseteq S \cup R$ such that $a \in A_i$.

We now see that the difference in accounting between the sums. the a accounted twice in the sum of $f(S \cup R) + f(S \cap R)$ are all also accounted twice for the sum of $f(S) + f(R)$ by considering the case $i = j$. The reverse for the twice accounted a may however not be true. Since weights w are positive, the possible a accounted twice in $f(S) + f(R)$ not accounted twice in $f(S \cup R) + f(S \cap R)$ make the latter sum larger, so that f is submodular.

Ex.CuSu: We consider a hypergraph (X, H) with positive hyperedge weights $w : H \to \mathbb{R}_+$. For a set of vertices $A \subseteq X$, the cut induced by A are the hyperedges containing both a vertex of A, and a vertex not in A. Such a cut has weight $W(A) = \sum_{h \in H : h \cap A \neq \emptyset, h \cap (X \backslash A) \neq \emptyset} w(h)$. Show that this is a submodular function.

An important concept is that of:

Marginal value:

For a submodular f and a set $A \subseteq U$, we define its **marginal value function** as $f_A : U \to \mathbb{R}$ through $f_A(a) = f(A \cup a) - f(A)$.
When f is submodular and we extend it to $f_A(X) = f(A \cup X) - f(A)$ for $X \subseteq U$, we have f_A submodular.

Proof: To see that f_A is submodular when f is, consider $f(A \cup X) - f(A) + f(A \cup Y) - f(A)$ and compare it to $f(A \cup Y \cup X) - f(A) + f((X \cap Y) \cup A) - f(A)$. But by noting that $A \cup Y \cup X = (A \cup Y) \cup (A \cup X)$ and $(X \cap Y) \cup A = (X \cup A) \cap (Y \cup A)$, this can be done directly from submodularity of f.

We'll now discuss a first approach to solving the maximum cover problem.
We'll try out the following greedy algorithm. Until we have obtained a set of indices of size k, we add elements to the current candidate set S by choosing among those that yield greatest marginal return $f_S(a)$ in the sense that we move to the next iteration by setting $S := S \cup a$ for $a = argmax_{a \notin S} f_S(a)$ (take one if multiple maxima) until S has size k.

Greedy max cover:

The greedy algorithm is a $0,6321 \approx (1 - \frac{1}{e})$-approximation algorithm for the max cover problem.

Proof: We denote by S_i the set produced by the greedy algorithm at iteration i, which contains i elements. We consider a set C achieving the maximum of this finite maximisation problem. We'll study the difference $f(C) - f(S_i)$. To relate it to $argmax_{a \notin S_i} f_{S_i}(a)$, we seek a bound in which the $f_{S_i}(a)$, which we'll find with a couple of tricks. The first remark is that for disjoint A and B, submodularity and the case that $f(\emptyset) = 0$ yield $f(A \cup B) \leqslant f(A) + f(B)$. Doing this recursivly for the elements that constitue a set, we get $f(A) \leqslant \sum_{a \in A} f(a)$. Now, to use this on the f_{S_i} of our problem, which we know to be submodular, since f is, we have to check that $f_{S_i}(\emptyset) = 0$, which is the case.
To compare $f(C) - f(S_i)$ and f_{S_i}, we'll use the following trick $f(C) - f(S_i) \leqslant f(C \cup S_i) - f(S_i) = f((C \backslash S_i) \cup S_i) - f(S_i) = f_{S_i}(C \backslash S_i)$, where the first inequality follows from the fact that in our problem, due to weights being positive, f is increasing under inclusion, so that $f(C) \leqslant f(C \cup S_i)$. With the previous trick, we have $f_{S_i}(C \backslash S_i) \leqslant \sum_{a \in C \backslash S_i} f_{S_i}(a)$. So we have the first intermediate result that $f(C) - f(S_i) \leqslant \sum_{a \in C \backslash S_i} f_{S_i}(a)$.

The rest of the proof is easier. Since $\max_{a \notin S_i} f_{S_i}(a) \geqslant f_{S_i}(a)$ for $a \in C \backslash S_i$ in particular, we get $f(C) - f(S_i) \leqslant |C \backslash S_i| \max_{a \notin S_i} f_{S_i}(a)$. Next, since $|C \backslash S_i| \leqslant |C| = k$ as C is optimal feasible, we get $f(C) - f(S_i) \leqslant k \max_{a \notin S_i} f_{S_i}(a)$. So if a_i attains the maximum, so that $S_{i+1} = S_i \cup a_i$, then $\frac{1}{k}(f(C) - f(S_i)) \leqslant f_{S_i}(a_i)$.
Finally, we can get an inductive bound on $(f(C) - f(S_i))$, by noting that $f(S_i) = f(S_{i-1}) + f_{S_{i-1}}(a_{i-1})$, so that $f(C) - f(S_i) \leqslant f(C) - f(S_{i-1}) - f_{S_{i-1}}(a_{i-1}) \leqslant \left(1 - \frac{1}{k}\right)(f(C) - f(S_{i-1}))$.
A quick induction yields $f(C) - f(S_k) \leqslant \left(1 - \frac{1}{k}\right)^k (f(C) - f(\emptyset))$. Smoothing things of with $\left(1 - \frac{1}{k}\right)^k \leqslant \frac{1}{e}$, and noting that $f(S_k)$ is the value of the greedy algorithms output, and $f(\emptyset) = 0$ in our problem, we can conclude.

Extensions:
An idea for solving or approximating submodular problems is to use relaxations. If we identify the power set 2^U to the vertices of the $|U|$-cube $[0,1]^{|U|}$, which are the elements of $\{0,1\}^{|U|}$, we can try to get a similar notion to the integer hull of an IP, if we can extend a submodular f to input values of $[0,1]^{|U|}$.

A first idea would be to seek a linear extension, in the sense that if we have convex combination $\sum_{S\in 2^U} \lambda_S 1_S = x$ for $x \in [0,1]^{|U|}$, where 1_S is the vertex corresponding to set S, we'd have function value $\sum_{S\in 2^U} \lambda_S f(S)$. However, there isn't always a unique way of writing x as a convex combination of vertices of $[0,1]^{|U|}$, which leads our definition to be problematic. A small clever adjustment does the job however.

Convex extension:

We define the convex extension of a submodular f to be

$$f_C(x) = \min\left(\sum_{S\in 2^U} \lambda_S f(S) : \sum_{S\in 2^U} \lambda_S 1_S = x, \sum_{S\in 2^U} \lambda_S = 1, \lambda \geqslant 0\right)$$

It's convex.

Proof: First, note that the minima in the definition are attained, as we're dealing with a compact, non-empty set. Next, take x and y in the cube, and simplex points λ and μ representing them as convex combinations respectively. Then, to investigate $f_C(tx+(1-t)y)$, note that $t\lambda+(1-t)\mu$ is also a simplex point, which represents $tx+(1-t)y$. By the minimum in the definition, we have $f_C(tx+(1-t)y) \leqslant \sum_{S\in 2^U} (t\lambda_S+(1-t)\mu_S)f(S)$, so that by distributing, we get $f_C(tx+(1-t)y) \leqslant tf_C(x)+(1-t)f_C(y)$, which is convexity.

Minimising the convex f_C on the hypercube provides a lower bound on the minimum of f. Indeed, all values of f can be attained on the vertices of the cube by f_L, where there is a unique convex combination representing the vertex, which has all its support on the set it represents. Form here, on could consider rounding methods in the hope of getting approximation algorithms for submodular problems. However, note that we don't know if f_C is differentiable, and worse, computing the value of the convex relaxation requires solving an LP on a large hypercube. So we can't use classic convex optimization algorithms.

Another approach is to define the relaxation value as an average. This yields good results:

Lovász extension:

We define the convex extention of a submodular f to be $f_L(x) = \int_0^1 f(\{i : x_i \leqslant t\})dt$.
We can compute this value explicitely, finding sets $S_0 = \emptyset \subsetneq S_1 \subsetneq ... \subsetneq S_{k_x-1} \subsetneq [|U|] = S_{k_x}$ and corresponding scalars such that $f_L(x) = \sum_{i=0]}^{k_x} \lambda_i f(S_i)$, where λ is a simplex point (a distribution).

Indeed, we can sort the $x_i \in [0,1]$ in increasing order with permutation π so that $x_{\pi(i)}$ is increasing in i.

Then $\int_0^1 f(\{i : x_i \leqslant t\})dt = \sum_{i=0}^{|U|} \int_{x_{\pi(i)}}^{x_{\pi(i+1)}} f(\{i : x_i \leqslant t\})dt$ (with $x_{\pi(0)} = 0$ and $x_{\pi(|U|+1)} = 1$), and since $\{i : x_i \geqslant t\}$ stays the same on $[x_{\pi(i)}, x_{\pi(i+1)}[$, we have $f_L(x) = \sum_{i=0}^{|U|}(x_{\pi(i+1)} - x_{\pi(i)})f(\{j : x_j < x_{\pi(i+1)}\})$.
As $x_{\pi(i)}$ is increasing and $x_{\pi(0)} = 0$ and $x_{\pi(|U|+1)} = 1$, the sequence $\lambda_i = (x_{\pi(i+1)} - x_{\pi(i)})$ forms a simplex point. The corresonding sets are $S_i = \{j : x_j < x_{\pi(i+1)}\}$, which forms an inclusion increasing sequence, as $S_{i+1} = S_i \cup \{\pi(i+1)\}$.
FIX: inequality directions and strictness, strictness of inclusion (same coord),...

We could show that this extension is convex for submodular f, however this follows from the following stronger result:

Extensions coincide:

For submodular f, we have $f_L = f_C$.

Proof: In order to relate the two concepts, we'll elucidate the convex combination defining f_L. We had a convex combination with coefficients λ indexed by an increasing sequence $S_1 \subseteq ... \subseteq S_n$. Does this combination relate to x ? The coefficients had for $\lambda_{S_i} = x_{\pi(i+1)} - x_{\pi(i)}$. If we now consider $\sum_{i=1}^{n} \lambda_{S_i} 1_{S_i}$, then we note that by inclusion of the S_i, the sequence of a coordinate in the sequence 1_{S_i} starts at 0, at which it stays until it becomes 1 (at the S_i where the corresponding element is added), where it stays for the rest of the sequence. The result is that the coordinate in the sum will be $1 - x_{\pi(i)}$, where i is the stage at which the coordinate sequence turned to 1, due to telescopic cancellations. Since for our sequence S_i, precisely $x_{\pi(i)}$ is added at stage i, we have $\sum_{i=1}^{n} \lambda_{S_i} 1_{S_i} = 1_U - x$.

FIX: order so that sum is $\sum_{i=1}^{n} \lambda_{S_i} 1_{S_i} = x$, same problem as for definition of lovasz extension
We then see that the convex combination in the Lovasz extension is of the type considered in the minimisation problem defining the convex extension, since $\sum_{i=1}^{n} \lambda_{S_i} 1_{S_i} = x$. This yields in particular $f_C \leqslant f_L$.

We can now try to establish a sort of converse: is a convex combination attaining the minimum in the definition of the convex extension, that happens to correspond to an increasing chain $S_1 \subseteq ... \subseteq S_n$ (with one element added at a time), actually the convex combination from the Lovasz extension ?
This turns out to be true. If our starting assumptions are $\sum_{i=1}^{n} \lambda_{S_i} 1_{S_i} = x$ and $S_1 \subseteq ... \subseteq S_n$, then we can show that $\lambda_{S_i} = x_{\pi(i+1)} - x_{\pi(i)}$. This is due to the system $\sum_{i=1}^{n} \lambda_{S_i} 1_{S_i} = x$ being triangular: if we order the elements in their order of apperance, in $S_1 \subseteq ... \subseteq S_n$, the system has form $\lambda(0|T) = x$, where T is upper triangular with diagonal an upper entries all 1.
So we can show that $f_C = f_L$, if we can display a convex combination attaining the minimum in the definition of the convex extension, that happens to correspond to an increasing chain $S_1 \subseteq ... \subseteq S_n$ (with one element added at a time) in terms of support.
We can do this with a tricky technique. Assume λ is a convex combination attaining the minimum in the definition of the convex extension. It doesn't satisfy the chain condition when we can find two sets A and B in its support so that $A \nsubseteq B$ and $B \nsubseteq A$. Note that for wlog $\lambda_A \geqslant \lambda_B$, we can rewrite the

combination $\lambda_A 1_A + \lambda_B 1_B = (\lambda_A - \lambda_B)1_A + \lambda_B 1_{A\cup B} + \lambda_B 1_{A\cap B}$, by accounting contributions with tableau

$$\begin{array}{ccc} A\backslash B & B\backslash A & A\cap B \\ (\lambda_A-\lambda_B) & 0 & (\lambda_A-\lambda_B) \\ \lambda_B & \lambda_B & \lambda_B \\ 0 & 0 & \lambda_B \end{array}.$$

For this new convex combination, the objective $\sum_{S\in 2^U} \lambda_S f(S)$ can only have decreased, due to submodularity, as $\lambda_A f(A)+\lambda_B f(B) = (\lambda_A-\lambda_B)f(A)+\lambda_B(f(A)+f(B)) \geqslant (\lambda_A-\lambda_B)f(A)+\lambda_B(f(A\cup B)+f(A\cap B))$. Since the original λ attained the minimum already, so does this new convex combination.

The main idea now is to use this shift as follows. On the space of convex combination attaining the minimum in the definition of the convex extension, we'll seek one maximising a strictly supermudular one. Performing the shift on it will yield a contradiction to maximality, provided $A \not\subseteq B$ and $B \not\subseteq A$, so that this maximum solution must obey the chain condition.

The supermodular function we'll maximise is $\sum_{S\in 2^U} \lambda_S |S|^2$. Indeed, we have $|A\cup B|^2 + |A\cap B|^2 \geqslant (|A| + |B\backslash A|)^2 + (|B| - |B\backslash A|)^2$, the right side simplifying to $|A|^2 + |B|^2 + 2|B\backslash A|(|A| - |B| + |B\backslash A|)$, which is strictly greater then $|A|^2 + |B|^2$, as $|A| - |B| + |B\backslash A| < 0$, when $A \not\subseteq B$ and $B \not\subseteq A$.

Using $|A\cup B|^2 + |A\cap B|^2 > |A|^2 + |B|^2$ and the shift, we'd get a convex combination still attaining the minimum, but one with greater value for $\sum_{S\in 2^U} \lambda_S |S|^2$, contradicting maximality. Hence, it's impossible to have set with $A \not\subseteq B$ and $B \not\subseteq A$ in the support of the maximum combination, so that the chain condition is satisfied.

We now introduce one more extension. It's also based on the notion of average, but under a different distribution:

Multilinear extension:

The multilinear extension is defined by $f_M(x) = \sum_{S\in 2^U} f(S) \prod_{i\in S} x_i \prod_{i\notin S} (1-x_i)$.

This is the expectation of $f(R)$, where R is a random set obtained by taking element i of U with probability x_i, independently of each other. It has the following properties:

Multilinear extension properties:

- It's multilinear.
- If f is inclusion increasing, then f_M satisfies $\partial_i f_M \geqslant 0$, so that in particular it's increasing along any vector in the positive quadrant.
- If f is submodular, then $\partial^2_{ij} f_M \leqslant 0$, so that in particular it's concave along any vector in the positive quadrant.
- If f is submodular, then it's convex in directions of for $e_i - e_j$ for $i \neq j$.

Proof: To see multilinearity, note that by develloping, on sees that the function is a sum of monomials, which are multilinear. Next, f f is increasing, then since $\partial_j f_M(x) = \sum_{S\in 2^U : j\in S} f(S) \prod_{i\in S\backslash j} x_i \prod_{i\notin S} (1-x_i) -$

$\sum_{S\in 2^U: j\notin S} f(S)\prod_{i\in S} x_i \prod_{i\notin S, i\neq j}(1-x_i)$, where the first sum can be rewritten as $\sum_{S\in 2^{U\setminus j}} f(S\cup j)\prod_{i\in S} x_i \prod_{i\notin S}(1-x_i)$ and the second as $\sum_{S\in 2^{U\setminus j}} f(S)\prod_{i\in S} x_i \prod_{i\notin S}(1-x_i)$, we get positivity due to $f(S\cup j)\geqslant f(S)$.

If f is submodular, then since $\partial^2_{kj} f_M(x) = \sum_{S\in 2^U: j,k\in S} f(S)\prod_{i\in S\setminus j,k} x_i \prod_{i\notin S}(1-x_i) + \sum_{S\in 2^U: j,k\notin S} f(S)\prod_{i\in S} x_i \prod_{i\notin S, i\neq j,k}(1-x_i) - \sum_{S\in 2^U: j\notin S, k\in S} f(S)\prod_{i\in S\setminus k} x_i \prod_{i\notin S, i\neq j}(1-x_i) - \sum_{S\in 2^U: k\notin S, j\in S} f(S)\prod_{i\in S\setminus j} x_i \prod_{i\notin S, i\neq k}(1-x_i)$, for $j\neq k$ and $\partial^2_{kj} f_M(x) = 0$ for $k = j$, we can for the first case rewrite the sums by indexing them over $2^{U\setminus j.k}$, so that we can group them under a sum that is the expectation of $f(S\cup j,k) + f(S) - f(S\cup k) - f(S\cup j)$, which are negative by submodularity, since $(S\cup k)\cup(S\cup j) = S\cup j,k$ and $(S\cup k)\cap(S\cup j) = S$, since $j\neq k$.

Note that by definition of f_M as a convex combination, we have $f_M \geqslant f_C = f_L$.
We conclude with the remark that evaluating the multilinear extension and its derivatives takes exponential time in $|U|$. Due to its nature of being the expectation of a random variable, one can however find good approximations of its value through sampling.

27.2 Submodularity and polyhedra

Polymatroids:

To a submodular f, we can associate a polyhedron called the (extended) **polymatroid** $P_f = \left\{x : x_A \leqslant f(A), \forall A \in 2^U\right\}$.

27.3 Maximisation

27.4 Minimisation

Schrijvers algorithm from Korte-Vygen

27.5 The continuous greedy algorithm

We consider n agents and m items, where agent i has valuation (willingness to pay) $w_i(S)$ for set $S \subseteq [m]$, a bundle of items. We assume $w_i \geqslant 0$, $w_i(\emptyset) = 0$ ("normalized"), increasing under inclusion ("free disposal"), and submodular (decreasing marginal gain).
Our goal is to allocate the items into n sets S_i so as to maximise the welfare/revenue of allocating bundle S_i to agent i, which is $\sum_{i\in[n]} w_i(S_i)$. We can rewrite the problem as choosing a set $S \subseteq [n] \times [m]$ consisting of pairs (i, j), corresponding to us giving item j to agent i, so that S must satisfy $\forall k \in [m], |\{(i, j) \in S : k = j\}| \leqslant 1$,meaning that each item is give to at most one agent.
COMPLETE: show matroid stuff, partition matroid Vondrack lectures with $E_i = i \times [m]$.
Then $f(S) = \sum_{i\in[n]} w_i(\{j \in [m] : \exists(i, j) \in S\})$ is a submodular function as a sum of such functions, which in turn are submodular since $\{j \in [m] : \exists(i, j) \in A \cup B\} \subseteq \{j \in [m] : \exists(i, j) \in A\} \cup \{j \in [m] : \exists(i, j) \in B\}$ and similarly for the inclusion, and we use monotonicity of w_i.

We'll study a randomised allocation scheme in which we give item j to agent i with probability x_{ij}, so that since we also have the possibility of keeping the item, we have constraints $x \geqslant 0$ and $\sum_{i\in[n]} x_{ij} \leqslant 1$, forming polytope P. We expect to get welfare $F(x)$, which is the multi-linear extension of f, that we hope to maximise as a function of the distribution.
The problem is therefore a non-convex one, but one that is based on submodularity.

We'll study a type of gradient flow algorithm that uses an initial value problem to define a function of feasible solution. We take as initial point $x(0) = 0$, which is feasible (we keep all items) and set up differential equation $x' = V(x)$, where $V(x) = argmax_{v\in P} \left(v^t \nabla F(x)\right)$ is piece-wise constant.
COMPLETE: use stability of LP to get this point...
We let x move in the direction that increases the first-order approximation of F the most. The condition $v \in P$ will ensure that we move slower near the boundary, and that $x \in P$, as we'll show later.

The continuous greedy algorithm:

For an optimal value $OPT = \max(F(y) : y \in P)$ and the solution x to the IVP, we have $x(1) \in P$ and $F(x(1)) \geqslant \left(1 - \frac{1}{e}\right) OPT$, so that this is an $\left(1 - \frac{1}{e}\right)$-approximation algorithm for the problem.

An immediate question is how to use this algorithm in practice. We can use a numerical approximation of the IVP defining x. In that case, $V(x)$ can be computed by solving an LP at all approximation points of x of the scheme. The numerical scheme will also impact the feasibility of the approximation of $x(1)$, as well as the approximation ratio.

Proof: Note that $\frac{1}{t}x(t) \in P$ for all $t \in]0, 1]$. This can be seen by writing $x(t) = 0 + \int_0^t V(x(s))ds$, where the latter integral is the limit of Riemann sums, which when scaled by $\frac{1}{t}$ can be interpreted as convex combinations of point of $V(x) \in P$, so that the limit is also in the closed convex P. In particular $x(1) \in P$.
We will first bound $V(x)$, before working with the differential equation. We will use the properties of the multi-linear relaxation of the submodular monotone f that we gave in the introductory chapter.

We consider the point $v \in P$ attaining the maximum OPT, and try to relate it to x and $F(x)$. To do that, we'll study $\phi(s) = F(x + sd)$ for a direction $d \geqslant 0$ and $s \in [0, 1]$, as we know it to be concave. To relate to v, we choose $d_i = \max(v_i - x_i, 0) \geqslant 0$. Now, $\phi(0) = F(x)$, and $\phi(1) = F(x + d) \geqslant F(v) = OPT$, since F is increasing along directions $D \geqslant 0$, and here $D = (x + d) - v \geqslant 0$ since $\max(v_i - x_i, 0) \geqslant v_i - x_i$. By concavity, $\phi(1) - \phi(0) \leqslant \phi'(0)(1 - 0)$, where $\phi'(0) = d^t \nabla F(x)$. Now, since $\nabla F \geqslant 0$ and $0 \leqslant d \leqslant v$ (if $\max(v_i - x_i, 0) \leqslant v_i$ since $x_i \geqslant 0$ as we assume $\frac{1}{t}x(t) \in P$ and $v \geqslant 0$ as $v \in P$), we have $d^t \nabla F(x) \leqslant v^t \nabla F(x)$. Finally, by its definition, $V(x)^t \nabla F(x) \geqslant v^t \nabla F(x)$, so that joining bounds yields $V(x)^t \nabla F(x) \geqslant OPT - F(x)$.

We can now bound the differential equation, starting by the study of $F(x(t))' = x'(t) \nabla F(x(t)) = V(x) \nabla F(x(t))$. Indeed, we have $F(x(t))' + F(x(t)) \geqslant OPT$ by the previous bound. Multiplying by e^t, we can see $\left(e^t F(x(t))\right)' \geqslant e^t OPT$, so that $eF(x(1)) - F(x(0)) \geqslant OPT(e - 1)$ after integration, and using $x(0) = 0$ and $F(0) = 0$ (since $w_i(\emptyset) = 0$), we finally get the desired bound after dividind by e.

27.6 Solutions

Ex.CuSu: To see that $W(A \cup B) + W(A \cap B) \leqslant W(A) + W(B)$, we will do some accounting on the edges considered. We'll disjoin hyperedges on which kind of vertices they contain. The types of vertices of interest are those of $A \cap B, A \backslash B, B \backslash A$, and $X \backslash (A \cup B)$, as these types partition the vertices X, and we can deduce $X \backslash A = (X \backslash (A \cup B)) \cup (B \backslash A)$, for example. In the following table, we disjoin hyperedges, so that on the left, $\times$ denotes the type of edge that contains vertices of the corresponding column set. We enumerate all $2^4 - 1$ possibilities. On the right, $\times$ denotes the sums (corresponding column), to which these types of hyperedges contribute:

$A \cap B$	$A \backslash B$	$B \backslash A$	$X \backslash (A \cup B)$	$W(A \cup B)$	$W(A \cap B)$	$W(A)$	$W(B)$
$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
	$\times$	$\times$	$\times$	$\times$		$\times$	$\times$
$\times$		$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
$\times$	$\times$		$\times$	$\times$	$\times$	$\times$	$\times$
$\times$	$\times$	$\times$			$\times$	$\times$	$\times$
		$\times$	$\times$	$\times$			$\times$
	$\times$		$\times$	$\times$		$\times$	
	$\times$	$\times$				$\times$	$\times$
$\times$		$\times$			$\times$	$\times$	
$\times$	$\times$				$\times$		$\times$
$\times$			$\times$	$\times$	$\times$	$\times$	$\times$
$\times$							
	$\times$						
		$\times$					
			$\times$				

For example, for edges containing element of $A \backslash B, B \backslash A$, and $X \backslash (A \cup B)$ but not $A \cap B$ (line 2), we the edge contains elements of A, B and $X \backslash (A \cup B)$, but not $A \cap B$, so that it contributes to $W(A \cup B)$, $W(A)$ and $W(B)$ but not $W(A \cap B)$.

We see that in all cases, the edge is accounted more or equally in $W(A) + W(B)$ then in $W(A \cup B) + W(A \cap B)$. Since weights are positive, this means the latter sum is bigger.

28 Discrepancy theory

28.1 Definitions, examples, properties

We want to perform a causal inference test on a group of n volunteers. We will split the volunteers into two groups of people, the first of which will take a drug, while the second one won't. We will then note the effect of having taken the drug, and compare it to results of the control group. The problem in this type of situation is limiting the effect of parameters others then having or not taken the drug. For example, if we split groups arbitrarily, we run the risk of one group happening to contain a specific behavior that the other doesn't, that will impact our observation of the effects of the drug, despite being unrelated to the drug.
Numbering volunteers from 1 to n, we can gather medical or socio-economic data from participants, so as to get groups $S_i \subseteq [n]$ for $i \in [m]$, if we consider m aspects to group people into. For example, S_1 can be an age group of peple over 70, S_2 can be the group of people that weigh more then 70 kg, S_3 the group of people having had a surgery in their life, etc. The goal is to split these people into two groups (for the drug study) as "balanced" as possible, in the sense that the split induced on the groups S_i should be balanced.
We can model the problem as follows: we consider a labeling $l : [n] \to \{-1, 1\}$ to determine the two groups, and consider the unbalance $\left|\sum_{j\in S_i} l(j)\right|$ (number of people more in the majoritary group) for each category S_i. To get a balanced biparition wrt. the categories, we can look for a labeling l that minimises the greatest unbalance, in the sense that we seek $\min_{l:[n]\to\{-1,1\}} \left(\max_i \left(\left|\sum_{j\in S_i} l(j)\right|\right)\right)$.

Combinatorial discrepancy:

For a family $S_i \subseteq [n]$ for $i \in [m]$, the problem $\min_{l:[n]\to\{-1,1\}} \left(\max_i \left(\left|\sum_{j\in S_i} l(j)\right|\right)\right)$ is known as the **combinatorial discrepancy** problem.

The example of an application we gave above is such a problem, and we'll see more applications in a later section. We now discuss a seemingly unrelated problem.

In numerical integration, one can estimate an integral $\int_A f(x)dx$ by an average $\frac{1}{n}\sum_{i=1}^{n} f(x_i)$ for some well distributed sample points $x_i \in A$. We know from the Riemann integral that for large n, this estimate will be good, for well distributed sample points, but an interesting question is what sample points to use, for a fixed number n of them, so that the approximation is good.

For $A = [0, 1]^d$, we seek a point set that mimics a uniformly distributed one as good as possible, without actually performing randomization. As a measure of quality of a sampling point set P, we can compare the expected number of points in some rectangle $R = \prod_{i=1}^{d}[a_i, b_i] \subseteq A$ under uniform sampling to the actual number of such points $|P \cap R|$, and look for the greatest difference between the two over all rectangles. For an idd. sequence of n points uniformily chosen in A, the probability of that point being in R is $vol(R) = \prod_{i=1}^{d}(b_i - a_i)$, so that we expect $n.vol(R)$ points to be in R (linearity of expecta-

tion over indicators). We'll consider the measure of quality $\sup_{R=\prod_{i=1}^d [a_i,b_i]\subseteq A} |n.vol(R) - |P\cap R||$ (supremum exists as we can bound by $2n$), and seek an n point set P minimising it so that we want to find $\inf_{\substack{P\subseteq A\\|P|=n}} \left(\sup_{R=\prod_{i=1}^d [a_i,b_i]\subseteq A} |n.vol(R) - |P\cap R||\right)$. More generally, we have:

Geometric discrepancy:

For a set A and a family $F \subseteq 2^A$ of measurable subsets of A, the **geometric discrepancy** problem is to find $D(A,n,F) = \inf_{\substack{P\subseteq A\\|P|=n}} \left(\sup_{S\in F} |n.vol(S) - |P\cap S||\right)$.
We denote by $D(P,A,n,F)$ the argument of the infinimum.

The relation to numerical integration is due to the Koksma-Hlawka inequality, which states that $\left|\int_{[0,1]^d} f(x)dx - \frac{1}{n}\sum_{x\in P} f(x)\right| = O\left(\frac{1}{n}D\left(P,[0,1]^d,n,F\right)\right)$ for a certain family of rectangles F and under certain assumptions on f. Therefore, seeking small discrepancy improves the approximation. Proving the Koksma-Hlawka inequality is a long and involved process, so we'll prove a much softer version as an excuse. A proof of the full inequality can be found on page 151 of "Uniform distribution of sequences" by Kuipers and Niederreiter (1974).

Excuse of a Koksma-Hlawka inequality:

For $d=1$, an $f\in C^1([0,1])$, and the family F of segements of $[0,1]$, we have

$$\left|\int_{[0,1]} f(x)dx - \frac{1}{n}\sum_{i=1}^n f(x_i)\right| \leqslant \frac{1}{n}\|f\|_\infty D(P,[0,1],n,F) + 2\|f'\|_\infty\left(1+\frac{1}{n}\right)$$

Proof: We consider P to be the sequence $x_1,...,x_n\in[0,1]$ and partition $[0,1]$ into interior disjoint segments containing a single x_i each: wlog. $x_1 < ... < x_n$ such as $S_1 = \left[0, x_1 + \frac{x_1+x_2}{2}\right], S_2 = \left[x_1+\frac{x_1+x_2}{2}, x_2+\frac{x_2+x_3}{2}\right], ..., S_n = \left[x_n - \frac{x_{n-1}+x_n}{2}, 1\right]$ does the job, for example.
We upper-bound $\left|\int_{[0,1]} f(x)dx - \frac{1}{n}\sum_{i=1}^n f(x_i)\right|$ by $\frac{1}{n}\sum_{i=1}^n \left|n\int_{S_i} f(x)dx - f(x_i)\right|$ (with the triangular inequality). Next, by the mean value theorem there are $m_i\in S_i$ such that $\int_{S_i} f(x) = vol(S_i)f(m_i)$, and by the choice of the S_i, we have $|P\cap S_i| = 1$, so that $\left|n\int_{S_i} f(x)dx - f(x_i)\right| = |n.vol(S_i)f(m_i) - |P\cap S_i|f(x_i)|$.
Next, with a first order Taylor approximation, Lipschitzness, $vol(S_i)\leqslant 1$, and the triangular inequality, we get a bound on the previous expression of $\left|f\left(\frac{x_i+m_i}{2}\right)\right| D(P,[0,1],n,F) + 2\|f'\|_\infty vol(S_i)(n+1)$.

Taking a final bound on f and taking the average $\left(\frac{1}{n}\sum_{i=1}^{n}\right)$, we then find that $\left|\int_{[0,1]} f(x)dx - \frac{1}{n}\sum_{i=1}^{n} f(x_i)\right| \leqslant \frac{1}{n}\|f\|_\infty D(P,[0,1],n,F) + 2\|f'\|_\infty\left(1+\frac{1}{n}\right)$, since the $vol(S_i)$ sum up to 1.

28.2 Continuous discrepancy

28.3 Combinatorial discrepancy

The Beck-Fiala method:
To get minimum possible discrepancy of 0, one could solve the system $\left\{\sum_{j\in S_i} x_j = 0 : i \in [m]\right.$, hoping to find a solution $x \in \{-1,1\}^n$, that would correspond to a labeling giving discrepancy 0. We may not get integer points, but we can, when there is a solution, find one in the n-cube $[-1,1]^n$ (scale the original solution so as to get a unit vector).

This is why the Beck-Fiala method works with a relaxation of the problem in which labels $l(j) \in \{-1,1\}$ are replaced by variables $x_j \in [-1,1]$. The idea of te method is to maintain subsystems of $\left\{\sum_{j\in S_i} x_j = 0 : i \in [m]\right.$ as long as possible, while adapting the solution to such a system to have one more ± 1 coordinate at eacg iteration. We start with initial solution $x^0 = 0$, maintaining a set $M_k \subseteq [m]$ of sets S_i at iterations k such that $\left\{\sum_{j\in S_i} x_j^k = 0 : i \in M_k\right.$, where x^k is the curent iterate. We let $M_0 = [m]$. Over the course of the algorithm, we'll have $M_{k+1} \subseteq M_k$. Another point is that we'de like to keep ± 1 coordinate at those values, once x_j^k reaches those values over the iterations k. We'll thus define a set $F_k \subseteq [n]$ of "floating" variables that haven't reached values ± 1, and call those that have, those of $[n]\backslash F_k$, "safe". We start with $F_k = [n]$, since $x^0 = 0$.

To determine x^{k+1} so as to keep safe variables safe, we'll solve system $\left\{\sum_{j\in S_i\cap F_k} x_j + \sum_{j\in S_i\backslash F_k} x_j^k = 0 : i \in M_{k+1}\right.$ for x, letting $x_j^{k+1} = \begin{cases} x_j^k : j \in S_i\backslash F_k \\ x_j^* : j \in S_i \cap F_k \end{cases}$ for one solution x^* of that system that we'll determine in a moment. This way, loop invariant $\left\{\sum_{j\in S_i} x_j^{k+1} = 0 : i \in M_{k+1}\right.$ is maintained in the next step. We know that there is a solution for x to the previous system since $x_j = x_j^k$ does the job, as $M_{k+1} \subseteq M_k$. However, the x_j^k is what we wanted to improve upon in terms of ± 1 coordinates. So we seek a condition for the system to have at least a 1-dimesnional line of solutions. This way, we can consider a point on the intersection of that line with the boundary of the cube $[-1,1]^{|F_k|}$: such a point x^* is a solution to $\sum_{j\in S_i\cap F_k} x_j^* + \sum_{j\in S_i\backslash F_k} x_j^k = 0 : i \in M_{k+1}$ and has at least one of its coordinates at value ± 1 (this is what it means to be on the boundary of the cube). This is the x^* we use to get x^{k+1}. We get F_{k+1} from F_k by deleting from it the variables who's value in x^* is ± 1, as they are now safe.

To find x^* in practice, we solve $\sum_{j\in S_i\cap F_k} x_j + \sum_{j\in S_i\backslash F_k} x_j^k = 0 : i \in M_{k+1}$ with Gaussian elimination, getting a representation of (some) solutions of form $x_{F_k}^k + \mathbb{R}d$ for some direction $d \neq 0$. We then solve $-1 \leqslant x_{F_k}^k + td \leqslant 1$ for the largest $t > 0$ by $\max_{j\in F_k} \left(\begin{cases} \frac{1-x_j^k}{d} : d > 0 \\ \frac{-x_j^k-1}{d} : d < 0 \end{cases} \right)$.
We now seek a condition that will guarantee that the solutions to $\left\{\sum_{j\in S_i\cap F_k} x_j + \sum_{j\in S_i\backslash F_k} x_j^k = 0 : i \in M_{k+1}\right.$ contain at least a line, which is the case if $\left\{\sum_{j\in S_i\cap F_k} x_j = 0 : i \in M_{k+1}\right.$ does (translate by $x_{F_k}^k$), which in turn is the case if there are more variables then contraints, aka. $|M_{k+1}| \leqslant |F_k|$. This can be ensured by making assumptions on the sets S_i and by choosing the M_k the right way. We'll get this bound by a combinatorial argument called double-counting. We image the bipartie graph with vertices M_{k+1} and F_k, where we connect $i \in M_{k+1}$ and $j \in F_k$ by an edge if $j \in S_i$. We have $\sum_{i\in M_{k+1}} \deg(i) = \sum_{j\in F_k} \deg(j)$. If we know that for our set-system, any j can be in at most t different sets S_i, then $\deg(j) \leqslant t$ so that

$\sum_{j\in F_k} \deg(j) \leqslant t|F_k|$. This will be our assumption on the set-system. In a similar vein and to eliminate t in the comming inequality, we'd like $\deg(i) \geqslant t$, which can be achieved by letting M_{k+1} be the sets S_i containing at least t floating variables in iteration k, in the sense that $|S_i \cap F_k| \geqslant t$. This is coherent with $M_{k+1} \subseteq M_k$, as the number of floating variables decreases, so that $F_{k+1} \subseteq F_k$, hence the $|S_i \cap F_k|$ can only get smaller, and the i's such that $|S_i \cap F_k| \geqslant t$ become fewer. We then have $\sum_{i\in M_{k+1}} \deg(i) \geqslant t|M_{k+1}|$, from which $|M_{k+1}| \leqslant |F_k|$ will follow.

Finally, we can discuss the analysis of the method. The algorithm terminates as the number of floating variables decreases to 0 in time $O(n)$. How low of a discrepancy do we get ? Recall that the idea was to keep the sets at "relaxed-discrepancy" for as long as possible. The only time from which $\sum_{j\in S_i} x_j^k \neq 0$ may occur is when $i \notin M_k$. By our definition of M_k, this happens when $|S_i \cap F_k| < t$. This meanst that for the rest of the algorithm, only at most t variables of S_i may still change value. At the final stage, we'll have $\left|\left(\sum_{j\in S_i\cap F_k} x_j^n + \sum_{j\in S_i\setminus F_k} x_j^n\right) - \left(\sum_{j\in S_i\cap F_k} x_j^k + \sum_{j\in S_i\setminus F_k} x_j^k\right)\right| = \left|\sum_{j\in S_i\cap F_k} x_j^n - x_j^k\right| \leqslant 2|S_i \cap F_k| < 2t$, via a triangular inequality and the fact that we're in $[-1, 1]$, where $x_j^n = x_j^k$ on $j \in S_i\setminus F_k$, since the variables where safe from that iteration onward and didn't change value.
Summarizing, we get:

The Beck-Fiala method:

The Beck-Fiala method provides a labeling of the combinatorial discrepancy problem for sets S_i such that any $j \in [n]$ is in at most t of the S_i, such that a discrepancy of value at most $2t - 1$ is achieved. It does so deterministicly in time $O(n)$.

We can actually get a better result with a simple randomization:

Uniform random labeling:

If we label the elements independently uniformily at random with ± 1,
then with probability at least $\frac{1}{2}$, we get a labeling of discrepancy less then $\sqrt{2n\ln(2m)}$.

Proof: If we label the elements uniformily with ± 1, then with probability at least $\frac{1}{2}$, we get a labeling of discrepancy less then $\sqrt{2n\ln(2m)}$.

This is because discrepancy perfectly fits the Chernov bound in that case, which is $p\left(\left|\sum_{j\in S} X_j\right| > \lambda\sqrt{|S|}\right) \leqslant 2e^{-\lambda^2/2}$, where X_j is the label of j, and has expected value 0 in this setting. Using this for the S_i and $\lambda = \sqrt{2\ln(2m)}$ provides $p\left(\left|\sum_{j\in S_i} X_j\right| > \sqrt{|S_i|2\ln(2m)}\right) \leqslant \frac{1}{2m}$.

With a union bound, $p\left(\cup_{i\in[m]}\left(\left|\sum_{j\in S_i}X_j\right|>\sqrt{|S_i|2\ln(2m)}\right)\right)\leqslant\sum_{i\in[m]}p\left(\left|\sum_{j\in S_i}X_j\right|>\sqrt{|S_i|2\ln(2m)}\right)\leqslant\frac{1}{2}$ so that taking negation leads to $p\left(\cap_{i\in[m]}\left(\left|\sum_{j\in S_i}X_j\right|\leqslant\sqrt{|S_i|2\ln(2m)}\right)\right)\geqslant 1-\frac{1}{2}=\frac{1}{2}$. Finally, since $\cap_{i\in[m]}\left(\left|\sum_{j\in S_i}X_j\right|\leqslant\sqrt{|S_i|2\ln(2m)}\right)$ implies $\cap_{i\in[m]}\left(\left|\sum_{j\in S_i}X_j\right|\leqslant\sqrt{n2\ln(2m)}\right)$,the latter has probability at least $\frac{1}{2}$, and it is that of discrepancy being at most $\sqrt{n2\ln(2m)}$.

INCLUDE: Rothvoss algorithm

28.4 Advanced applications

Applications to approximation algorithms (rothvoss bin packing) and numerical integration.

28.5 Solutions

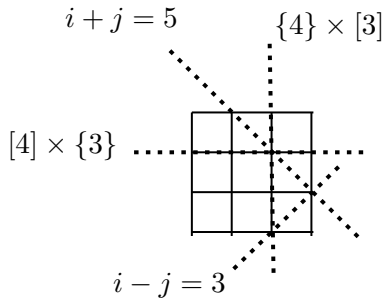
29 Inverse problems

29.1 Examples

Inverse problems generally study problems surrounding equations $y = f(x)$ from numerical and probabilistic viewpoints. They're best described by examples:

X-ray tomography:
X-ray tomography is a method that allows to reconstruct 3D objects (so also their interior) in a certain sense from a series of images of the object taken with light in the X-ray spectrum. If we consider an object with material density given by the map $f : \mathbb{R}^3 \to \mathbb{R}_+$, then we can place an X-ray source at $x_s \in \mathbb{R}^3$ and a receptor at $x_r \in \mathbb{R}^3$ and shoot X-rays from x_s to x_r in a straight line. When the X-rays come into contact with the object, a part of them is absorbed or deflected. We consider this portion to be proportional to the material density at that point, so that the intensity of the ray satisfies $I'(x) = -f(x)I(x)$ for $x \in [x_s, x_r]$ (for a fixed density, the more photons arrive, the more get deflected, hence the term in I). We get $\ln\left(\frac{I(x_s)}{I(x_r)}\right) = \int_0^1 f(x_s + t(x_r - x_s))dt$, where the left term is known to us, as we set and collect these values. A single such ray gives little information on the object, which is represented by f, as many f can satisfy $\ln\left(\frac{I(x_s)}{I(x_r)}\right) = \int_0^1 f(x_s + t(x_r - x_s))dt$. However, if we shoot multiple different rays intersecting each other, we hope to constrain f enough so that a solution to the equations we set will be close to the real material density map.

For simplicity, we will discretize the problem and use convenient rays. We imagine our object to be in a cube in the positive orthant in space, and consider a grid of lattice points $l\mathbb{Z}^3 \cap [0, M]^3$ where l is the grid length and M is the size of the cube the object is in. We will shoot rays in directions e_i for all 3 dimensions i, as well as diagonal rays in directions $e_i \pm e_j$ for pairs of dimensions (never in opposite directions, as we'd get the same measurements). There are for a grid of n^3 points a total of $3n^2$ grid parallel rays, each meeting n grid-points of form $l(i, j, k)$ for just one coordinate varying. There are $6n(2n + 1)$ diagonal rays that meat q points, for $q = 0, ..., 2n$ or $q = -n, ..., n$, of form $l(i, j, k)$ where one of the coordinates, say k, is fixed and where the other two satisfy $i \pm j = q$.
We could also have added rays non-parallel to orthant planes. Here is a picture in dimension 2:



We will consider the material density at grid point $l(i, j, k)$ to be f_{ijk}. We then discretize the integrals over the rays as Rieman sums. For grid-parallel rays, we get for example $\sum_{k=1}^{n} l f_{ijk}$, and for diagonal ones $\sum_{i \pm j = q} l\sqrt{3} f_{ijk}$. We then seek to solve a linear system of form $Af = m$, where m are the X-ray intensity measurements corresponding to the rays. For dense discretizations, so large n, the number of variables n^3 will surpass the number of constraints $3n^2 + 6n(2n + 1)$ by a long shot.
FIX: actually LP feasibility due to positivity.

29.2 Deterministic case

Müller Siltanent SIAM book, chapters on truncated SVD and Tikhonov regularization

29.3 Probabilistic case

Part 1 of "Inverse Problems and Data Assimilation" in folder "InversePorblems"

29.4 Solutions

30 Basics in functional optimization

30.1 Calculus of variations

We focus on the following functional optimization problem:

Problem:

For a function $f : C^1(\mathbb{R}^3 \to \mathbb{R})$, we which to minimise $F(y) = \int_a^b f(x, y(x), y'(x))dx$
over all functions $y \in C^1(a, b)$ satisfying $y(a) = y_a$ and $y(b) = y_b$.

For example, when we seek the shortest path between two points on a surface, if we can parametrize the surface with a smooth $\phi : \mathbb{R}^2 \to \mathbb{R}^3$ and the points are $\phi(y_a)$ and $\phi(y_b)$, then we're looking for curves the surface of form $\phi \circ y$ where $y \in C^1(a, b)$ satisfies $y(a) = y_a$ and $y(b) = y_b$ and the length of the curve $F(y) = \int_a^b \|(\phi \circ y)'(x)\|dx = \int_a^b \|D_{y(x)}\phi(y'(x))\|dx$ is minimum.

The Euler-Lagrange equation:
Euler initially derived this necessary condition for a y to be an optimum to the problem by discretising it and analysing the "asymptotic conditions" that come from the first order condition on the finite problems. Lagrange derived it from a similar argument to the finite dimensional case, as we now develop.

If y is a local optimum of F, then by mimicking the finite dimensional first order condition derivation, we consider a variation $h \in C^1(a, b)$ with $h(a) = h(b) = 0$, and analyse the one dimensional $H : \mathbb{R} \to \mathbb{R}$ defined by $H(t) = F(y + t.h)$. For this function, we expect 0 to be a local minimum.
A local minimum y^* of F over the solution space is a function such that for all y in the solution space with $\|y^* - y\|_N \leqslant \varepsilon$ for some ε and some function space norm $\|.\|_N$, we have $F(y^*) \leqslant F(y)$. For the moment, we'll ignore the dependency of this notion on the function norm and take $\|.\|_N = \|\|_{\infty,[a,b]}$. Then 0 is a local minimum of H since $\|y - (y + t.h)\|_N = |t|.\|h\|_N \leqslant \varepsilon$ for $|0 - t| \leqslant \frac{\varepsilon}{\|h\|_N}$, so that $H(0) = F(y) \leqslant F(y + t.h) = H(t)$. Note that we assumed y to be a local minimum over the solution space, but since $y + t.h$ satisfies the smoothness and boundary conditions, we get the desired result.

We can then use the first order condition on the one dimensional case, provided that $H \in C^1$.
The latter is the case, and the derivation formula for parameterised integrals and the chain rule yield $H'(t) = \int_a^b (h(x).\partial_2 f(x, y(x) + t.h(x), y'(x) + t.h'(x)) + h'(x)\partial_3 f(x, y(x) + t.h(x), y'(x) + t.h'(x)))dx$.
We can simplify this expression with partial integration, as $\int_a^b h'(x)\partial_3 f(x, y(x)+t.h(x), y'(x)+t.h'(x))dx = [h(x)\partial_3 f(x, y(x) + t.h(x), y'(x) + t.h'(x)))]_a^b - \int_a^b h(x)\partial_x\partial_3 f(x, y(x) + t.h(x), y'(x) + t.h'(x))dx$, as $h(a) = h(b) = 0$, so that $H'(t) = \int_a^b h(x)(\partial_2 f(x, y(x) + t.h(x), y'(x) + t.h'(x)) - \partial_x\partial_3 f(x, y(x) + t.h(x), y'(x) + t.h'(x)))dx$, by assuming (**with** loss of generality) that $y, h, f \in C^2$.
In particular, $H'(0) = \int_a^b h(x)(\partial_2 f(x, y(x), y'(x)) - \partial_x\partial_3 f(x, y(x), y'(x)))dx$.

So for a local minimum y over the solution space, the first order condition must hold for all variations h, we have for all $h \in C^2(a,b)$ with $h(a) = h(b) = 0$, $\int_a^b h(x)(\partial_2 f(x, y(x), y'(x)) - \partial_x \partial_3 f(x, y(x), y'(x)))dx = 0$. As in the finite dimensional case, we'll show that this implies that $\partial_2 f(x, y(x), y'(x)) + \partial_x \partial_3 f(x, y(x), y'(x)) = 0$. Unfortunately, contrary to the finite dimensional case, we don't have a basis of C^2, nor do we have the guarantee that $\partial_2 f(x, y(x), y'(x)) + \partial_x \partial_3 f(x, y(x), y'(x))$ is 0 at a and b.

The idea is that we can choose h to be a function that acts as an indicator function, or a basis vector if we follow the analogy to the finite case, by having a small support. For a point $p \in (a,b)$, we can set $h_{p,\delta}(x) = \begin{cases} \left(x - p + \frac{\delta}{2}\right)^3 \left(p + \frac{\delta}{2} - x\right)^3 on \ \left[p - \frac{\delta}{2}, p + \frac{\delta}{2}\right] \\ 0 \ else \end{cases}$. Note that $h_{p,\delta}(x) > 0$ on $\left(p - \frac{\delta}{2}, p + \frac{\delta}{2}\right)$. The analogy to the basis vector is that if there was a $p \in (a,b)$ such that $\partial_2 f(p, y(p), y'(p)) + \partial_x \partial_3 f(p, y(p), y'(p)) > 0$ or $\partial_2 f(p, y(p), y'(p)) + \partial_x \partial_3 f(p, y(p), y'(p)) < 0$, then by using variation $\pm h_{p,\delta}(x)$ for δ small enough so that the sign of $\partial_2 f(x, y(x), y'(x)) + \partial_x \partial_3 f(x, y(x), y'(x))$ doesn't change on $\left(p - \frac{\delta}{2}, p + \frac{\delta}{2}\right)$, we have the condition $\int_a^b h_{p,\delta}(x)(\partial_2 f(x, y(x), y'(x)) + \partial_x \partial_3 f(x, y(x), y'(x)))dx = \int_{p-\frac{\delta}{2}}^{p+\frac{\delta}{2}} \pm h_{p,\delta}(x)(\partial_2 f(x, y(x), y'(x)) + \partial_x \partial_3 f(x, y(x), y'(x)))dx = 0$, which is impossible as the integrand is positive on the integration segment.

For this argument to work we need to show that $h_{p,\delta}(a) = h_{p,\delta}(b) = 0$ and $h_{p,\delta} \in C^2$. The first condition is achieved if $\left[p - \frac{\delta}{2}, p + \frac{\delta}{2}\right] \subsetneq (a,b)$, which is the case. The second condition requires us to check that the first and second derivatives are 0 at $p \pm \frac{\delta}{2}$, which is the case since all terms contain either $\left(x - p + \frac{\delta}{2}\right)$ or $\left(p + \frac{\delta}{2} - x\right)$ in their factors.

Summarizing:

Euler-Lagrange:

If F (with $f \in C^2$) has a C^2 local minimum y over the boundary value solution space, then it must be a solution to the differential equation $\partial_2 f(x, y(x), y'(x)) - \partial_x \partial_3 f(x, y(x), y'(x)) = 0$, which is called the Euler-Lagrange equation.

Ex.EP: We consider the functional $F(y) = \int_a^b f\left(x, y(x), y'(x), ..., y^{(n)}(x)\right) dx$ with $f \in C^2$ to be optimised over the space of functions $y \in C^{2n}$ satisfying boundary constraints of type $y^{(i)}(a) = y_{a,i}$ and $y^{(i)}(b) = y_{b,i}$ for $i \in \{0, ..., n-1\}$. Show that a local extremum must satisfy the **Euler-Poisson** equation $\partial_2 f(x, y, ..., y^{(n)}) - \partial_x \partial_3 f(x, y, ..., y^{(n)}) + ... + (-1)^n \partial_x^n \partial_{n+2} f(x, y, ..., y^{(n)}) = 0$.

Ex.ELV: We consider the functional $F(y) = \int_a^b f(x, y_1(x), ..., y_n(x), y_1'(x), ..., y_n'(x))dx$ with $f \in C^2$ to be optimised over the space of vectors Y of functions $y_i \in C^2$ satisfying boundary constraints of type $y_i(a) = y_{a,i}$ and $y_i(b) = y_{b,i}$ for $i \in [n]$. Show that a local extremum must satisfy the equations $\partial_{i+1} f_i(x, Y, Y') - \partial_x \partial_{i+n+1} f(Y, Y') = 0$ for all $i \in [n]$.

Ex.ELMD: We consider the functional $F(u) = \int_A f(x, y, u(x,y), \partial_1 u(x,y), \partial_2 u(x,y))da$ with $f \in C^2$

to be optimised over the space of functions $u \in C^2$ satisfying boundary constraints $u|_{\partial A} = v$ for compact set $A \subseteq \mathbb{R}^2$ such that ∂A can be parameterized by a single smooth curve. Show that a local extremum must satisfy the equations $\partial_3 f - \partial_x \partial_4 f - \partial_y \partial_5 f = 0$.

Constraints:
We now constrain the problem with $G(y) = \int_a^b g(x, y(x), y'(x))dx = l$ in addition to the boundary values constraint, with $g \in C^2$. Our goal is again to reduce this to a finite dimensional problem by considering variations. The problem is that for variations h with $h(a) = h(b) = 0$, we may not necessarily have $\int_a^b g(x, y(x) + t.h(x), y'(x) + th'(x))dx = l$, yet all we're assuming is that y is a local minimum among all functions satisfying $G(y) = l$
A clever trick can then be used if we investigated the effect of multidimensional variations. Indeed, for y to be a local minimum, $(0, 0)$ has to be a local minimum of $H(t, s) = \int_a^b f(x, y(x)+t.h_1(x)+s.h_2(x), y'(x)+t.h_1'(x) + s.h_2'(x))dx$, for variations satisfying $h_i(a) = h_i(b) = 0$. The study of the first order condition just leads back to the regular Euler-Lagrange equation.
However, requiring $V(t, s) = \int_a^b g(x, y(x) + t.h_1(x) + s.h_2(x), y'(x) + t.h_1'(x) + s.h_2'(x))dx - l = 0$ is more interesting as it defines a manifold. In fact, we may apply the implicit function theorem to V arround $(0, 0)$ since its C^1 and since $\partial_2 V(0, 0) = \int_a^b h_2(x)(\partial_2 g(x, y(x), y'(x)) - \partial_x \partial_3 g(x, y(x), y'(x)))dx$ (same integration by parts as for the unconstrained case), which we can ste to be non-zero if y doesn't satisfy the Euler-Lagrange equation for g as we can set h_2 to be an "indicator" of a non-zero valued point .We get the existence of $p \in C^1$ such that there is a inteval $]-\varepsilon, \varepsilon[$ on which $V(t, p(t)) = 0$.
Now we have the result that 0 is a local minimum for $H(t, p(t))$.
TO CONCLUDE: Lagrange multipliers or variation with expression of p via IFT.

Second variation:
We've considered the first order condition on H to be a necessery condition for a local minimum, but there's also the second order condition. For $H(t) = \int_a^b f(x, y(x)+th(x), y'(x)+th'(x))dx$, we can compute the second derivative at 0 to be $H''(0) = \int_a^b h(x)^2.\partial_2^2 f(x, y(x), y'(x)) + 2h(x)h'(x).\partial_{2,3} f(x, y(x), y'(x)) + h'(x)^2.\partial_3^2 f(x, y(x), y'(x))dx$, assuming that $f \in C^2$. Now if y is a local minimum of F, then it's required that $H''(0) \geqslant 0$, in addition to the first order condition. We can make this last condition more digestible, just like we did for the first order condition by deriving the Euler-Lagrange equation.

Legendre condition:

If y is a local minimum of F, then in addition to it solving the Euler-Lagrange equation, we must have $\partial_3^2 f(x, y(x), y'(x)) \geqslant 0$ on $[a, b]$.

Proof: We will prove the contrapositive. So we assume that there is an x_n in $[a, b]$ for which $\partial_3^2 f(x_n, y(x_n), y'(x_n)) < 0$, and we want to show that there is a variation h such that $H''(0) < 0$, so that for some small enough t's, we have $F(y + th) < F(y)$, and therefore y isn't a local minimum.
To simplify notation, we'll write $H''(0) = \int_a^b h(x)^2 P(x) + 2h(x)h'(x)Q(x) + h'(x)^2 R(x)dx$ where $P(x) = \partial_2^2 f(x, y(x), y'(x))$, $Q(x) = \partial_{2,3} f(x, y(x), y'(x))$ and $R(x) = \partial_3^2 f(x, y(x), y'(x))$.

The starting point of the proof is that the integrand in $H''(0)$ looks like a square of a sum. To complete the square however, we must handle P, Q, R. We will modify the expression with a clever trick: for any $w \in C^1$, we can use boundary conditions on h to deduce that $\int_a^b \left(2h(x)h'(x)w(x) + h(x)^2w'(x)\right) dx = \int_a^b \left(h(x)^2w(x)\right)' dx = \left[h(x)^2w(x)\right]_a^b = 0$. By linearity on the integral, we can therefore introduce this into $H''(0)$ to get $\int_a^b h(x)^2(P(x) + w'(x)) + 2h(x)h'(x)(Q(x) + w(x)) + h'(x)^2R(x)dx$. The idea is to use a clever w that will allow us to complete the square. First, we'd like to factor by R to get $\int_a^b R(x)\left(h(x)^2\frac{(P(x)+w'(x))}{R(x)} + 2h(x)h'(x)\frac{(Q(x)+w(x))}{R(x)} + h'(x)^2\right)dx$. This may not be possible on all of $[a, b]$, as R may vanish. This is what leads us to consider the contrapisitive: by assumption and continuity of R, we can find an interval $]c, d[\subseteq [a, b]$ such that $R < 0$ on it, so that we can factor by R on it. As the goal is to find a variation h where the second order condition fails, we can choose h as for the derivation of the Euler-Lagrange equation so as to be non-zero, with it and its derivative vanishing outside of $]c, d[$. For this variation, we can reduce the integral defining $H''(0)$ to the support of the integrand, which is contained in $]c, d[$, so that we may factor by R and get $H''(0) = \int_c^d R(x)\left(h(x)^2\frac{(P(x)+w'(x))}{R(x)} + 2h(x)h'(x)\frac{(Q(x)+w(x))}{R(x)} + h'(x)^2\right)dx$.
Finally, we can complete the square by letting w satisfy $w'(x) = -P + (Q(x) + w(x))^2$, as we'd then have $H''(0) = \int_c^d R(x)\left(h(x)\frac{Q(x)+w(x)}{R(x)} + h'(x)\right)^2 dx$, the sign of which depends on R. There is one subtlety (that escaped Legendre and was pointed out by Lagrange) we must take into account: the fact that $w'(x) = -P(x) + \frac{(Q(x)+w(x))^2}{R(x)}$ defines a solution on all of $[c, d]$. This is not at all obvious when one considers examples such as $y' = 2xy^2$ subject to $y(0) = \frac{1}{b} > 0$ which has solution $y(x) = \frac{1}{b - x^2}$, which is only defined on $]-\sqrt{b}, \sqrt{b}[$, due to blowing up at the boundary. By continuity of the second derivatives of f, Peano's theorem provides the existence of a w such that $w'(x) = -P(x) + \frac{(Q(x)+w(x))^2}{R(x)}$ for some initial condition in $]c, d[$, that is defined only on a subinterval of $]c, d[$, of which we have no guarantee of filling all of $]c, d[$. We can however save the proof with this existence by replacing $]c, d[$ by that subinterval. Indeed, the chronology for determining $]c, d[$ is to find an $x \in]a, b[$ for which $R(x) < 0$, then determine intervals around x where $R < 0$ and the solution to $w'(x) = -P(x) + \frac{(Q(x)+w(x))^2}{R(x)}$ exists, take their intersection, and define the variation h so as to vanish outside of that intersection.
To conclude, note that due to the square in the integrand and monotony of R on the interval, the sign of $H''(0) = \int_c^d R(x)\left(h(x)\frac{Q(x)+w(x)}{R(x)} + h'(x)\right)^2 dx$ is negative, as we desired to show.

30.2 Linear programming in function spaces

The problem:

We which to solve the **functional linear programming problem** $\inf\left(\int_0^1 g(t)u(t)dt\right)$ st. $\int_0^1 f_i(t)u(t)dt = \beta_i$ and $0 \leqslant u(t) \leqslant 1$ ae. on $[0,1]$, for some $g, f_i \in L^1[0,1]$ and $\beta_i \in \mathbb{R}$ with $i \in [m]$, for the variable $u \in L^\infty[0,1]$.

The key point is that the objective and the constraints are linear/affine. We may not have the nice structure of a polyhedral constraint set in infinite dimension, but we may still hope to develop a notion of duality. The idea is that under duality, the number of constraints and variables is interchanged. So for a linear functional optimization problem with finitely many constraints, we hope that the dual is a finite optimization problem. We also hope that strong duality holds, so that one can exploit duality and complementary slacknesss to solve the initial functional problem.

Recall that for integrable $f \in L^1[0,1]$ and bounded $g \in L^\infty[0,1]$, we have a dot-product $f \cdot g = \int_0^1 f(t)g(t)dt$.
In fact, for $f \in \left(L^1[0,1]\right)^d$ and $g \in (L^\infty[0,1])^d$ we have dot-product $f \cdot g = \sum_{i\in[d]} \int_0^1 f_i(t)g_i(t)dt$.
With these and any dot-products, we can generalize the notion of duality as follows:

Abstract duality:

For cones $K \subseteq E$ and $Q \subseteq F$ of vector spaces E and F, equipped with a **duality** $\cdot$, which is a bilinear map $\cdot : E \times F \to \mathbb{R}$ that is never constant in one argument, we define their **dual cones** as $K^* = \{g \in L^\infty[0,1] : f \cdot g \geqslant 0, \forall f \in K\}$ and $Q^* = \left\{f \in L^1[0,1] : f \cdot g \geqslant 0, \forall g \in Q\right\}$.

For cones $K \subseteq L^1[0,1]$ and $Q \subseteq L^\infty[0,1]$, we define their dual cones as
$K^* = \left\{g \in L^\infty[0,1] : f \cdot g = \int_0^1 f(t)g(t)dt \geqslant 0, \forall f \in K\right\}$ and
$Q^* = \left\{f \in L^1[0,1] : f \cdot g = \int_0^1 f(t)g(t)dt \geqslant 0, \forall g \in Q\right\}$ (and similarly for the product spaces).

TO COMPLETE: Barvinok chapter 4

30.3 Optimal control

The general optimal control problem is the following:

Problem:

We're dealing with a dynamical system with parameter $\alpha(t)$, the "control", given by equation $x'(t) = f(x(t), \alpha(t))$ and initial condition $x(0) = x_0$, from time $t = 0$ to time $t = T$. The control takes values in an action set A (which may be finite, or multi-dimensional) and we restrict it to a set $\mathcal{A} \subseteq A^{[0,T]}$ of feasible controls, depending on the problem. One permanent condition on $\mathcal{A}$ is that for all controls in it, the ODE $x'(t) = f(x(t), \alpha(t))$ with $x(0) = x_0$ has a unique solution on $[0, T]$.

We're interested in finding a control α^* that maximises a functional $F(x_\alpha, \alpha)$, where x_α is the solution to the ODE corresponding to α. For example, the functional can have form $F(x_\alpha, \alpha) = \int_0^T r(x_\alpha(t), \alpha(t))dt + g(x_\alpha(T))$.

Time optimal control for linear systems:
In time optimal control, we follow the system until it reaches a certain state and try to minimise the total time spent reaching that state. If our goal is to reach a state in a set of target states $S \subseteq \mathbb{R}^d$ starting from x_0 with some control α, then we restrain the set of feasible controls to $\mathcal{A} \subseteq \{\alpha : \exists t \in \mathbb{R}_+, x_\alpha(t) \in S\}$, which may be empty.
On this set, $\tau_\alpha = \inf(t : x_\alpha(t) \in S)$ is finite (not $+\infty$) and represents the first time we arrive at a target state. We'll then try to minimise τ_α over $\mathcal{A}$, aka maximise τ_α over $\mathcal{A}$.

We will focus on a particular case of this problem. We're dealing with a linear system $\begin{cases} x' = Mx + N\alpha \\ x(0) = x_0 \end{cases}$, for matrices $M \in \mathbb{R}^{n\times n}$ and $N \in \mathbb{R}^{n\times m}$ where the controls are piece-wise constant functions $\alpha : [0, \infty) \to \{-1, 0, 1\}^m$ with at most p jumps. Our target set will be the origin $S = \{0\}$. For such systems, we have a nice analytic solution $x(t) = e^{tM}x_0 + e^{tM}\int_0^t e^{-sM}N\alpha(s)ds$, where e^{tM} is the matrix exponential, which can be computed explicitly using the Jordan normal form.

The first question is if the problem is feasible, that is, if there are controls that let the system reach 0. This is the case when $x_0 = 0$, as we can chose $\alpha(s) = 0$.
Actually since, $x_\alpha(t) = 0 \Leftrightarrow e^{tM}x_0 + e^{tM}\int_0^t e^{-sM}N\alpha(s)ds = 0 \Leftrightarrow x_0 = -\int_0^t e^{-sM}N\alpha(s)ds$, we see that any α yields a feasible starting point, so that there seems to be a wide choice of solutions. The reverse question, whether given an x_0, we can tell if there is a control α leading it to the origin is a much more difficult one.
Note that for a given x_0, the number of controls α and times t for which the system starting at x_0 reaches the origin may not be unique. For the case $p \geqslant 4$, $n = m = 2$, $M = 0$ and $N = I$, and $x_0 = (1, 1)$, for example, all controls $\alpha(s) = \begin{cases} (0,-1) : s \in [0, r] \\ (-1, 0) \ : s \in (r, r+1] \\ (0,-1) : s \in (r+1, 2] \\ (0, 0) \ else \end{cases}$ for some $r \in (0, 1)$ yield trajectories that reach

the origin for all $t \geqslant 2$. Indeed, integration yields $x(t) = (1,1) + \begin{cases} (0,-s) : s \in [0,r] \\ (-s+r,-r) \ : s \in (r,r+1] \\ (-1,-s+(r+1)-r) : s \in (r+1,2] \\ (-1,-1) \ else \end{cases}$.

For all α such that the origin is eventually reached, the first arrival time τ_α is finite. The next question is whether the infinimum τ of these times over all arrival times is reached by a certain α^*. To show this we have to show that there is an α^* verifying $x_0 = -\int_0^\tau e^{-sM} N\alpha^*(s)ds$.

To get it, we consider a (decreasing) sequence t_k and associated α_k, so that the t_k converge to τ. We then cut up $x_0 = -\int_0^\tau e^{-sM} N\alpha_k(s)ds - \int_\tau^{t_k} e^{-sM} N\alpha_k(s)ds$. Note that $\left|\int_\tau^{t_k} e^{-sM} N\alpha_k(s)ds\right| \leqslant \sqrt{n}\int_\tau^{t_k} \left\|e^{-sM}N\right\| ds \leqslant (t_k - \tau)\sqrt{n}\max_{[0,t_0]}\left(\left\|e^{-sM}N\right\|\right)$ (for an appropriate sub-multiplicative matrix norm), so that the second term in the cut will converge to 0. Next, note that a subsequence of the α_k will "converge"! This is due to our restriction to piece-wise constant functions with at most p jumps. If the jumps occur at points $s_{1,k} \leqslant .. \leqslant s_{p,k} \in [0,\tau]$, then by successive extraction (Bolzano-Weierstrass), we can assume them to converge to $s_1 \leqslant .. \leqslant s_p$ respectively. With another layer of successive extractions on the intervals $[s_{i,k}, s_{i+1,k}]$, we can assume the functions to have the same value in $\{-1,0,1\}$ on their respective intervals, in that order. We can then set α^* to have that value on the interval $[s_i, s_{i+1}]$. By cutting up $\int_0^\tau e^{-sM} N\alpha_k(s)ds$ on a grid given by both the s_i and $s_{i,k}$ for large k, we can see that the parts of $[0,\tau]$ where α_k and α^* disagree decreases in total length to 0. So, after all this hand-waving, we can conclude that by taking the limit, $x_0 = -\int_0^\tau e^{-sM} N\alpha^*(s)ds$.

We've tried to give a simplified version of the following result, which requires advanced functional analysis such as Alaoglu's theorem and the Krein-Millman theorem for a weak function space topology.

Bang-Bang-Principle:

For the time optimal control problem with integrable controls $\alpha : [0,\infty) \to [-1,1]^m$, for each x_0 from which the origin can be reached, there is a control $\alpha : [0,\infty) \to \{-1,1\}^m$ achieving that trajectory. Furthermore, the shortest time τ to reach the origin is achieved by a control $\alpha^* : [0,\infty) \to [-1,1]^m$ and therefore also by a control $\alpha^* : [0,\infty) \to \{-1,1\}^m$.

A maximum principle:

30.4 Solutions

Ex.EP:
For a local extremum y, we consider variations $V_h(t) = F(y + t.h)$ where $h \in C^{2n}$ satisfies $h^{(i)}(a) = 0$ and $h^{(i)}(b) = 0$ for $i \in \{0, ..., n-1\}$. The first order condition at $t = 0$ becomes $\int_a^b \sum_{i=0}^{n} h^{(i)} \partial_{i+2} f dx = 0$. We then use repeated integration by parts to get rid of the higher derivatives of h, which we formally prove by induction. The case $n = 1$ is the regular Euler-Lagrange derivation. For the step, assume that we have $\int_a^b \sum_{i=0}^{n} h^{(i)} \partial_{i+2} f dx = \int_a^b h \left(\partial_2 f(y) - \partial_x \partial_3 f(y) + ... + (-1)^n \partial_x^n \partial_{n+2} f\right) dx$. Then by linearity of the integral, we only have to show that $\int_a^b h^{(n+1)} \partial_{n+3} f dx = \int_a^b h (-1)^{n+1} \partial_x^{n+1} \partial_{n+3} f dx$.
By integration by parts, $\int_a^b h^{(n+1)} \partial_{n+3} f dx = \left[h^{(n)} \partial_{n+3} f\right]_a^b - \int_a^b h^{(n)} \partial_x \partial_{n+3} f dx$ and since $h^{(n)}(a) = 0$ and $h^{(n)}(b) = 0$, we get $\int_a^b h^{(n+1)} \partial_{n+3} f dx = -\int_a^b h^{(n)} \partial_x \partial_{n+3} f dx$. Repeating this partial integration on h gets us the desired result.
To conclude the proof, we have to find a "smooth indicator" to replace h with to adapt the arguments from the initial Euler-Lagrange derivation. If we consider the original h from the Euler-Lagrange derivation, we note that the powers h^m and its m first derivatives are polynomials that vanish at a and b, since h does, and are therefore smooth. Thus h^{2n} does the job for this problem.

Ex.ELV:
This time, we consider variations of form $V_H(t) = F(Y + t.H)$ where $H \in C^2$ is a vector of variations. the first order condition becomes $\int_a^b \left(\sum_{i=1}^{n} H_i \partial_{i+1} f_i(x, Y, Y') + H_i' \partial_{i+n+1} f(Y, Y')\right) dx = 0$. By considering H_i to be the variation h of the initial Euler-Lagrange derivation and $H_{j \neq i} = 0$, we recover the Euler-Lagrange equations for each function $\partial_{i+1} f_i(x, Y, Y') - \partial_x \partial_{i+n+1} f(Y, Y') = 0$ for all $i \in [n]$ by following the Euler Lagrange derivation.

Ex.ELMD:
Again, we study a variation $V_h(t) = F(u + t.h)$ for $h \in C^2$ such that $h|_{\partial A} = 0$. The first order condition is $\int_A (h \partial_3 f + \partial_1 h . \partial_4 f + \partial_2 h . \partial_5 f) da = 0$. Instead of integration by parts, we aim to use its 2-variable equivalent, Green's theorem: $\int_A (\partial_1 Q + \partial_2 P) da = \int_{\partial A} (-P, Q) dl$. If h is present in both functions in Greens formula, we can make use of $h|_{\partial A} = 0$.
To make the derivatives of Greens theorem appear, we use the product formula $\partial_x h . \partial_4 f = \partial_x (h . \partial_4 f) - h . \partial_x \partial_4 f$ and $\partial_y h . \partial_5 f = \partial_y (h . \partial_5 f) - h . \partial_y \partial_5 f$ to get $\int_A (h \partial_3 f + \partial_1 h . \partial_4 f + \partial_2 h . \partial_5 f) da = \int_A h (\partial_3 f - \partial_x \partial_4 f - \partial_y \partial_5 f) da + \int_A (\partial_x (h . \partial_4 f) + \partial_y (h . \partial_5 f)) da$. Using Greens theorem on the second integral yields $\int_A (\partial_x (h . \partial_4 f) + \partial_y (h . \partial_5 f)) da = \int_{\partial A} h (-\partial_5 f, \partial_4 f) dl = 0$, as we planned.
Thus the first order condition translates to $\int_A h (\partial_3 f - \partial_x \partial_4 f - \partial_y \partial_5 f) da = 0$. Again, we have to find a "smooth indicator" for h. If we denote by h_o the function used in the original Euler-Lagrange derivation, then $h(x, y) = h_o(x) h_o(y)$ does the job, where the "indication" is over a square.

31 Numerics of differential equations

31.1 Numerical quadrature

31.2 Ordinary differential equations

Differential equations are equations on differentiable functions on a domain D, such as $f(x, y(x), \nabla_x y) = 0$ to be solved for $y : D \to \mathbb{R}$, for example. Solving differential equations has fueled mathematics for a long time. Some equations such as $y'(x) + y(x)^2 = 0$ may have solutions in the form of rational functions, on some maximum domain: in this example, $y = \frac{c}{x}$ on $]0, \infty[$ for constants c ranging in $\mathbb{R}$ parametrising the solution space. Others, such as $y' = y$ may have solutions that can only be represented as power series (except for the 0 solution), as no rational function $\frac{P(x)}{Q(x)}$ can solve the equation, for otherwise $P'(x)Q(x) - P(x)Q'(x) = P(x)Q(x)$, which yields a contradiction on degrees unless $P = 0$. Yet, power series may not converge at points where a solution is still existent. If we seek rational function solutions for $y' - \frac{1}{x+1} = 0$, then these solutions would have to solve the problem on intervals of form $[k, \infty[$ since $\frac{P(x)}{Q(x)}$ has only finitely many poles. But the equation $(x+1)(P'(x)Q(x) - P(x)Q'(x)) = Q(x)^2$ requires P to have the degree of Q, so that $\lim_{x\to\infty} \frac{P(x)}{Q(x)}$ is a constant, contradicting $\lim_{x\to\infty} \int_0^x \frac{1}{t+1} dt = \infty$ (which can be show directly using divergence of the harmonic series). So the solution $\ln(1+x) = \int_0^x \frac{1}{t+1} dt$, which we know isn't a rational function, still can be expressed as a Taylor series around 0. However, one can show that the series diverges for $x \geqslant 1$, despite $\int_0^x \frac{1}{t+1} dt$ still being perfectly defined there.

In the line of increasingly pathological examples, equations such as $y' - e^{-x^2} = 0$, which despite being a simple integration problem, has no solution that can be expressed as a formula involving usual functions, as differential Galois theory tells us.
The horror culminates in equations such as $x^3 y' - 2y = 0$ may have solutions that can't be expressed as power series at particular points. Indeed if we prolong $C : x \mapsto e^{-\frac{1}{x^2}}$ and its derivatives by continuity at 0, we get Cauchy's example of a C^∞ function that isn't its Taylor series at 0, and which solves $x^3 y' - 2y = 0$.

Faced with this ineptitude to get a computable solutions, even when solutions are guaranteed to exist by non-constructive theorems, mathematicians took increasing interest the field of numerical analysis, now generally called numerics.

Euler, Runge-Kutta and multi-step methods:
For problems of the form $y'(x) = f(x, y(x))$ with an initial value $y(x_0) = y_0$ specified, we seek methods to estimate a possible solution $y(x)$ at points other then x_0. In its integral formulation $y(x) = y_0 + \int_{x_0}^x f(t, y(t))dt$, we have quadrature methods for solving integrals at disposal. For example, with the (lower) Rieman sums of form $\int_a^b f(t)dt \approx \frac{1}{h} \sum_{0 \leqslant i \leqslant \lfloor h(b-a) \rfloor} f\left(a + i\frac{1}{h}\right)$ with step h, we can approximate $y(x) = y_0 + h \sum_{0 \leqslant i \leqslant \lfloor h(b-a) \rfloor} f\left(a + i\frac{1}{h}, y\left(a + i\frac{1}{h}\right)\right)$. If we fix step length h, then we have $y(x_0+h) \approx y_0 + hf(x_0, y_0)$, and $y(x_0+2h) \approx y_0 + hf(x_0, y_0) + hf(x_0+h, y(x_0+h)) \approx y_0 + hf(x_0, y_0) + hf(x_0+h, y_0 + hf(x_0, y_0))$ and so on. To keep things simple, we'll consider approximations by a single term Rieman sum, so approximations y_k of $y(x_0 + kh)$ defined by $y_k = y_{k-1} + hf(x_{k-1}, y_{k-1})$ for $k \geqslant 1$ and $x_k = x_0 + kh$: this is the **Euler scheme**.

Another path to walk is keeping $y(x_0 + 2h) \approx y_0 + hf(x_0, y_0) + hf(x_0 + h, y(x_0 + h))$ as heuristic and turning it into approximation scheme $y_k = y_{k-2} + hf(x_{k-2}, y_{k-2}) + hf(x_{k-1}, y_{k-1})$, where we need an

estimate y_1 of $y(x_0+h)$ due to the double recurrence. This is an example of a **multistep method**.

We can also use different quadrature methods. For example, by linear interpolation, one can estimate $\int_a^b f(t)dt \approx (b-a)\left(\frac{1}{2}f(a)+\frac{1}{2}f(b)\right)$ for small intervals $[a,b]$.
Then $y(x_0+h) \approx y_0 + h\left(\frac{1}{2}f(x_0,y_0)+\frac{1}{2}f(x_0+h,y(x_0+h))\right)$. To turn this in an approximation scheme in which we don't have to solve a complicated equation at each step, we'll approximate the inner $y(x_0+h)$ by the Euler scheme, so that $y(x_0+h) \approx y_0 + h\left(\frac{1}{2}f(x_0,y_0)+\frac{1}{2}f(x_0+h,y_0+hf(x_0,y_0))\right)$, and we use approximations $y_k = y_{k-1} + h\left(\frac{1}{2}f(x_{k-1},y_{k-1})+\frac{1}{2}f(x_{k-1}+h,y_{k-1}+hf(x_{k-1},y_{k-1}))\right)$. This is a (one of a family of) **Runge-Kutta schemes**.

We'll now discuss the convergence of one-step methods, which we'll group under schemes of the form $y_{k+1} = y_k + h\phi(y_k,h)$. For example, for the Euler method, $\phi(y_k,h) = f(x_k,y_k)$ (recall that x_k implicitly depends on h).

Consistency and convergence:
The immediate question is if the above approximations are any good.
To do this we will introduce some notions of errors:

Errors:

For a numerical method, we define the **local error** at x with step h to be $le(x) = y(x+h) - \overline{y}$ where y is the solution the the initial value problem and $\overline{y}$ is the first approximation y_1 of the solution at point $x+h$ of the scheme, with initial point $y_0 = y(x)$.
The **global error** at step k for step length h is $ge_k = y(x_0+hk) - y_k$, where y_k is the approximation of the solution at point x_0+hk of the scheme for the IVP, with $y_0 = y(x_0)$.

Local error is easier to study then the global one, and will inform us on the global one via the following notion:

Consistency:

A numerical method is **consistent** on $[x_0,x_e]$ if we have $\max_k\left(\frac{le(x_k)}{h}\right) \xrightarrow[0]{h} 0$, where k ranges over the number of grid-points of a grid of $[x_0,x_e]$ with step-length h. It's consistent with **order** p if $\max_k\left(\frac{le(x_k)}{h}\right)$ is $O(h^p)$ for all h, in the sense that there is a constant C independent of h such that $\left|\max_k\left(\frac{le(x_k)}{h}\right)\right| \leqslant Ch^p$.

The connection is:

Convergence:

We say that a numerical method **converges** if $\max_k(ge_k) \xrightarrow[0]{h} 0$, where k ranges over the number of

grid-points of a grid of $[x_0, x_e]$ with step-length h. It does so with **order** p if there is a constant C independent of h such that $\max_k(ge_k) \leqslant Ch^p$.
COMPLETE

Proof: COMPLETE

Lemma proof:
We show this by induction, using without proof the inequality $1 + \delta \leqslant e^\delta$ for $\delta > 0$.
For the base case $n = 0$, we remark that $e^{n\delta}|x_0| + \dfrac{e^{n\delta} - 1}{\delta}\beta = |x_0| \geqslant |x_0|$, so the base case is true. For the step, assume that $|x_n| \leqslant e^{n\delta}|x_0| + \dfrac{e^{n\delta} - 1}{\delta}\beta$.
Then from $|x_{n+1}| \leqslant (1 + \delta)|x_n| + \beta$, we get $|x_{n+1}| \leqslant (1 + \delta)\left(e^{n\delta}|x_0| + \dfrac{e^{n\delta} - 1}{\delta}\beta\right) + \beta$ by positivity. Next, with $1 + \delta \leqslant e^\delta$ we note that $(1 + \delta)e^{n\delta}|x_0| \leqslant e^{(n+1)\delta}|x_0|$ and $(1 + \delta)\dfrac{e^{n\delta}}{\delta}\beta \leqslant \dfrac{e^{(n+1)\delta}}{\delta}\beta$.
To bound $\left(1 + (1 + \delta)\dfrac{e^{n\delta} - 1}{\delta}\right)\beta$, we use the previous inequality and the fact that $1 - \dfrac{1 + \delta}{\delta} = \dfrac{-1}{\delta}$ to get $\left(1 + (1 + \delta)\dfrac{e^{n\delta} - 1}{\delta}\right)\beta \leqslant \dfrac{e^{(n+1)\delta} - 1}{\delta}\beta$. Combining these bounds yields $|x_{n+1}| \leqslant e^{(n+1)\delta}|x_0| + \dfrac{e^{(n+1)\delta} - 1}{\delta}\beta$, thereby proving the induction step.

Forward Euler incremental is f...
We'd like to remark that the Lipschitz assumption is rather strong, so that the theorem is rather weak for the Euler scheme. For example, these assumptions exclude IVPs of the form $y' = \sum_{i=0}^{d} g_i(x)y^i$ for $d \geqslant 2$ and $g_d(x) \neq 0$ on $[x_0, x_e]$, since for a particular point x with $g_d(x) \neq 0$, this polynomial in y isn't Lipschitz as its derivative is unbounded on $\mathbb{R}$.

31.3 Partial differential equations

31.4 Numerical methods for optimal control

Paper "SeqQuadProg..."

31.5 Solutions

32 Basics in stochastic optimization

32.1 Bandits

Consider a casino with a particular machine, a k-armed bandit (generalizing the real-life one-armed bandit). It has k arms (actions) that we can choose sequentially, where at each step the choice results in a reward X_a, distributed according to some p_a, for action $a \in [k]$. Usually, we don't know the distributions p_a, but we know some constraints on them. Our goal is to find a (possibly randomized) strategy/policy, which is a way of choosing which arm to play at what iteration, that will maximise the expected total reward of the game, if we play it for n turns.

Consider for example the following bandit. We're given a connected graph with two target vertices and are asked to select path between them at each turn. So here, actions/arms are paths between the two vertices, and the action space A, of size k, is the set of these paths. Assume that each edge e is then deleted with propability $\theta_e \in [0,1]$, independently per edge. We win 1 if the path a is possible after the deletions, and 0 otherwise. Hence, X_a is a Bernoulli variable, with parameter $\theta_a = \prod_{e \in a} \theta_e$.

By a **strategy/policy**, we mean a family of distributions $((\pi_{h_i})_{h_i \in H_i})_{i \in [n]}$ where h_i is the history up to turn i, in the sense that $H_i = \{(a_1, x_1, a_2, ..., a_{i-1}, x_{i-1}) : a_j \in A, x_j \in Im(X_{a_j})\}$. So π_{h_i} is a distribution over the actions A that we will use to select a_i, where the previous actions and rewards are recollected in $h_i = (a_1, x_1, a_2, ..., a_{i-1}, x_{i-1})$. Deterministic policies correspond to Dirac-distributed π_{h_i}.

If we denote by $X^{i,\pi}$ the reward at stage i when playin π, maximising $E\left(\sum_{i=1}^{n} X^{i,\pi}\right)$ is easy if we know the distributions p_a for all actions $a \in A$. Indeed, we have $E\left(\sum_{i=1}^{n} X^{i,\pi}\right) = \sum_{i=1}^{n} E\left(X^{i,\pi}\right) \leqslant n \max_{a \in A} E(X_a)$ for all poilcies, and this bound is attained by the policy defined by $\pi_{h_i}(a) = \begin{cases} 1 : a = argmax_{a \in A} E(X_a) \\ 0 : else \end{cases}$ for all histories h_i, aka. the policy of playing the action with the best expected reward at each turn.

The situation becomes more intresting, when we don't know the p_a, so that we can't compute the $E(X_a)$, but we still have some form of information on them. For example, in our graph example, we could only know that the rewards are Bernoulli, but not know their parameters. These froms of constraints are called problems with an **environnement class**.
In such a problem, we often can't optimize over policies, but we can analyse particular policies, and bound their performance for all cases of the environnement class. We'll give an example of this in the next section.

We conclude with a definition that makes some expressions easier:

Regret:

For a bandit with reward distributions p_a, playing n turns with policy π yields **regret** $R_n(\pi, (p_a)) = n \max_{a \in A} E(X_a) - \sum_{i=1}^{n} E\left(X^{i,\pi}\right)$.

Maximising the reward is the same as minimising the regret.

In a Bayesian version of bandits, we may have the knowledge over a distribution p over the parameters, and seek a policy maximising the expected expected reward, where the first expectation ranges over the parameters of the problem, in our running example the θ_a.
We then seek to minimise:

Bayesian regret:

For a distribution p over the parameters $\theta \in D$ of an environment class, the **Bayesian regret** is $BR_n(\pi) = \int_D R_n(\pi, (p_a)(\theta))p(\theta)d\theta$.

32.2 Explore-then-commit algorithm for bandits

Consider the following intuitive strategy for the bandit problem with environnement class. We play each of the k actions for m times, assuming $mk \leqslant n$, and fater the mkth turn, we look at the action that has yielded the best result so far, and play that action for the rest of the rounds. This is an explore-then-commit tactic as in the first phase, we explore the actions by playing them all a sufficient number of times to tell estimate them, and then we commit to the best action we've observed so far, to get high rewards.

Formally, $\pi_{h_i}(a) = \begin{cases} 1 : a = \left\lfloor \frac{i-1}{m} \right\rfloor + 1 \\ 0 : else \end{cases}$ for turns $i \leqslant mk$, where we simplify notation by letting $A = [k]$ directly, and $\pi_{h_j}(a) = \begin{cases} 1 : a = argmax_{a \in A} \left(\sum_{i : \left\lfloor \frac{i-1}{m} \right\rfloor + 1 = a} x_i \right) \\ 0 : else \end{cases}$ for turns $j > mk$, models this strategy.

How good is this strategy ? Can we upper-bound its regret over all θ of the environment class ?
For our running example, we have:

Explore-then-commit for Bernoulli environment:

We'll use the notion of:

Immediate regret:

We define the **instant regret** of action a to be $\Delta_a = \max_{b \in A} E(X_b) - E(X_a)$.

We have $R_n(\pi, (p_a)) = \sum_{a \in A} \Delta_a E(T_a)$, where $T_a = \sum_{i=1}^{n} \chi_{A_i = a}$ counts the number of times action a was used, where A_i is the random variable representing the action played at turn i, so that A_i has distribution π_{h_i}.

Proof: $R_n(\pi, (p_a)) = n \max_{a \in A} E(X_a) - \sum_{i=1}^{n} E\left(X^{i,\pi}\right) = \sum_{i=1}^{n} \left(\max_{a \in A} E(X_a) - E\left(X^{i,\pi}\right) \right)$. By conditioning $E\left(X^{i,\pi}\right) = \sum_{a \in A} E\left(X^{i,\pi} | A_i = a\right) P(A_i = a)$, and rewriting $E\left(X^{i,\pi} | A_i = a\right) = E(X_a)$, we can rearange to get the result (using $\max_{a \in A} E(X_a) = \sum_{a \in A} P(A_i = a) \max_{a \in A} E(X_a)$ and swapping sums along the way).

Proof of the bound: In $R_n(\pi, (p_a)) = \sum_{a \in A} \Delta_a E(T_a)$, we know that

$E(T_a) = m + (n - mk) P\left(a = argmax_{a \in A} \left(\sum_{i : \left\lfloor \frac{i-1}{m} \right\rfloor + 1 = a} X^{i,\pi} \right) \right)$, since a is played m times for certain in the mk first turns, and is only played again if it had the best results so far. The whole problem reduces to bounding the latter probability.

We can reformulate $a = argmax_{a \in A} \left(\sum_{i : \left\lfloor \frac{i-1}{m} \right\rfloor + 1 = a} X^{i,\pi} \right)$ as $\sum_{i : \left\lfloor \frac{i-1}{m} \right\rfloor + 1 = a} X^{i,\pi} \geqslant \sum_{i : \left\lfloor \frac{i-1}{m} \right\rfloor + 1 = b} X^{i,\pi}$ for all $b \in$

A, in particular $b = argmax_{c \in A} E(X_c)$. We bound first with $P\left(a = argmax_{a \in A}\left(\sum_{i:\lfloor \frac{i-1}{m} \rfloor + 1 = a} X^{i,\pi}\right)\right) \leqslant$
$P\left(\sum_{i:\lfloor \frac{i-1}{m} \rfloor + 1 = a} X^{i,\pi} \geqslant \sum_{i:\lfloor \frac{i-1}{m} \rfloor + 1 = b} X^{i,\pi}\right)$.
Next, we rewrite $\sum_{i:\lfloor \frac{i-1}{m} \rfloor + 1 = a} X^{i,\pi} \geqslant \sum_{i:\lfloor \frac{i-1}{m} \rfloor + 1 = b} X^{i,\pi}$ as
$\left(\sum_{i:\lfloor \frac{i-1}{m} \rfloor + 1 = a} X^{i,\pi} - \sum_{i:\lfloor \frac{i-1}{m} \rfloor + 1 = b} X^{i,\pi}\right) - E\left(\sum_{i:\lfloor \frac{i-1}{m} \rfloor + 1 = a} X^{i,\pi} - \sum_{i:\lfloor \frac{i-1}{m} \rfloor + 1 = b} X^{i,\pi}\right) \geqslant \Delta_a$ since $\sum_{i:\lfloor \frac{i-1}{m} \rfloor + 1 = a} X^{i,\pi} = \sum_{i:\lfloor \frac{i-1}{m} \rfloor + 1 = a} X^a$ and similarly for b, and the expectation...
...is a sum of i.d.d variables, which are independent because the A_i are (indeed $P(A_i = a_i \cap A_j = a_j) = \chi_{a_i = supp(\pi_{h_i}), a_j = supp(\pi_{h_j})} = \chi_{a_i = supp(\pi_{h_i})} \chi_{a_j = supp(\pi_{h_j})} = P(A_i = a_i) P(A_j = a_j)$), we may intro...
COMPLETE, FIX, BURN: the Bernoulli case might be doable with Hoeffding's inequality...

COMPLETE: exercise: subgaussian version and chernoff bounds from the bandit book.

32.3 Solutions

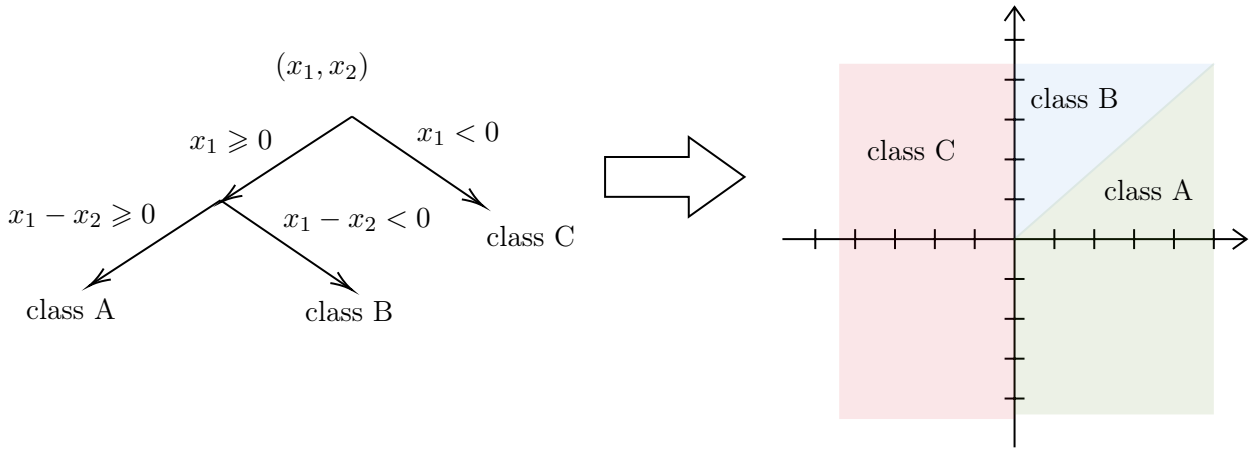
33 Data analysis

33.1 Regression

Regression trees:
We collected n data in the form of real vectors x_i of d data parameters, as well as a real quantity y_i associated to these parameters, that we want to predict, given any d data parameter vector x. Multiple approaches are possible. For example, for a given x, we could return an avareage of the y_i, weighted inversely propertional to the distance of x_i to x. Such approaches are rather computationally demading, especially when the size of the data is large. An alternative approach is to cluster the data first, hoping that we have a computationally easy way of attributing a given input x to a cluster, and estimating y by an average over that cluster. Here, computing the clusters may be computationally demanding, but once it's performed, we hope that estimation is easy.

We collected n data in the form of real vectors x_i of d data parameters, as well as a real quantity y_i associated to these parameters, that we want to predict, given any d data parameter vector x. Multiple approaches are possible. For example, for a given x, we could return an avareage of the y_i, weighted inversely propertional to the distance of x_i to x. Such approaches are rather computationally demading, especially when the size of the data is large. An alternative approach is to cluster the data first, hoping that we have a computationally easy way of attributing a given input x to a cluster, and estimating y by an average over that cluster. Here, computing the clusters may be computationally demanding, but once it's performed, we hope that estimation is easy.

An easy way to classify data points is through **decision trees**. In a binary affine decision tree, we repreatedly ask quesions of form $a_q^t x \geqslant b_q$, where the questions depend on previous answers. For example, the following tree makes three classes, which correspond to half-closed polyhedra (their colsures are polyhedra) that partition the data space.



The polyhedra are obtained by following the path from the root to the class, interecsecting by the corresponding halfspace along each edge. For a tree providing c classes, a total of $O(\log_2(c))$ questions have to be asked to classify the input x, as a binary tree of c leaves has depth at most $\log(c)$. Also, at most $O\left(\log_2(c)^2\right)$ questions of form $a_q^t x \geqslant b_q$ have to be stored, since there are at most $\sum_{l=1}^{\log_2(c)} l = O(\left(\log_2(c)^2\right)$ non-leaf nodes, which may correspond to different questions. So storing this tree and classifying with it if

very computationally simple.

To estimate y given x with such a tree, we first use the tree to classify x, so that x is in half-closed-polyhedron P. Then we estimate for example with $y = \frac{1}{|\{i : x_i \in P\}|} \sum_{i:x_i \in P} y_i$. We can perform the last part with more sophisticated methods, such as $y = \sum_{i:x_i \in P} \left(\frac{\|x_i - x\|}{\sum_{i:x_i \in P} \|x_i - x\|} \right) y_i$. The point is that if we construct a tree such that $|\{i : x_i \in P\}|$ is relatively small for all classes P, then computing this last estimate isn't computationally hard anymore.

The question now is how to produce such a tree in a meaningful way. We will develop such a tree iteratively, adding a question at each iteration, attaching it to a leaf to produce two new leaves at each turn. We will stop the algorithm, stopping such expansions of the tree, once each leaf have less then a specified threshold as size.

To decide which question to ask, we would like that question to split the pre-class data points into two groups, such that the total uniform variance of both groups is minimum. This can be expressed as a quadratic MIP.
We let a and b be the question variables, P be the pre-class polyhedron, and g_i be a binary indicator variable telling the answer to the question. We can have constraints $(1 - g_i) \geqslant x_i^t a - b \geqslant -g_i + \varepsilon$ for all i with $x_i \in P$ for example, for an $1 > \varepsilon > 0$ so that for a question a, b such that $1 \geqslant x_i^t a - b \geqslant -1 + \varepsilon$ for all i with $x_i \in P$, we have $g_i = \begin{cases} 1 : x_i^t a \leqslant b \\ 0 : x_i^t a > b \end{cases}$.
The average values over the groups would be obtained by introducing variables m_t and m_b and constraints $\sum_{i:x_i \in P} (1 - g_i)(y_i - m_t) = 0$ and $\sum_{i:x_i \in P} g_i(y_i - m_b) = 0$, which makes for quadratic constraints We would then optimize over these constraints, with quadratic objective $\sum_{i:x_i \in P} g_i(m_b - y_i)^2 + (1 - g_i)(m_t - y_i)^2$.

We can modify the problem to solve a problem with linear constraints. Indeed, at each step, we can check the parity of $N = |\{i : x_i \in P\}|$ and require that the new classe have exactly $N/2$ data points, when N is even, and some split when N is odd. Then, we can handle averages with $\sum_{i:x_i \in P} (1 - g_i) y_i = \frac{2}{N} m_t$ for example.
In fact, we can place ourselves in the framework of separable convex integer optimization, which we know how to solve with Graver base methods. Note that with average of form $\sum_{i:x_i \in P} (1 - g_i) y_i = \frac{2}{N} m_t$, we can get rid of the average variables by substitution, and get a quadratic form as objective, which is separably convex. We can then seek integer a, b in some range $[-m, m]$ to transform this into a full integer problem, and replace $(1 - g_i) \geqslant x_i^t a - b \geqslant -g_i + \varepsilon$ with $M(1 - g_i) \geqslant x_i^t a - b \geqslant -Mg_i + 1$, for some large M. Note that these problems may now be unfeasible, even when a separating hyperplane exists, since it can't be represented with integer coordintes in $[-m, m]$.

Another restriction is allowing questions of form $e_{j_q}^t x \geqslant b_q$, which makes classification even faster, as we check an inequality on a coordinate per question. This will classify the data space into rectangular boxes. In that case, we can solve the branching problem by ranging over the dimensions j, where each time we sort the jth coordinates of the $x_i \in P$ into values $v_i < ... < v_p$, and seek the minimum total variance obtained by partitioning along $e_j^t x \geqslant v_k$ for k ranging over $[p]$.

INCLUDE: random forests and Breiman's algorithm

33.2 Clustering

k-means-clustering:
In this version of clustering, we consider the data to be n points in euclidean space (n real criteria, distinguished by euclidean distance). We want to classify these points into at most k clusters. Each such cluster would have a mean point. The goal is to find a clustering so that the total distance from the data to their cluster mean is minimum.
To be more precise, we can start with a more general setting by defining indicator variables $c_{ij} \in \{0,1\}$ that indicate if data point $x_i \in \mathbb{R}^d$ is in cluster $j \in [k]$ and centroid variables $m_j \in \mathbb{R}^d$ to represent the centroid of cluster j. We can then try to minimise $\sum_{j=1}^{k}\sum_{i=1}^{n} c_{ij}\|x_i - m_j\|$, the total distance from the data to their cluster centroid, subject to constraints $\sum_{j=1}^{k} c_{ij} = 1$ for all data i.

However, this is a difficult task as this is a non-linear MIP. A simpler approach is decouple the minimisation by minimizing over the c_{ij} and the m_j separately. Indeed, if the m_j are fixed, minimising over the c_{ij} can be done as follows: since we have to chose exactly one of the c_{ij} to be set to 1 for a fixed data i, choosing a cluster from $argmin_j(\|x_i - m_j\|)$ beats all other assignments. Next, when the c_{ij} are fixed, we can try to minimise the objective for the m_j. At this stage, we change the objective to $\sum_{j=1}^{k}\sum_{i=1}^{n} c_{ij}\|x_i - m_j\|^2$, as it allows us to use differential optimisation. Note that $argmin_j(\|x_i - m_j\|) = argmin_j\left(\|x_i - m_j\|^2\right)$ due to monotony of the square, so the previous step is the same.
$\sum_{j=1}^{k}\sum_{i=1}^{n} c_{ij}\|x_i - m_j\|^2$ is convex and has a unique minimum atteined in (solving the first order condition $\sum_{j=1}^{k}\sum_{i=1}^{n} 2c_{ij}(x_{i,q} - m_{j,q}) = 0$ for all dimensions q) $m_j = \frac{\sum_{j=1}^{k}\sum_{i=1}^{n} c_{ij}x_{i,q}}{\sum_{i=1}^{n} c_{ij}}$ (when the size of the cluster is $\sum_{i=1}^{n} c_{ij} > 0$, otherwise we discard the cluster), the mean of the cluster.
The problem is that if we took random initial values for c_{ij} and minimised over m_j for them, then minimised over c_{ij} for this m_j solution, we might have a better possibility for the m_j at this stage, in the sense that it would contribute to a lower objective. The same holds the other way around.
We can repeat the two steps of minimising over the c_{ij} and then the m_j separately. At each step, the objective value is lowered. Since the objective values are positive, the monotonely decreasing sequence of objective values must converge. In fact, it does so in a finite number of steps. This is because there are at most 2^{nk} options for the c_{ij} and if one such configuration appears twice in the sequence $v_1 \geqslant v_2 \geqslant ... \geqslant v_t$, wlog. at 1 and t, where v_t is the objective value after t iterations of minimising over the c_{ij} and then the m_j (which it will because there are finitely many configurations for the c_{ij}), then the m_j of iteration 1 and t must also be the same, since they depend only on the c_{ij}: we get $v_1 = v_k$ and the sequence is constant from there on.

Unfortunately, the decoupling of the optimization problem leads this algorithm to converge to certain "local" optimum, which might not be a global one. Consider as a counterexample the data points $x_1 = \begin{pmatrix}0\\1\end{pmatrix}$ and $x_2 = \begin{pmatrix}0\\-1\end{pmatrix}$ which we which to classify into 2 clusters. The optimum is attained by setting each point in one cluster and letting the centroid be that point (to get objective value 0). If we start with poorly

chosen centroids $m_1 = \begin{pmatrix}1\\0\end{pmatrix}$ and $m_2 = \begin{pmatrix}2\\0\end{pmatrix}$, then the first iteration places both points in the same cluster and sets $m_1 = \begin{pmatrix}0\\0\end{pmatrix}$, and this is already the limit configuration. The same would have happened if we initially put both points in the same cluster and started by optimising the m_j.
INCLUDE: PERFORMANCE GUARANTEE ??

k-spacing-clustering:
In this version of clustering, we only need to be able to compare pairs of data points $\{i, j\}$ with a real weight w_{ij} representing the notion of distance. We can then represent the situation by a complete graph with vertex set the data set and edge weights w_{ij}. We want to partition the vertices/data into k clusters exactly, so that the closest vertices of different clusters are far apart. Formally, this means that we seek partition sets C_r that maximise $\min_{\{r,s\}\in\binom{[k]}{2}, i\in C_r, j\in C_s} (w_{ij})$. The optimal objective value will be a lower bound on the "distance" between any pair of data in different clusters.

I first idea would be to avoid low wight edges to appeat in the objective by placing the endpoints in the same cluster. So an algorithm would be to sort the edges and successively go over them in increasing order, putting the vertices in the same cluster when the edge they are endpoints of appears, until we have k clusters. This sounds a lot like Kruskals MST algorithm (greedy MST): in that context, the clusters correspond to connected components and we perform Kruskal until we reach k components, starting from $|V|$ ones.
Can we use the connection to MSTs to prove a quality result for this algorithm ?
We denote by C_i^* the clusters of the greedy algorithm we developed, and consider an arbitrary different partition C_i. Difference means that one of the C_i^* will intersect two C_r and C_s, for otherwise all C_i^* would have to be contained in only one of the C_i, and since both are partitions, this implies $C_i = C_i^*$. So we have $a \in C_i^* \cap C_r$ and $b \in C_i^* \cap C_s$. We can then consider an edge e of the the path of MST connecting a to b in C_i^*, that has one endpoint in C_r and the other in C_s. The weight of e will be considered in the objective value of the C_i, as its a cluster-crossing edge, so that it's an upper bound on that objective value.
The objective value of C_i^* will be w_{ij}^* where $\{i, j\}$ is the edge of the MST added to achieve $k-1$ components, because Kruskal selects the lowest weight edge between connected components at each step. In particular, w_{ij}^* is greater then the weights of the edges of the MST in the C_i^*. This includes the edge e, so that w_{ij}^* is greater then the objective value of the C_i. So this algorithm actually computes an optimal clustering.

33.3 Principle component analysis

Principal component analysis (PCA):
When we're analysing complicated data, a concern is reducing the dimension of the data without affecting its structure too much. For example, we can ask for an projection to a vectorial line spanned with unit normal vector u with special properties. If we denote the data points by $x_i \in \mathbb{R}^d$, their projection on the line to u is $\left(u^t x_i\right) u$. One criterion for a good such projection would be that the total change between data points is minimum, in the sense that we seek unit vector u such that $\sum_{i \in S} \left\| x_i - \left(u^t x_i\right) u \right\|^2$ is minimum, where we used squared distances as they allow for easier computation. Another criterion would be that a notion of "dispersion" is maximised, so that the reduction in dimension doesn't affect discernability between data too much. One such notion of dispersion is $\sum_{i \in S} \left(u^t x_i - \left(\frac{1}{|S|} \sum_{i \in S} u^t x_i \right) \right)^2$, which measures the variance of the projection.

It turns out that for centered data, that is after translation $y_i = x_i - \frac{1}{|S|} \sum_{i \in S} u^t x_i$ such that $\frac{1}{|S|} \sum_{i \in S} y_i = 0$ and then $\frac{1}{|S|} \sum_{i \in S} u^t y_i = 0$ (linearity), these two notions are equivalent!
Indeed, $\sum_{i \in S} \left\| y_i - \left(u^t y_i\right) u \right\|^2 = \sum_{i \in S} \left(y_i - \left(u^t y_i\right) u\right)^t \left(y_i - \left(u^t y_i\right) u\right) = \sum_{i \in S} \left(y_i^t y_i - 2\left(u^t y_i\right)^2 + \left(u^t y_i\right)^2 u^t u \right)$ and since u is a unit vector and we can ignore constants in the objective, minimising the previous objective is the same as minimising $\sum_{i \in S} -\left(u^t y_i\right)^2$. This in turn is the same as maximising $\sum_{i \in S} \left(u^t y_i\right)^2$ which is $\sum_{i \in S} \left(u^t y_i - \left(\frac{1}{|S|} \sum_{i \in S} u^t y_i \right) \right)^2$ for centered data.

If we devellop $\sum_{i \in S} \left(u^t y_i\right)^2 = \sum_{i \in S} \sum_{a,b \in [d]} u_a y_{ia} y_{ib} u_b$, we see that it coincides with $\sum_{i \in S} u^t y_i y_i^t u = u^t \left(\sum_{i \in S} y_i y_i^t \right) u$.
So we're maximising the quadratic form $u^t Y u$ where $Y = \sum_{i \in S} y_i y_i^t$ is symmetric positive semidefinite over the unit sphere. With a a spectral decomposition of Y, one can see that the maximum is the largest eigenvalue.

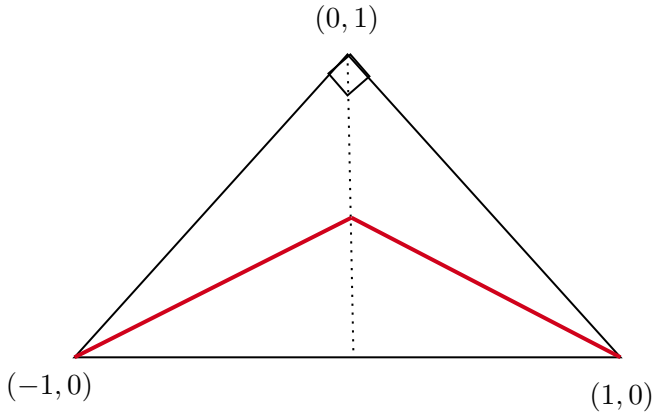
COMPARE AND MERGE THESE TWO SECTIONS.

PCA through projection:
We consider n data given in the form of p-dimensional vectors x_k. The problem is that of compressing the data, so that it requires less then np entries to be stored, in a way that preserves the structure of the data. A first idea is to project the data on an affine subspace of dimension d where d is much smaller then p. The projection of x_k would have form $\mu + V y_k$, where $V = (v_1, ..., v_n) \in \mathbb{R}^{p \times d}$ is a orthonormal base, $\mu \in \mathbb{R}^p$ is a translation and $y_k \in \mathbb{R}^d$ is the vector of scalars in the base V. Here, we'd store $pd + p + nd$ entries, which is much lower then np for $d \leqslant \frac{(n-1)p}{n+p}$. The goal is then to find suitable V, μ and y_k.

We'll look for the V, μ and y_k that minimise the squared norm error $\sum_{k=1}^{n} \|x_k - (\mu + V y_k)\|^2$, which can be thought of as seeking the subspace that's closest to the points x_k. We don't know if this function is convex, so it's not obvious how to optimize it.

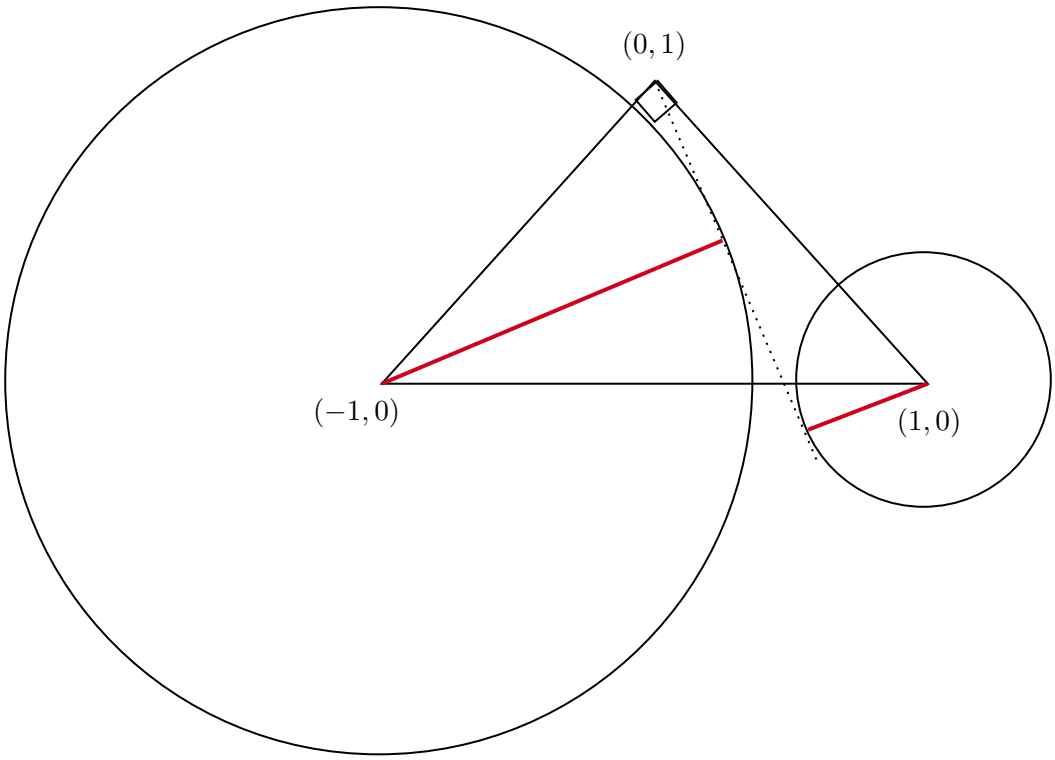
Indeed, note that the constraint on V isn't convex: the "midpoint" of I and $-I$ is not an orthonormal base, in fact it isn't even a base. If we relax the condition on V to be a base, then the problem still isn't convex. Indeed, consider the case where $n = 3$, $p = 2$ and $d = 1$, so that we have 3 data points in the plane $(-1, 0)$, $(1, 0)$ and $(0, 1)$, that we hope to project on a line, so that the sum of square distances of the point to the line is minimized.

We consider the lines given by $(0, 1) + \mathbb{R}(-1, 1)$ and $(0, 1) + \mathbb{R}(1, 1)$, which both contain the data point $(0, 1)$. For both, the objective value is $\left(\sqrt{2}\right)^2 = 2$, where in both cases $y_k = 0$ for all data points except for the ones not projecting to $(0, 1)$, where $y_k = -1$. If we take the convex combination of these variables with coefficient $\frac{1}{2}$, we get line $(0, 1) + \mathbb{R}(0, 1)$. The convex combination of the y_k yield $y_{(0,1)} = 0$, and $y_{(\pm 1,0)} = -\frac{1}{2}$. This corresponds to the distances shown in red in the figure below:



They are bigger then those to the projection of the corresponding points on $(0, 1) + \mathbb{R}(0, 1)$ (shown in dotted lines), which are both 1. So the objective value is greater at the convex combination, contradicting convexity.

Even when we consider the problem where the y_k are paraemerized by μ and V to yield orthogonal projections of points, convexity fails. To see this, you can take a convex combination other then $\frac{1}{2}$, say for example $\left(\frac{1}{4}, \frac{3}{4}\right)$, and the convex combination lines isn't the median/bisector of $(-1, 0)$ and $(1, 0)$, so that the distance don't sum up to two. A proof by picture, where we relate distances with circles:



To see that the sum of squares decreases as well, recall that $a^2+b^2 \leqslant (a+b)^2$ for positive values. So when $a+b$ decreases from $2 = 1^2 + 1^2$, then so does $a^2 + b^2$. Thus, the objective value attains value 2 only at convex combinations $(1, 0)$, $(0, 1)$ and $\left(\frac{1}{2}, \frac{1}{2}\right)$, but achieves lower values "in between" these combinations. This contradicts convexity, as the combination $\left(\frac{1}{2}, \frac{1}{2}\right)$ forms a local maximum.

Faced with a non-convex problem, We will therefore constrain the problem to a case that we understand better.
First, we note that we can write $x_k = \mu + Vy_k + w_k$, where w_k is the orthogonal to the vector space spanned by V, and is obtained from decomposing $x_k - \mu$ in an orthogonal base containing V. We then have the identity $V^t x_k = V^t t + y_k + 0$, as $V^t w_k = 0$ and $V^t V = I$. So $y_k = VV^t(x_k - t)$, and we're optimizing $\sum_{k=1}^{n} \left\| x_k - \left(\mu + VV^t(x_k - \mu)\right)\right\|^2$. For $z_k = x_k - \mu$ fixed, minimising $\sum_{k=1}^{n} \left\| z_k - VV^t z_k \right\|^2$ can be done efficiently. This requires fixing μ. A good choice for it will be to let it be the mean $\mu = \frac{1}{n}\sum_{k=1}^{n} x_k$, so that $\sum_{k=1}^{n} z_k = 0$, which will simplify things later on.

33.4 Classification

Linear classification:
We're given data in form of points of $\mathbb{R}^d$ (d real measurements), each of which belongs to a class $i \in [k]$. We seek a partition of $\mathbb{R}^d$ into k subsets C_i, each containing all points of a class precisely, so that we'll classify new points by assigning them class i if they belong to C_i.
The problem of telling in which C_i a given point $x \in \mathbb{R}^d$ is should be accomplished with an efficient algorithm.

We'll classify using LPs. We start with the case of two classes: we're give n_1 points p_i in class 1 and n_2 points q_j in class 2. The simplest bipartition we can think of is when the points are on different sides of a hyperplane given by equation $a^t x = b$. We can then determine in which class a new point x is by checking if $a^t x > b$ or $a^t x < b$. Of course, it may happen that there is no hyperplane strictly separating the classes: then our particular form of classification isn't adapted to the data.
We can find such a hyperplane or tell that none exist by solving the LP:

$$\max(\varepsilon) : \begin{cases} a^t p_i \geqslant b + \varepsilon, \forall i \in [n_1] \\ a^t q_j \leqslant b - \varepsilon, \forall j \in [n_2] \\ \varepsilon \geqslant 0 \end{cases}$$

If it has a solution with $\varepsilon > 0$, then the classification is possible, otherwise it isn't. The variables are $a \in \mathbb{R}^d$, $b \in \mathbb{R}$ and $\varepsilon \in \mathbb{R}_+$, the first two providing the hyperplane.
.....

Perceptron algorithm:
A slightly restricted version of linear classification can be solved without using a linear programming algorithm. As in the previous section, we're given $n_1 + n_2$ points p_i and q_j in $\mathbb{R}^d$, and our goal is to find a vector w such that $w^t p_i \geqslant 1$ and $w^t q_j \leqslant -1$ to classify points.
Before describing the perceptron algorithm for finding such a w if it exists, we remark that finding such a w is equivalent to finding a v such that $v^t p_i > 0$ and $v^t q_j < 0$. Indeed, if we have w, then $v := w$ does the job, and conversely, if we have v, then $w := \dfrac{1}{\min_{ij}\left(|v^t p_i|, |v^t q_j|\right)} v$ works, as $v^t p_i \geqslant \min\limits_{ij}\left(|v^t p_i|, |v^t q_j|\right)$ and $v^t q_j \leqslant -\min\limits_{ij}\left(|v^t p_i|, |v^t q_j|\right)$.

The perceptron algorithm starts with $w = 0$ and corrects it iteratively by pushing w along the direction of the miss-classified point to adjust the dot product. More precisely, we repeat the adjustment until we have $w^t p_i > 0$ and $w^t q_j < 0$ for all points. In the adjustment phase, we consider the miss-classified point and adjust as follows: if $w^t p_i \leqslant 0$, then $w := w + p_i$ and if $w^t q_j \geqslant 0$, then $w := w - q_j$.
If the algorithm terminates, w will by definition have the require property $w^t p_i > 0$ and $w^t q_j < 0$, so that by our previous discussion, we can normalise it to find the desired output. But does the algorithm terminate if an classifying vector W such that $W^t p_i \geqslant 1$ and $W^t q_j \leqslant -1$ exists ?

In an attempt to study the convergence to W, we'll analyse the behavior of $w^t W$ and $\|w\|$, as these terms appear in $\|w - W\|^2$. Note that at a misclassification at sample s, the new dot product is $(w \pm s)^t W = w^t W \pm s^t W > w^t W + 1$, since W classifies s correctly. So after M iterations, we'll have $w^t W > M$ (as we started with $w = 0$). We next look at $\|w \pm s\|^2 = \|w\|^2 \pm 2 s^t w + \|s\|^2$, so that $\|w \pm s\|^2 \leqslant \|w\|^2 + \max\limits_{ij}\left(\|p_i\|^2, \|q_j\|^2\right)$, since w misclassified s so that $\pm 2 s^t w \leqslant 0$. Thus, after M iterations $\|w\|^2 \leqslant M . \max\limits_{ij}\left(\|p_i\|^2, \|q_j\|^2\right)$.
Using Cauchy-Schwartz, $M^2 \leqslant \left(w^t W\right)^2 \leqslant \|w\|^2 \|W\|^2 \leqslant \|W\|^2 M . \max\limits_{ij}\left(\|p_i\|^2, \|q_j\|^2\right)$, we obtain the bound

on the iterations $M \leqslant \|W\| \sqrt{\max_{ij} (\|p_i\|^2, \|q_j\|^2)}$, proving that the algorithm terminates.

If a classifier W exists, the algorithm outputs a correct classifier which is a sum and difference of the points, scaled by the inverse of the smallest absolute dot product.

It will have form $w' := \dfrac{1}{\min_{ij} \left(|\left(\sum_{i \in I} p_i - \sum_{j \in J} q_j\right)^t p_i|, |\left(\sum_{i \in I} p_i - \sum_{j \in J} q_j\right)^t q_j| \right)} \left(\sum_{i \in I} p_i - \sum_{j \in J} q_j \right)$.

We can then replace W by w' in the analysis of termination of our algorithm to get $M \leqslant \|w'\| \sqrt{\max_{ij} (\|p_i\|^2, \|q_j\|^2)}$.

Theoretically, we can bound $\|w'\|$ by data on the points, so that we could conclude that no classifier W existed in the first place if the algorithm requires more iterations then this bound. The problem is that the finding this explicit bound requires minimizing $\min_{ij} \left(|\left(\sum_{i \in I} p_i - \sum_{j \in J} q_j \right)^t p_i|, |\left(\sum_{i \in I} p_i - \sum_{j \in J} q_j \right)^t q_j| \right)$ over all I and J such that this minimum is non-zero, which is exponential in the number of points if done with brute force.

33.5 Graph partitioning

33.6 Data assimilation

Part 2 and 3 of "Inverse Problems and Data Assimilation" in folder "InversePorblems"

33.7 Solutions

34 Graphical models

34.1 Definitions, examples, properties

Graphical models deal with a finite family of random variables $(X_i)_{i\in[w]}$ taking values from finite sets W_i. Usually, some in-/dependence realtions, as well as some conditional distributions, are known and the task is to find marginal or other conditional distribution, etc. We start by giving a few examples of such graphical models, right after a short example of a type of problem we can model with these tools.

Assume we have a patient who displays observable symptoms of a disease that we'd like to determine. For disease d, we have binary random variable X_d indicating if the patient has disease d, for which we know the distribution, which we'll estimate statistically by letting the probability of the patient having the disease be the ratio of patients that had the disease to the total number of patients. Next, biology will provide us with random variables X_p index by phenomena p that may happen in the body, and conditional distributions $p(X_p|X_{C(p)})$, where $C(p)$ is a set of phenomena we know to collectively impact p. These conditional distributions can be determined from experiments in labs. For example, if X_p is the discretized concentration of a metabolite in the blood-stream, and X_q is a binary variable indicating the presence of an enzyme in some cells (yes, this example is completely made up), then we can let $C(p) = \{q\}$ and $p(X_p = c|X_{C(p)} = (i))$ be the ratio of times we observed concentration c in the blood-stream when the enzyme presence was indicated by i.
Some of these phenomena will be observed when the patient goes to the doctor. For example, we can determine the value of X_p if p is the presence of a certain metabolite in urine. The question is then to determine which disease is the most likely for the patient to have, conditioned of the values observed for the symptoms.
This is a **graphical model** in the sense that we can "represent" it with a graph: if we create vertices for all diseases d and all phenomena p, we can add an edge (p, q) whenever we have $p(X_p|X_{C(p)})$ available, with $q \in C(p)$. In that case, we have $C(p) = \delta^-(p)$. This leads us to the first type of graphical model we'll study:

Bayesian networks:

Bayesian network:

A directed acyclic graph G and a family of conditional distributions $p(X_v|X_{\delta^-(v)})$ (with $p(X_v|X_\emptyset) = p(X_v)$) for $v \in V$ is called a **Bayesian network** if the joint probability distribution $p(X = x)$ can be written as $p(X = x) = \prod_{v\in V} p(X_v = x_v|X_{\delta^-(v)} = x_{\delta^-(v)})$.

We have the following characterization of Bayesian networks, which can also be taken as definition. It gives a better understanding of what a Bayesian network is: it's a graphical model in which we assume (conditional) independence among variables we can't relate in out network.

Characterization:

We're dealing with a Bayesian network $\Leftrightarrow$for all $v \in V$ and $S \subseteq V$ such that there is no directed path from v to any of the elements of S, we have X_v and X_S conditionally independent knowing X_P, where P is the set of vertices without parents, in the sense that $P = \{v \in V : \delta^-(v) = \emptyset\}$.

Proof: TO COMPLETE, also fix proposition ?? If S is an ancestor of v, no conditional independence ?

For a sink s of the graph G of a Bayesian network, $G\backslash s$ is also a bayseian network. (Generalize to sets S with only incoming edges): $p(X_{V\backslash s}) = \sum_{w\in W_s} p(X_s = w, X_{V\backslash s}) = \sum_{w\in W_s} p(X_s = w|X_{\delta^-(s)}) \prod_{v\in V\backslash s} p(X_v|X_{\delta^-(v)}) =$

$$\prod_{v\in V\backslash s} p(X_v|X_{\delta^-(v)}) \left(\sum_{w\in W_s} p(X_s = w|X_{\delta^-(s)}) \right) = \prod_{v\in V\backslash s} p(X_v|X_{\delta^-(v)})$$

Boltzmann machines:
These are sometimes called **pairwise graphical models**.

Boltzmann machines:

The variables X_i form a **Boltzmann machine**, if there is a graph G and a set of functions $\phi_v : W_v \to \mathbb{R}_+$ for $v \in V$ and $\psi_{\{u,v\}} : W_v \times W_u \to \mathbb{R}_+$ for $\{u, v\} \in E$ that are real-valued, positive, and symmetric in the sense that $\psi_{\{u,v\}}(a, b) = \psi_{\{u,v\}}(b, a)$, so that $p(X = x) = \frac{1}{Z} \prod_{v\in V} \phi_v(x_v) \prod_{\{u,v\}\in E} \psi_{\{u,v\}}(x_v, x_u)$, where $Z = \sum_{x\in W_V} \prod_{v\in V} \phi_v(x_v) \prod_{\{u,v\}\in E} \psi_{\{u,v\}}(x_v, x_u)$ is called the **partition function**.

INCLUDE: Ising and Hopfield

Factor graphs:
Factor graphs aren't a graphical models in the way we defined them, but they can still be represented by graphs. The goal is to determine the joint values of random variables X_i for $i \in [w]$, that we assume to depend on a set of factors F. The dependencies can be represented as a bipartite graph on vertices $[w] \cup F$, where edge $\{i, f\}$ expresses dependence of variable i on factor f. The dependence is the following one:

Factor graphs:

We're dealing with a **factor graph**(-ical model) when there is a bipartite graph $([w] \cup F, E)$ and there are positive function $\phi_f : \left(\prod_{i\in\delta(f)} W_i \right) \to \mathbb{R}_+$ such that $p(X = x) = \frac{1}{Z} \prod_{f\in F} \phi_f(x_{\delta(f)})$ where again we have so-called **partition function** $Z = \sum_{x\in W} \prod_{f\in F} \phi_f(x_{\delta(f)})$.

An example of such a model is a simple version of group testing. We let X_i indicate a cause i and F be a set of features f that can be in one of two possible states (observed or not) and depend on the causes. Then feature f is observed if one of the causes i it depends on, in the sense $i \in \delta(f)$, is true, an f is not observed if all causes i it depends on are false. We also assume to have knowledge of the probabilities p_i that X_i is true ($X_i = 1$), and that the X_i are independent.
If we observe $y_F \in \{0, 1\}^F$ for the features, we would like to determine the most likely state of the causes $X = x$ producing these observations. To compute $p(X = x)$, before maximising it, we note that x can be a state producing y precisely if one of the $x_i = 1$ for $i \in \delta(f)$, if $y_f = 1$, and all $x_i = 0$ for $i \in \delta(f)$, if $y_f = 0$. We can inidcate that x can cause y with $\prod_{f:y_f=0} \left(\prod_{i\in\delta(f)} (1 - x_i) \right) \prod_{f:y_f=1} \left(1 - \prod_{i\in\delta(f)} (1 - x_i) \right)$. The

probability of x is $\prod_{i\in[w]} p_i^{x_i}(1-p_i)^{1-x_i}$. By letting Z be the probability of y being observed, we can write it as $Z = \sum_{x\in\{0,1\}^w} \prod_{f:y_f=0} \left(\prod_{i\in\delta(f)}(1-x_i)\right) \prod_{f:y_f=1}\left(1-\prod_{i\in\delta(f)}(1-x_i)\right)\prod_{i\in[w]} p_i^{x_i}(1-p_i)^{1-x_i}$. Then the probability of observing x knowing y is $\frac{1}{Z}\prod_{f:y_f=0} \left(\prod_{i\in\delta(f)}(1-x_i)\right) \prod_{f:y_f=1}\left(1-\prod_{i\in\delta(f)}(1-x_i)\right)\prod_{i\in[w]} p_i^{x_i}(1-p_i)^{1-x_i}$.

This is actually a factor graph model. We first assign variables to features in some arbitrary way, so that each variable is assigned to exactly one feature among its neighbours. Formally, we partition $[w]$ into disjoint set $a(f)$ over $f \in F$, so that $a(f) \subseteq \delta(f)$. We can then define $\phi_f(x_{\delta(f)}) = \prod_{i\in a(f)} p_i^{x_i}(1-p_i)^{1-x_i}\left(\prod_{i\in\delta(f)}(1-x_i)\right)$ for $y_f = 0$ and $\phi_f(x_{\delta(f)}) = \prod_{i\in a(f)} p_i^{x_i}(1-p_i)^{1-x_i}\left(1-\prod_{i\in\delta(f)}(1-x_i)\right)$ for $y_f = 1$. We'll then have $\frac{1}{Z}\prod_{f:y_f=0} \left(\prod_{i\in\delta(f)}(1-x_i)\right) \prod_{f:y_f=1}\left(1-\prod_{i\in\delta(f)}(1-x_i)\right)\prod_{i\in[w]} p_i^{x_i}(1-p_i)^{1-x_i} = \frac{1}{Z}\prod_{f\in F}\phi_f(x_{\delta(f)})$, so that we do indeed deal with a factor graph model.

Markov random fields:

Markov random fields:

The variables X_i for $i \in V$ form a **Markov random field** if there is a graph G with collection $\mathcal{C}$ of cliques $C \subseteq V$ as well as positive functions $\phi_C : \left(\prod_{i\in C} W_i\right) \to \mathbb{R}_+$ such that $p(X = x) = \frac{1}{Z}\prod_{C\in\mathcal{C}}\phi_C(x_C)$, where again we have so-called **partition function** $Z = \sum_{x\in W}\prod_{C\in\mathcal{C}}\phi_C(x_C)$.

Again, we have a more transparent definition of Markov random fields.

Markov random field characterization (Hamersley-Clifford):

COMPLETE

INCLUDE: characterisation from MCMC notes

Partition function positivity:
A question of rigour is that the partition functions used in the previous definitions are non-zero. It turns out that, given the functions ϕ and ψ of the definitions, checking if $Z > 0$ efficiently is an actual problem on its own.

Partition function complexity:

Checking if $Z > 0$ by computing an finding an x such that $p(X = x) > 0$ for a factor graph or a Boltzmann machine is NP-hard.

Proof: We'll reduce SAT to these problems. We consider n random variables, for n variables in the SAT instance, and let all state-spaces be $W_i = \{0, 1\}$, for false and true respectively. For factor graphs, we let factors be the clauses, so that $Z > 0 \Leftrightarrow \exists x, \prod_{f \in F} \phi_f(x_{\delta(f)}) > 0 \Leftrightarrow \exists x, \forall f, \phi_f(x_{\delta(f)}) > 0$. If we let $\delta(f)$ be the variables of clause f, and let $\phi_f(x_{\delta(f)}) = \prod_{i \in \delta(f)} \overline{x_i}$ where $\overline{x_i} = x_i$ if variable i isn't negated in the clause and $\overline{x_i} = 1 - x_i$ if it is, then $\phi_f(x_{\delta(f)}) = 1 > 0$ is equivalent to the clause f being satisfied, the only other value taken by $\phi_f(x_{\delta(f)})$ being 0. So finding such that $p(X = x) > 0$ amounts to finding a truth assignment satisfying the SAT instance for factor graphs.
This same construction can be used to show the result for Boltzmann machines, where the $\phi = 1$, the state space of the factor variables is $W_F = \{1\}$, and $\psi_{\{i,f\}} = \overline{x_i x_f} = \overline{x_i}$ (which is symmetric).

34.2 Inference

Inference of Boltzmann machines on trees:
The problem of inference is, given a set of observations y_O of variable values, to find the most likely values for the other variables of $V \backslash O$, knowing these values. Since $p(X_{V\backslash O} = x_{V\backslash O} | X_O = y_O)$ and $p(X_{V\backslash O} = x_{V\backslash O}, X_O = y_O)$ are proportional for fixed y, our goal is to find the values $x_{V\backslash O}$ that maximise $p(X_{V\backslash O} = x_{V\backslash O}, X_O = y_O)$ (efficiently). This problem is sometimes called **max marginal problem**.

Inference on Bolzmann machines:

The max marginal problem is NP-hard, in full generality.

Proof:
We can reduce max-cut instances to certain max marginal problems. For a max-cut instance, we use the same graph, and set $\phi = 1$ and $\psi_{\{u,v\}}(x_v, x_u) = e^{1-x_u x_v}$ with state spaces $W_v = \{-1, 1\}$. We'll deal with a max marginal instance where $O = \emptyset$, so that we seek maximising $\frac{1}{Z} \prod_{v \in V} \phi_v(x_v) \prod_{\{u,v\} \in E} \psi_{\{u,v\}}(x_v, x_u)$, which is equivalent to maximising (logarithm, constants, positive scaling) $\frac{1}{2} \sum_{\{u,v\} \in E} (1 - x_u x_v)$, which is a max-cut problem if we relate bipartition sets to variables taking values in $\{-1, 1\}$.

We can still find efficient algorithms for specifications of the problem. We'll consider the case of trees. We'll solve this problem with dynamic programming. Boltzmann machines are particularly nice since the distribution is easy to break up.

First, note that we can ignore the scaling $\frac{1}{Z}$ in the optimization problem, which will ease notation.
If we consider some vertex s, then we can write $\prod_{v \in V} \phi_v(x_v) \prod_{\{u,v\} \in E} \psi_{\{u,v\}}(x_v, x_u)$ as $\phi_s(x_s) \prod_{\{s,v\} \in \delta(s)} \psi_{\{s,v\}}(x_v, x_s) \prod_{v \in V \backslash s} \phi_v(x_v) \prod_{\{u,v\} \in E \backslash \delta(s)} \psi_{\{u,v\}}(x_v, x_u)$. To get a recursion, since we can't fully separate the vertex from its neighbours in the function, we could consider $OPT(z_{N(s)})$, the optimal objective value of the the separated $\prod_{v \in V \backslash s} \phi_v(x_v) \prod_{\{u,v\} \in E \backslash \delta(s)} \psi_{\{u,v\}}(x_v, x_u)$ over variables $x_{V\backslash s}$, under the constraint $x_{N(s)} = z_{N(s)}$.

This approach becomes problematic at the next recursive iteration already, since it isn't clear how to split on a further vertex or handle the constraints. It does however work well on trees. For a tree T, deleting s produces $|\delta(s)|$ trees T_j, so that when we number them from 1 to $|\delta(s)|$, we can split $\prod_{v \in V} \phi_v(x_v) \prod_{\{u,v\} \in E} \psi_{\{u,v\}}(x_v, x_u)$ into $\phi_s(x_s) \prod_{\{s,v\} \in \delta(s)} \psi_{\{s,v\}}(x_v, x_s) \prod_{j \in [|\delta(s)|]} \left(\prod_{v \in V_j} \phi_v(x_v) \prod_{\{u,v\} \in E_j} \psi_{\{u,v\}}(x_v, x_u) \right)$.
If we consider the trees to be rooted, then we can introduce $OPT(T, z_r)$, the optimal objective value for tree T over variables $x_{V\backslash r}$, with only $x_r = z_r$ as constraint. Then, if x^* maximises the probability for the max marginal problem, then first $p(X = x^*) \leqslant OPT(T, x_s^*) \leqslant \max_{x_s}(OPT(T, x_s))$. By optimality, this inequality is tight. Next, for any s we have $OPT(T, x_s) \leqslant \phi_s(x_s) \prod_{\{s,v\} \in \delta(s)} \psi_{\{s,v\}}(x_v, x_s) \prod_{j \in N(s)} OPT(T_j, x_j)$,

so $OPT(T, x_s) \leqslant \max_{x_{N(s)}} \left(\phi_s(x_s) \prod_{\{s,v\} \in \delta(s)} \psi_{\{s,v\}}(x_v, x_s) \prod_{j \in N(s)} OPT(T_j, x_j) \right)$ and again, the inequality is tight by optimality.

We thus have the dynamic programming recursion

$$OPT(T, x_s) = \max_{x_{N(s)}} \left(\phi_s(x_s) \prod_{\{s,v\} \in \delta(s)} \psi_{\{s,v\}}(x_v, x_s) \prod_{j \in N(s)} OPT(T_j, x_j) \right)$$

, since this worked for any tree and root. We therefore solve the problem for smaller and smaller trees. It can be improved to $OPT(T, x_s) = \phi_s(x_s) \prod_{j \in N(s)} \max_{x_j} (\psi_{\{s,j\}}(x_j, x_s) OPT(T_j, x_j))$, since the trees T_j are disjoint. This truns out to be crutial to efficiency of the procedure.

A important fact is that the roots appear only in one of the $OPT(T, x_s)$ over all such values to compute, as they aren't contained in the trees of the deeper recursion iterations, since we split on them. So at most $|V|$ value $OPT(T, x_s)$ have to be computed. Computing one such value requires finding at most $\Delta(G)$ maxima, each in at most $\max_{i \in V}(|W_i|)$ time. So a basic bound for the runtime of our algorithm is $O(|V|\Delta(G) \max_{i \in V}(|W_i|))$ or $O\left(|V|^2 \max_{i \in V}(|W_i|)\right)$.

The base case of the recursion is when the trees are single vertices. In that case $OPT(T, x_s) = \max_{x_s} \phi_s(x_s)$. Finally, we deal the constraint on observed variables. We can handle this by solving an instance of the problem for which $W_O = \prod_{v \in O} \{y_v\}$.

To solve the problem in true dynamic programming fashion, note that for an initial s, we can orient T so that the the edge point from s to the vertices. The vertices appear as roots in the recursion in the order induced by this orientation. We would therefore start computing $OPT((\{l\}, \emptyset), x_l)$ for the leaves of the tree T, and move on with decreasing distance to s. Then, we'd conclude with $p\left(X = x^*\right) = \max_{x_s}(OPT(T, x_s))$.

34.3 Solutions

35 Machine learning

35.1 Neural networks basics

Neural networks originated from an attempt to model actual neural activity of our brains. As an input, the neural network receives a vector $x \in \mathbb{R}^{d_{in}}$ of signals, like our brain receives signals from our eyes, ears, etc. It then outputs a response $F(x) \in \mathbb{R}^{d_{out}}$ of signals, like our brain tells our muscles to contract. The neural network does this ... with a network of neurons. We can think of the neurons as the vertices of a digraph, where a neuron v will receive signals form the neurons of $\delta^-(v)$, process them, and transmit them to the neurons of $\delta^+(v)$. At the start, there is a set of d_{in} neurons i are the input neurons, which will transmit signal x_i. Then, if a neuron v receives signals $a_u \in \mathbb{R}$ from the $u \in \delta^-(v)$, it will process this signal. Some neighbours may be more relevant then others, so the we introduce weights $w_{(u,v)}$ that model the importance of neighbour u to v. For biological observation, it was observed that neurons transmit (electrical) signals only if the input signals reach a certain intensity, and with a certain intensity: the neuron "fires". We will let our neurons output signal $a_v = f\left(b_v + \sum_{u \in \delta^-(v)} w_{(u,v)} a_u\right)$ to its neighbours, where f is an **activation function**, such as $f(x) = \begin{cases} 1 : x \geqslant 0 \\ 0 : x < 0 \end{cases}$, if we try to imitate biological neurons, and $b_v \in \mathbb{R}$ is the **bias** that will determine the critical point at which the neuron fires. These signal propagate from the input neurons, along the topological order of the digraph that we assume to be acyclic, until we reach a set of d_{out} output neurons who's signals are meant for the rest of the body, in our biological analogy, and form $F(x)$.

Though neural network aren't yet an accurate model of our brain, they can be used for classification, a task at which they are surprisingly efficient. For example, the input x could be the pixels in RBG values of pictures of animals, and the output neurons could represent the category of the input image, such as dog, cat, hamster, in the sense that we classify the input image according to the highest output signal.

Once we've fixed an activation function f, we would like our choice of weights w and biases b to be set so that classification is done properly. In the analogy to biological neurons, we can take an evolutionary perspective, and argue that through natural selection, the neural networks in our brain are those that did their job "best". This hints at the idea of setting the weights w and biases b so as to optimise a certain objective.

If we have data available, which in our context will come in the form of N points (x_i, y_i), where x_i is an input signal and y_i is the known target value we want the NN to predict for x_i, then we can minimise a **penalty/cost function**, such as the average squared distance $C(w,b) = \frac{1}{N}\sum_{i=1}^{N} \|y_i - F(x_i)\|_2^2$ over weights and biases to get the desired result. In our example, we can let $y_i = e_j$ if the picture given by x_i is animal j.

We point out that many choices of activation and penalty functions exist. One can also use different activation functions at different neurons, or different penalty functions for different data points. However, choosing one such function makes analysis eaysier.

Here are some examples of activation functions:

- Indicator/step-function or McCulloch-Pitts function: $f(x) = \begin{cases} 1 : x \geqslant 0 \\ 0 : x < 0 \end{cases}$, or sometimes it takes values in $\{-1, 1\}$, depending on scources.
- Linear function: $f(x) = kx$ for some $k > 0$.

- Sigmoid: $f(x) = \dfrac{1}{1+e^{-x}}$, which one gets from solving the logistic equation $\begin{cases} f' = f(1-f) \\ f(0) = \frac{1}{2} \end{cases}$, and which has a shape that smoothly approximates the indicator $f(x) = \begin{cases} 1 : x \geqslant 0 \\ 0 : x < 0 \end{cases}$.

- Rectified linear unit (relu): $f(x) = \begin{cases} x : x \geqslant 0 \\ 0 : x < 0 \end{cases}$, which is continuous but not smooth at 0.

Some examples of cost functions are:

- Least square/L^2-error: $C(w,b) = \sum_{i=1}^{N} \|y_i - F(x_i)\|_2^2$ or a positively scaled version of it.

- L^∞-error: $C(w,b) = \max_{i \in [N]} \|y_i - F(x_i)\|$.

Learning:
Minimizing $C(w,b)$ is in general a difficult problem. Note that many efforts to make this a convex minimisation problem by making assumptions on f fail. Convex sums and composition with increasing convex function preserves convexity, but products, which arise in the form $w_{(u,v)}a_u$, don't. We will therefore most-likely be dealing with a non-convex problem.
We can still use gradient descent optimization algorithms for this problem, though we have no guarantee of convergence to a global minimum. If we assume the activation function f differentiable (for example, the sigmoid), then we're faced with the problem of computing the gradient of F, which is used in many optimization algorithms.
Note also that for large data used in the objective, we're in the context that stochastic gradient descent was developed for.

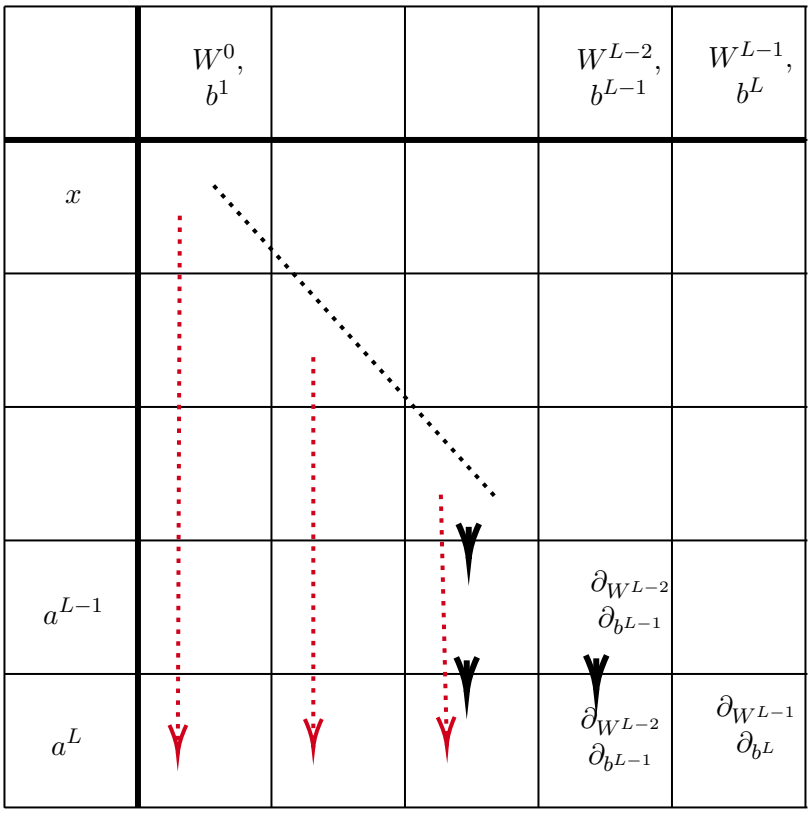
We represent our network in the form of layers $l \in [L]$, where the d_{in} neurons of layer 1 are the input neurons, and the d_{out} neurons of layer L are the output neuron. We number the neurons on each layer from 1 to n_l, where n_l is the number of neurons on that layer. We also assume that for v on layer l, its out-neighbourhood $\delta^+(v)$ is all of the layer $l+1$. We can then index output signals by a_j^l where l is the layer, and $j \in [n_l]$ the neuron on it, where the sup-script isn't a power. The same holds for biases b_j^l and for weights, we use matrix W^l who's entries w_{ij}^l are the weights of neuron i on layer $l+1$ attributed to input-neuron j on layer l. In compact notation, we may then write $a^{l+1} = f\left(W^l a^l + b^{l+1}\right)$ (apply f component-wise, where a and b are vectors with components being signals and biases).
We consider a^l as a function of W^i for $i \leqslant l-1$ and b^j for $j \leqslant l$, but for the use of the chain rule it will be convenient to consider it as a function of W^l, a^l, b^{l+1} (think $a^{l+1}(M,x,y) = f(Mx+y)$).
Note that with our indexing, we care only about layers $\geqslant 2$, so in $a^2 = f\left(W^1 a^1 + b^2\right)$, we set $a^1 = x$, the input data point.
We'll have partial derivatives $\left(\partial_{M_{ij}} a_i^l\right)(M,x,y) = f'(M_{i*}x + y_i)x_j$ and for $k \neq i$, $\left(\partial_{M_{ij}} a_k^l\right)(M,x,y) = 0$, $\left(\partial_{x_j} a_i^l\right)(M,x,y) = f'(M_{i*}x + y_i)M_{ij}$ and $\left(\partial_{y_i} a_i^l\right)(M,x,y) = f'(M_{i*}x + y_i)$ and for $k \neq i$ we have $\left(\partial_{y_i} a_k^l\right)(M,x,y) = 0$.
Now, if we seek derivatives of $a^{l+1} = f\left(W^l a^l + b^{l+1}\right)$ in the coefficients of W^l and b^{l+1}, which don't show up in a^l, we have directly $\left(\partial_{W_{ij}^l} {a_i}^{l+1}\right)(W^l, a^l, b^{l+1}) = f'({W_{i*}}^l a^l + b_i^{l+1})a_j^l = f'(a_i^{l+1})a_j^l$ and $\left(\partial_{b_i^{l+1}} a^{l+1}\right)(W^l, a^l, b^{l+1}) = f'({W_{i*}}^l a^l + b_i^{l+1}) = f'(a_i^{l+1})$ (and the zero cases).
For derivatives in W^r for $r \leqslant l-1$ and b^s for $s \leqslant l$ of a^{l+1}, we have to use the chain rule. Indeed, we

have $\left(\partial_{W_{ij}^r} {a_k}^{l+1}\right)(W^l, ..., b^{l+1}, ...) = f'(W_{k*}{}^l a^l + b_k^{l+1}) \sum_{q=1}^{n_l} W_{kq}^l \left(\partial_{W_{ij}^r} a_q^l\right) = f'(a_i^{l+1}) \sum_{q=1}^{n_l} W_{kq}^l \left(\partial_{W_{ij}^r} a_q^l\right)$ for $r \leqslant l-1$ and

$\left(\partial_{b_i^s} a^{l+1}\right)(W^l, a^l, b^{l+1}) = f'(W_{k*}{}^l a^l + b_k^{l+1}) \sum_{q=1}^{n_l} W_{kq}^l \left(\partial_{b_i^s} a_q^l\right) = f'(a_i^{l+1}) \sum_{q=1}^{n_l} W_{kq}^l \left(\partial_{b_i^s} a_q^l\right)$ for $s \leqslant l$.

We'll now clarify how to use these relations to compute the partial derivatives. First, the values of a^l can are determined by successive evaluation, as they depend only on the a of the previous layers, on what is called a **forward pass**.

In the following tableau, the entry at row l column t will be the values of the partial derivatives of the $\partial_{W_{ij}^{t-1}} a^l$ and $\partial_{b_i^t} a^l$. The values on the diagonal can be combuted directly, knowing only the value of a^l, since they don't make use the the chain rule. The black small arrows indicate the use of the product rule. Indeed, the rule requires the knowledge of the derivative in the same variable on the preceding layer, as well as the signal values a and parameter values.



Finally, we can compute the derivatives of $C(W, b)$, assuming we chose a differentiable cost function such as the least square one. By linearity and minimisation, we can look at $\frac{1}{2} \left\| y_i - a^L \right\|_2^2$ for one data point i only. We then have for any variable v, $(\partial_v C_i)(W, b) = \sum_{j=1}^{d_{out}} \left(\partial_v a_j^L\right) \left(a_j^L - y_{i,j}\right)$.

Optimizing with SGD:

We are precisely in the context to apply stochastic gradient descent, if we make a few assumptions and remarks. To get the similarity condition, which in this context states $E\left(\sum_{j=1}^{d_{out}} \left(\partial_v a_j^L\right) \left(a_j^L - y_{i,j}\right)(W, b)\right) =$

$\left(\frac{1}{|F|}\sum_{i\in F}\sum_{j=1}^{d_{out}}\left(\partial_v a_j^L\right)\left(a_j^L - y_{i,j}\right)\right)(W,b)$, where the distribution is over data points i, we can simply use uniform distribution.
COMPLETE: bound $\left(\partial_v a_j^L\right)\left(a_j^L - y_{i,j}\right)$ and L-smoothness...

A (mixed) integer programming based neural network:

In this section, we'll try to implement the neural network with a step-function $f_b(x) = \begin{cases} 1 : x \geqslant b \\ 0 : x < b \end{cases}$.
We'll do this by rephrasing the learning optimisation problem as a non-linear MIP first, followed by a trick that makes it a linear MIP.

We'll denote an input vector by x, and those of the training data by x^i, where i is the index. Due to the binary nature of activation functions, we'll also assume that the labels we're supposed to predict are vectors y with entries in $\{0,1\}$ (a binary string).

We then denote by $u_{l,k}$ the signal emitted by the kth neuron on the lth layer: we treat it as a variable of the MIP, with constraint $u_{l,k} \in \{0,1\}$. By denoting with W^l the matrix of weights who's entry at (i,j) is the weight associate to the signal to neuron i on the $(l+1)$th layer from neuron j on the lth layer, which are variables of our MIP, the signal arriving at the neuron i on the $(l+1)$th layer is $\left(W^l\right)_{i*} u_l = \sum_{j\in[d_l]} w_{ij}^l u_{l,j}$.

To model the step-function, we use the following "big-M" trick by adding contraints $\begin{cases} \left(W^l\right)_{i*} u_l \leqslant M u_{l+1,i} + b_{l+1,i} \\ \left(W^l\right)_{i*} u_l \geqslant M(u_{l+1,i} - 1) + b_{l+1,i} \end{cases}$. We see that if $\left(W^l\right)_{i*} u_l > b_{l+1,i}$, then the first constraint implies $u_{l+1,i} = 1$ (for $M > 0$), and the second constraint is coherent in that case. Now if $\left(W^l\right)_{i*} u_l < b_{l+1,i}$, the second constraint requires $u_{l+1,i} = 0$, and the first constraint is coherent in that case. We now have to find a suitable M that makes the inequalities coherent. To do this, we'll actually constraint the weight and biases to be in $[-1,1]$. This way, $\left(W^l\right)_{i*} u_l = \sum_{j\in[d_l]} w_{ij}^l u_{l,j} \in [-d_l, d_l]$, so that $\left(W^l\right)_{i*} u_l - b_{l+1,i} \in [-(d_l+1),(d_l+1)]$ and choosing $M = d_l + 1$ does the job.
Note that in the case that $\left(W^l\right)_{i*} u_l = b_{l+1,i}$, both values are possible for $u_{l+1,i}$.

We set these constraints for all neurons. On the first/second layer, we'll have $\begin{cases} \left(W^l\right)_{i*} x \leqslant M u_{1,i} + b_{l+1,i} \\ \left(W^l\right)_{i*} x \geqslant M(u_{1,i} - 1) + b_{l+1,i} \end{cases}$,
where we assume that the input data has entries in $[-1,1]$.
This is for one training data vector. For training data vector x^i, we'll index the variables step-function modelling variables by $u_{l,k}^i$. Weights and biases are the same for all data points. So for L layers of size at most D and I training data, we have $O(LD)$ biases, $O\left(LD^2\right)$ weights, and $O(LDI)$ step-function modelling variables.

To get a linear objective, we'll use the norm $\|.\|_1$ as penalty, since it can be modeled with linear constraints. We use variables z_k^i and constraints $\begin{cases} z_k^i \geqslant u_{L,k}^i - y_k^i \\ z_k^i \geqslant y_k^i - u_{L,k}^i \end{cases}$, and finally set linear objective $\sum_{i\in[I]}\sum_{k\in[d_L]} z_k^i$, to be minimized. This adds $O(DI)$ variables and constraints.

This isn't yet a linear MIP, as we have quadratic terms in the constraints involving $\left(W^l\right)_{i*} u_l = \sum_{j\in[d_l]} w_{ij}^l u_{l,j}$.

We can linearise this with a bunch of tricks. First, we'll use variable $p_{ij}^l \in [-1, 1]$ to model the product $w_{ij}^l u_{l,j}$. Since $u_{l,j} \in \{0, 1\}$, we'll handle the case disjunction on the value of the product via constraints. Indeed, consider $\begin{cases} p_{ij}^l \leqslant u_{l,j} \\ p_{ij}^l \geqslant -u_{l,j} \\ p_{ij}^l \leqslant w_{ij}^l - (u_{l,j} - 1) \\ p_{ij}^l \geqslant w_{ij}^l + (u_{l,j} - 1) \end{cases}$: when $u_{l,j} = 0$, the first two constraints imply $p_{ij}^l = 0$, so that $p_{ij}^l = w_{ij}^l u_{l,j}$ in that case, and the last two constraints are coherent, and in the case that $u_{l,j} = 1$, the last two constraints imply $p_{ij}^l = w_{ij}^l$, so that $p_{ij}^l = w_{ij}^l u_{l,j}$ in that case, and the first two constraints are coherent.
This adds $O\left(LD^2\right)$ variables. We the replace $\left(W^l\right)_{i*} u_l = \sum_{j \in [d_l]} w_{ij}^l u_{l,j}$ by $\sum_{j \in [d_l]} p_{ij}^l$ in the previous constraints.

The size of this MIP can make it impossible to handle in practice. However, it's usable for few, small layers, and little data. This model was developed by Kurtz and Bah and they claim that this architecture beats classical architectures in particular contexts/instances.

35.2 Learnability

We'll now try to answer the question of what neural networks can learn.

Learning Boolean funcitons:
We'll first present a neural network that can simulate a Boolean function. Recall that a Boolean function is a function from $\{True, False\}^n$ to $\{True, False\}$, where n is the number of Boolean variables that are the input of the function. We'll use convetions $True \equiv 1$ and $False \equiv 0$ and the fact that a Boolean can be represented by a DNF to simplify things. A DNF on Boolean variables x_i has form $\vee_{c\in C}(\wedge_{l\in L_c} b_l(x_l)))$, where C is the finite set of clauses, the L_c are finite and the b_l are either the identity or the negation $b_l(x) = \neg x$. We note that in the DNF representation of a Boolean function on n variables, the L_c have size at most n, but C may have size exponential in n.

To model a Boolean function as a neural network, we'll use indicator activation functions $f(x) = \begin{cases} 1 : x > 0 \\ 0 : x \leqslant 0 \end{cases}$.
To find the weights and biases, we note that we can express the connectors $\vee$, $\wedge$ and $\neg$ by neurons. The easyiest is $\neg$, which we can model by neuron $f(1-x)$, so with weight -1 and bias 1. Next, we'll represent $\wedge_{l\in L_c} x_l$ by $f\left(\sum_{l\in L_c} x_l - |L_c| + 1\right)$, as $\sum_{l\in L_c} x_l - |L_c| + 1 > 0$ precisly when all $x_l = 1$, assuming they take value in $\{0,1\}$. So here, weights are all 1 and bias is $-|L_c| + 1$. Finally, we'll model $\vee_{c\in C} y_c$ by $f\left(\sum_{c\in C} y_c - \frac{1}{2}\right)$, as $\sum_{c\in C} y_c - \frac{1}{2} > 0$ precisely if at least one of the y_c is 1, assuming they take value in $\{0,1\}$. So here, weights are all 1 and bias is $\frac{1}{2}$.

Now, we can start constructing our network. The input layer has n nodes, which we'll input values of $\{0,1\}$ to simulate the Boolean function. The first hidden/inner layer will handle negation: for all clauses, we add a neuron for when a variable with negation appears. for the signal transmitted to be $f(1-x)$, we set bias to 1, weight of the edge coming from the corresponding variable to -1, and all other weights of incoming edges to 0. Technically, we can add neurons representing the identity when the variable appears without negation, via $f(x)$, but we can also just directly feed the variable appearing without negation the the second hidden layer. In the second hidden layer, we'll deal with $\wedge$. For each clause c, we add a neuron which will transmit signal $f\left(\sum_{l\in L_c} x_l - |L_c| + 1\right)$, with input edges from the first hidden layer (and depending on our conventions, from the input layer). Finally, we add a third and final layer to handle $\vee$, which is a single output neuron transmitting signal $f\left(\sum_{c\in C} y_c - \frac{1}{2}\right)$, where y_c is the signal emitted by neuron c on the second layer. In total, our network uses at most $(n+1)|C|$ hidden neurons, which may be exponential in n, depending on the DNF representing the Boolean function, to simulate it.

This is a example of what we mean by a neural network being able to learn a function, which here was learning a Boolean function. We do not study the convergence to the weights an biases we've given.

ADD: from Venkatesh's ML notes, learning finite automata, as exercise ? Also the rest of his section...

FIT IN:
We'll show that the XOR-function can't be obtained as a $(2,1)$ network. Indeed, no hyperplane can seper-

ate $(0,0)$ and $(1,1)$ from $(0,1)$ and $(1,0)$. If there where, then both pairs of points would be in different halfspaces, yet the midpoint of the pairs is the same $\left(\frac{1}{2},\frac{1}{2}\right)$, which would have to be in both halfspaces, by convexity.

Learning a class of continuous functions efficiently:
Learning continuous functions is more difficult to describe. We'll describe networks that can approximate continuous functions, and try to relate the size of the network to the quality of the approximation.

The context is that of approximating a continuous function f from a compact set $K \subset \mathbb{R}^n$ with values in $\mathbb{R}^m$. We can reduce this to the task of approximating real valued functions: if we can perform the latter task, we can approximate the coordinate functions f_c for $c \in [m]$ with m neural networks, that we align in parallel, identifying their input layers, and considering the union of output neurons to be the output layer. So we consider $f : K \to \mathbb{R}$.
The notion of distance we'll use for measuring approximation will be the norm $\|f\|^2 = \|f\|_2^2 = \int_K f(x)^2 dx$.

We'll use the following lemma from functional analysis:

An approximation lemma:

We consider a family F of real-valued continuous functions from a compact set $K \subset \mathbb{R}^n$ that is bounded in the sense that $\|h\| \leqslant b$ for all $h \in F$. By denoting with $\overline{F}$ the closure of the convex hull of F, we can approximate an $f \in \overline{F}$ by first choosing choosing a parameter $\beta > b^2 - \|f\|^2$, so that for any $N \geqslant 1$, there exists a convex combination g of at most N functions of F, such that $\|f - g\|^2 < \frac{\beta}{N}$.

Proof: First, we take a function $\overline{f}$ in the convex hull of F that approximates f, in the sense that $\|f - \overline{f}\|^2 < \frac{\varepsilon}{N}$ for ε such that $\beta - b^2 + f\|^2 > 2\varepsilon$, where these choices are for later convenience.
We can write $\overline{f} = \sum_{i=1}^{m} \lambda_i f_i$ where $f_i \in F$ and λ is a simplex point, or said differently, a distribution. We have no control on m, which is the parameter we want to bound by N. We can however try to approximate $\overline{f}$ by a convex combination of at most N functions of F. We'll do this by considering a random function g obtained as a convex combination of N function among the f_i, each chosen independently with probability λ_i, in the hope that such functions are sufficiently close to $\overline{f} = \sum_{i=1}^{m} \lambda_i f_i$. It turns out that we can use a probabilitic method, where we show existence of a convex combination of at most N functions of F with certain properties by bounding some expectation on this class of functions.

We write $g = \frac{1}{N}\sum_{j=1}^{N} g_j$, where g_j is f_i with probability λ_i, and the g_j are chosen independetly. We'll now investigate $E\left(\|g - \overline{f}\|^2\right)$. Since g obeys a finite distribution, there is no problem in performing the switch in $E\left(\|g - \overline{f}\|^2\right) = E\left(\int_K (g(x) - \overline{f}(x))^2 dx\right) = \int_K E\left((g(x) - \overline{f}(x))^2\right) dx$. By noting that $E(g) = \frac{N}{N} E(g_1)$ by identical distribution, so that by choice of distribution for g_j we have $E(g) = E(g_1) = \sum_{i=1}^{m} \lambda_i f_i = \overline{f}$, we see that $E\left((g(x) - \overline{f}(x))^2\right) = Var(g(x))$. By independence, square-linearity of the

variance, and identical distribution, we get $Var(g(x)) = \frac{1}{N} Var(g_1(x))$. Therefore $E\left((g(x) - \overline{f}(x))^2\right) = \frac{1}{N}\left(\sum_{i=1}^{m} \lambda_i f_i(x)^2 - \overline{f}(x)^2\right)$ by definition of the variance and our choice of distribution for g_j. By linearity of the integral, we then get $E\left(\|g - \overline{f}\|^2\right) = \frac{1}{N}\left(\sum_{i=1}^{m} \lambda_i \|f_i\|^2 - \|\overline{f}\|^2\right)$. By bounding $\|f_i\| \leqslant b$ and $-\|\overline{f}\|^2 \leqslant \|f - \overline{f}\|^2 - \|f\|^2$ (FIX: Venkatesh calls it triangular inequality but what about the squares ?), where we recall $\|f - \overline{f}\|^2 < \frac{\varepsilon}{N}$, we then get $E\left(\|g - \overline{f}\|^2\right) \leqslant \frac{1}{N}\left(b^2 - \|f\|^2 + \frac{\varepsilon}{N}\right)$.
Finally, with the "triangle inequality" (FIX v2), and linearity and monotonicity of expecations $E\left(\|g - \overline{f}\|^2\right) \leqslant E\left(\|g - \overline{f}\|^2\right) + E\left(\|f - \overline{f}\|^2\right)$, so that $E\left(\|g - \overline{f}\|^2\right) \leqslant \frac{1}{N}\left(b^2 - \|f\|^2 + \varepsilon + \frac{\varepsilon}{N}\right) < \frac{\beta}{N}$ for our choice of β and ε.
We can now use the probabilistic argument: there must be a convex combination $g' = \frac{1}{N}\sum_{j=1}^{N} f_{i_j}$ for some indices i_j such that $\|g' - \overline{f}\|^2 < \frac{\beta}{N}$, for otherwise, $\|g - \overline{f}\|^2 \geqslant \frac{\beta}{N}$ for all possible values of the random g so that $E\left(\|g - \overline{f}\|^2\right) \geqslant \frac{\beta}{N}$ in expectation, which we've just disproved.
COMPLETE: from Venkatesh

Capacity of a neural network

35.3 Hidden Markov Models

Speach recognition is the task of recieving a sound signal and returning a sentence we estimate most likely for the sound signal to represent. We can try to use statistical learning theory to solve this task. Usually, the sound signal is first pre-processed, before it's interpreted. In this phase, the sound signal is partitioned into a finite sequence of features. Features can be vector valued and based on a pre-processing phase useing Fourier analysis, for example. The important thing for our context is that possible features are part of a finite set.

We'll build an answer based on the following 3 forms of gathered statistical data. The most obvious form of data are the probabilities $p_c(f)$ of letter/word c being expressed by feature f. We can gather this data by letting a wide variety of people pronounce c and observing the feature f they produce, taking the average number of appearances of f as $p_c(f)$. We can also anticipate the following problem: how will a machine make the difference between "I" and "eye", two english words with different meanings and spellings, but with the same pronunciation ? We'll let the machine solve this the same way we do: with context. Indeed, we can deduce from the rest of the sentence these words are used in, which of "I" or "eye" is meant. In the context of HMMs, we'll use the probabilities $t_c(w)$ of letter/word w following letter/word c in the sentence (so our context is only the previous word). This data can be gathered by analysing many texts commonly spoken, noting the times letter/word w followed letter/word c, and taking the average over the times letter/word c appeared. Finally, for probability theory purposes, we'll need the probabilities $\alpha(c)$ of the sentence to be recognized starting with letter/word c. This can also be done by taking averages on texts. We've just described an first example of an HMM:

Hidden Markov Models:

In an **hidden Markov model (HMM)**, we're given a finite set of states S and and finite alphabet of characters A. We select an initial state $s_0 \in S$ according to a distribution α and from there, we repeat the following: we output a character $a \in A$ selected according to probability distribution p_{s_n} over A and move to a new state s_{n+1} according to probability distribution t_{s_n} over S.
The distributions are assumed independent of each other.

In our example, the states are letters/words and the alphabet are the features.

We can now phrase the speach recognition problem as an HMM problem. Given a sequence of characters (features) $a_1, ..., a_n$ from the alphabet, what is the most likely sequence of states (letters/words) $s_1, ..., s_n$ that an HMM would have followed to produce $a_1, ..., a_n$?

First, let's compute the probability of following sequence $s_1, ..., s_n$ and observing $a_1, ..., a_n$, which we'll denote by $P(s_1, ..., s_n; a_1, ..., a_n)$. Since the output a_n given state s_n is independent of the previous states and outputs, and since since state s_n depends only on s_{n-1}, we have recursion $P(s_1, ..., s_n; a_1, ..., a_n) = P(s_1, ..., s_{n-1}, a_1, ..., a_{n-1}) t_{s_{n-1}}(s_n) p_{s_n}(a_n)$. We can apply this recursion until we arrive at s_1, which has probabilty $\alpha(s_1)$ of starting the sequence.

We can now look for the sequence for which $P(s_1, ..., s_n; a_1, ..., a_n)$ is highest, meaning that we wan to solve $\max_{s_1,...,s_n} P(s_1, ..., s_n; a_1, ..., a_n)$. In our example, we search for the sequence of letter/words that is most likely to have produced these features. Brute force enumeration leads us to consider $|S|^n$ cases, which is impossible to perform in practice for speach recognition, where S is an entire language. Instead, we can use a dynamic programming approach:

The Viterbi algorithm:
If we know the most likely subsequence $s_1^*, ..., s_k^*$ for $k < n$, which solves $\max_{s_1,...,s_k} P(s_1, ..., s_k; a_1, ..., a_k)$, how can we deduce the one solving $\max_{s_1,...,s_{k+1}} P(s_1, ..., s_{k+1}; a_1, ..., a_{k+1})$? By writing $P(s_1, ..., s_{k+1}; a_1, ..., a_{k+1}) = P(s_1, ..., s_k, a_1, ..., a_k)t_{s_k}(s_{k+1})p_{s_{k+1}}(a_{k+1})$, we see that the dependence of t on s_k prevents us from doing a direct recursion.

Instead, we'll condition on the two last state: we have look for a recursion in $\max_{s_1,...,s_k} P(s_1, ..., s_k, s; a_1, ..., a_{k+1})$ for a fixed $s \in S$. We have $\max_{s_1,...,s_k} P(s_1, ..., s_k, s; a_1, ..., a_{k+1}) = \max_{r \in S} \max_{s_1,...,s_{k-1}} P(s_1, ..., s_{k-1}, r, s; a_1, ..., a_{k+1})$ so that $\max_{s_1,...,s_k} P(s_1, ..., s_k, s; a_1, ..., a_{k+1}) = \max_{r \in S} \max_{s_1,...,s_{k-1}} P(s_1, ..., s_{k-1}, r, a_1, ..., a_k)t_r(s)p_s(a_{k+1})$. Now the last part makes a recursion appear, a recursion in $L(k, s) = \max_{s_1,...,s_k} P(s_1, ..., s_k, s; a_1, ..., a_{k+1})$, which is $L(k, s) = \max_{r \in S}(L(k-1, r)t_r(s)p_s(a_{k+1}))$. The base of this recursion is $L(0, s) = P(s; a_1) = \alpha(s)p_s(a_1)$.

We can recover $\max_{s_1,...,s_k} P(s_1, ..., s_k; a_1, ..., a_k)$ from the $L(k, s)$ by noting $\max_{s_1,...,s_k} P(s_1, ..., s_k; a_1, ..., a_k) = \max_{s \in S} \max_{s_1,...,s_k} P(s_1, ..., s_{k-1}, s; a_1, ..., a_k)$, so that $\max_{s_1,...,s_k} P(s_1, ..., s_k; a_1, ..., a_k) = \max_{s \in S} L(k-1, s)$. Computing the $O(|S|n)$ values of L takes time $O\left(|S|^2 n\right)$, so this is quite an improvement over brute force.

35.4 Markov Decision Processes

We consider an optimal control problem in which we move from states x_k to states x_{k+1} in a finite state space X according to dynamics given by $f : X \times U \to X$ with $x_{k+1} = f(x_k, u_k)$, where $u_k \in U$ is a parameter in a finite sate of actions U that we can influence at each step. We can let the available actions depend on the state one is in, so that we'd write $U(x_k)$ for the actions available at stage k if on is at state x_k. Further, one could let actions depend on the stage k, so that we'd have sets $U(x_k, k)$.
We start at initial state x_0 and move N times according to the dynamics and our choices of actions until we arrive at state x_N. Each choice of u_k at state x_k incurs a cost $g(x_k, u_k, x_{k+1})$, and at termination, we have cost $g(x_N)$ (we abuse notation by referring to all types of costs by g). The goal is to minimise the total cost $J(u_0, ..., u_{N-1}) = g(x_N) + \sum_{i=0}^{N-1} g(x_k, u_k, x_{k+1})$, where $x_{k+1} = f(x_k, u_k)$.
For example, imagine us playing a game against of adversary who's moves we can accurately predict. For example, we could play the video game pacman against an AI. The states are the collection of positions of pacman, the ghosts and the remaining pallets. Actions are moving pacman up, down, left, right, according to what's available at the current position in the maze, so implicitly according to the state. Costs could be 1 if packman moves to a ghostless paletless spot, 0 if it moves to a ghostless spot with a pallet (from where on all actions lead to states in which this pallet isn't present), and 10 if it encounters a ghost. If we play for N turns, playing well corresponds to finding actions so that the total cost is minimum.

We can solve this problem as a shortest path problem on a directed graph. We introduce a source vertex s representing x_0, and a first layer of vertices representing X. We connect s to vertex $x \in X$ if there is a u_0 so that $x = f(x_0, u_0)$. We then add layers of copies of X as vertices, so as to have N layers is total, and connect vertex x_k in layer k to x_{k+1} in layer $k+1$ along an arc if $x_{k+1} = f(x_k, u_k)$ for some $u_k \in U$. This can be adapted to the cases where actions depend on the state (vertex), and the stage (layer). The edges are given weight $w(x_k, x_{k+1}) = g(x_k, u_k, x_{k+1})$. We then add a target vertex t that is connected from the last layer along edges with weight $g(x_N)$.
Then $J(u_0, ..., u_{N-1}) = g(x_N) + \sum_{i=0}^{N-1} g(x_k, u_k)$ is the cost of an s-t-path in that graph, and minimising it is equivalent to finding a shortest s-t-path in the graph. Note that since all directed s-t-paths have length $N+1$, we can additively modify the costs without modifying the problem, in the following sense. For the case that some costs are negative, or that we which to maximise costs (rewards) by minimising $-J$, which for positive costs now has negative ones, we can set costs to $g - \min(g) \geqslant 0$. The difference in objective value between the problems for the two costs will allways be the constant $(N+1)\min(g)$, so that optimial paths are the same in both problems. We can therefore reduce the problem to one in which weights are positive, so that we may use Dijkstra.

Markov Decision Process:
In the pacman example, we assumed that we knew how the game AI would react at each step for each action. What if we don't have this information ? What if the AI takes random moves according to some probability distribution ?
We now introduce randomization into the problem. Now, choosing action u_k at state x_k leads one to state x_{k+1} with probability $p(x_k, u_k, x_{k+1})$, in the sense that $p(x_k, u_k, \cdot\,)$ is a probability distribution over X. We could let the probabilities depend on the stage k, but we won't discuss this case. We distinguish between two cases: either, we know these probability distributions, so that we deal with a "**model-based**" problem, or we don't (but we assume such a probability distribution exists), so that we deal with a "**model-free**" problem.
Since we don't know at which state we'll be at stage k, we now seek to find a policy given by the $\pi_k(x) \in U$ to play if we're at state x at stage k. With this policy, each sequence of states $x_0, x_1, x_2, ..., x_N$ has prob-

ability $\prod_{k=0}^{N-1} p(x_k, \pi_k(x_k), x_{k+1})$ of occurring, assuming that the transition probabilities are independent, which we do.
We can however go one step further in randomization and choose our actions at random. In that case, we'll let π_{k,x_k} be a probability distribution over U (or $U(x_k)$ or $U(x_k,k)$), so that we select action u to be u_k with probability $\pi_{k,x_k}(u)$, independently of the previous moves. The probability of sequence $x_0, x_1, ..., x_N$ occuring is then $\prod_{k=0}^{N-1} \left(\sum_{u \in U(x_k,k)} \pi_{k,x_k}(u)\ p(x_k, u, x_{k+1}) \right)$, where we condition the probability of moving from x_k to x_{k+1} on that of doing so through action u, with total probability on u.
We then seek to solve the following problems of minimising expected costs:

Markov Decision Process (finite time):

A **Markov decision process (MDP)** on a finite time horizon is the problem of finding a policy π that minimises the expected cost
$J(\pi) = \sum_{x_1,...,x_N \in X} \left(\prod_{k=0}^{N-1} p(x_k, \pi_k(x_k), x_{k+1}) \right) \left(g(x_N) + \sum_{i=0}^{N-1} g(x_k, u_k, x_{k+1}) \right)$ in the case of deterministic policy, and $J(\pi) = \sum_{x_1,...,x_N \in X} \left(\prod_{k=0}^{N-1} \left(\sum_{u \in U(x_k,k)} \pi_{k,x_k}(u) p(x_k, u, x_{k+1}) \right) \right) \left(g(x_N) + \sum_{i=0}^{N-1} g(x_k, u_k, x_{k+1}) \right)$ in the case of randomized policy.

There are many variants of MDPs. For example, one can consider random costs, or an infinite time horizon problem in which the costs get discounted by some factor $\delta \in [0, 1[$, the the sense that costs are $\sum_{i=0}^{\infty} \delta^k g(x_k, u_k, x_{k+1})$ (which converges since there are only finitly many possible costs, so that g is bounded).

Dynamic programming for the finite time problem:
We'll first deal with a finite time MDP with determinisitc costs and policy.
We can take a recursion approach to the problem. The case where $N = 1$ allows just a single move and the problem is to maximise $\sum_{x_1 \in X} p(x_0, \pi_0(x_0), x_1)(g(x_1) + g(x_0, \pi(x_0), x_1))$, which can be solved for π_0 by searching the maximum over $|U(x_0)|$ possible $\pi_0(x_0) \in U(x_0)$.
Assume we have a method to know how to best play for N rounds, for any starting point x_0, say with policy $\pi^*(N, x_0) = \{\pi_0^*, ..., \pi_{N-1}^*\}$: how do we find one for $N+1$ rounds ? The appeal of recursion is that after the first move, we have N moves to go and we already know how to make the best of them. So we will compare these possibilities by searching the minimum cost among playing $\pi_0(x_0) \in U(x_0)$, and then playing $\pi^*(N, x_1)$, if playing $\pi(x_0)$ lead to x_1.

To make the recursion formally appear, we consider
$\sum_{x_1,...,x_N \in X} \left(\prod_{k=0}^{N-1} p(x_k, \pi_k(x_k), x_{k+1}) \right) \left(g(x_N) + \sum_{i=0}^{N-1} g(x_k, \pi_k(x_k), x_{k+1}) \right)$, in which we split the sums as $\sum_{x_1,...,x_N \in X} = \sum_{x_1 \in X} \sum_{x_2,...,x_N \in X}$, which allows us to factor
$\sum_{x_1 \in X} p(x_0, \pi_0(x_0), x_1) \sum_{x_2,...,x_N \in X} \left(\prod_{k=1}^{N-1} p(x_k, \pi_k(x_k), x_{k+1}) \right) \left(g(x_N) + \sum_{i=0}^{N-1} g(x_k, \pi_k(x_k), x_{k+1}) \right)$. To isolate

the first cost at $i=0$, we consider $\sum_{x_2,\dots,x_N\in X}\left(\prod_{k=1}^{N-1}p(x_k,\pi_k(x_k),x_{k+1})\right)\left(g(x_N)+\sum_{i=0}^{N-1}g(x_k,\pi_k(x_k),x_{k+1})\right)$,
in which we distribute the product and swap $\sum_{x_1,\dots,x_N\in X}$ and $\sum_{i=0}^{N-1}$ to get
$\sum_{x_2,\dots,x_N\in X}\left(\prod_{k=1}^{N-1}p(x_k,\pi_k(x_k),x_{k+1})\right)g(x_N)+\sum_{i=0}^{N-1}\sum_{x_2,\dots,x_N\in X}\left(\prod_{k=1}^{N-1}p(x_k,\pi_k(x_k),x_{k+1})\right)g(x_k,\pi_k(x_k),x_{k+1})$.
For the $i=0$ term, we can factor out $g(x_0,\pi(x_0),x_1)$ as it doesn't contain the summation terms $x_2,...,x_N\in X$. We then note that $\sum_{x_2,\dots,x_N\in X}\left(\prod_{k=1}^{N-1}p(x_k,\pi_k(x_k),x_{k+1})\right)=1$ as these are the probabilities of all possible sequences of states, starting from state x_1, of length $N-1$.
Pulling out $g(x_0,\pi_0(x_0),x_1)$ this way yields a cost of $\sum_{x_1\in X}p(x_0,\pi(x_0),x_1)g(x_0,\pi_0(x_0),x_1)+p(x_0,\pi(x_0),x_1)$
$\sum_{x_2,\dots,x_N\in X}\left(\prod_{k=1}^{N-1}p(x_k,\pi_k(x_k),x_{k+1})\right)\left(g(x_N)+\sum_{i=1}^{N-1}g(x_k,\pi_k(x_k),x_{k+1})\right)$, which can be rewritten as
$E_{\pi_0}(x_0)+\sum_{x_1\in X}p(x_0,\pi_0(x_0),x_1)V_{\pi_1,\dots,\pi_{N-1}}(N-1,x_1)$, where $E_{\pi_0}(x_0)=\sum_{x_1\in X}p(x_0,\pi_0(x_0),x_1)g(x_0,\pi_0(x_0),x_1)$
is the expected cost incurred at state x_0 by playing according to π_0, and $V_{\pi_1,\dots,\pi_{N-1}}(N-1,x_1)$ is the cost of the subpropblem of $N-1$ stages, staring at x_1, by playing $\pi_1,...,\pi_{N-1}$. We then have recursion $V_{\pi_0,\dots,\pi_{N-1}}(N,x_0)=E_{\pi_0}(x_0)+\sum_{x_1\in X}p(x_0,\pi_0(x_0),x_1)V_{\pi_1,\dots,\pi_{N-1}}(N-1,x_1)$.

We know that by optimality of $\pi_1^*(N-1,x_1)$, we have $V_{\pi_1,\dots,\pi_{N-1}}(N-1,x_1)\geqslant V_{\pi_1^*,\dots,\pi_{N-1}^*}(N-1,x_1)$ for all policies π. So for all policies, $V_{\pi_0,\dots,\pi_{N-1}}(N,x_0)\geqslant E_{\pi_0}(x_0)+\sum_{x_1\in X}p(x_0,\pi_0(x_0),x_1)V_{\pi_1^*,\dots,\pi_{N-1}^*}(N-1,x_1)$.
Now, by minimising $E_{\pi_0}(x_0)+\sum_{x_1\in X}p(x_0,\pi_0(x_0),x_1)V_{\pi_1^*,\dots,\pi_{N-1}^*}(N-1,x_1)$ over π_0, we get π_0^*, which will satisfy $V_{\pi_0,\dots,\pi_{N-1}}(N,x_0)\geqslant V_{\pi_0^*,\dots,\pi_{N-1}^*}(N,x_0)$ for all policies π.

To analyse our dynamic programing procedure, note that at each step, we compute $|X|$ minima (one per state x_0) over $O(\max_{x\in X}|U(x)|)$ possibilities of $\pi_0(x_0)$, to get the $|X|$ images of π_0^*, based on the $|X|$ values $V_{\pi_1^*,\dots,\pi_{N-1}^*}(N-1,x_1)$ of the previous iteration.

The Bellman equation:
We now discuss that case of an infinite time horizon, again with deterministic costs and policies.
We can think of this problem as having objective value
$\lim_{N\to\infty}\left(\sum_{x_1,\dots,x_N\in X}\left(\prod_{k=0}^{N-1}p(x_k,\pi_k(x_k),x_{k+1})\right)\left(\delta^N g(x_N)+\sum_{i=0}^{N-1}\delta^k g(x_k,\pi_k(x_k),x_{k+1})\right)\right)$ for some discount factor $\delta\in]0,1[$ that results will depend on. The dynamic programming approach doesn't work here, because there is no base case. Generally, the idea of a policy that depends on the stage k is flawed, as we'd have to base it on infinitely further ones. This is why we'll restrain ourselves to seek the best **stationary** policy, which is a policy that's independent of the stage, the the sen that all $\pi_k=\pi_0$.

We can adapt the previous part to get, $V_{\pi_0}(x_0)=E_{\pi_0}(x_0)+\sum_{x_1\in X}p(x_0,\pi_0(x_0),x_1)\delta V_{\pi_0}(x_1)$, where
$V_{\pi_0}(x_0)=\lim_{N\to\infty}\left(\sum_{x_1,\dots,x_N\in X}\left(\prod_{k=0}^{N-1}p(x_k,\pi_0(x_k),x_{k+1})\right)\left(\delta^N g(x_N)+\sum_{i=0}^{N-1}\delta^k g(x_k,\pi_0(x_k),x_{k+1})\right)\right)$ and

$E_{\pi_0}(x_0) = \sum_{x_1 \in X} p(x_0, \pi_0(x_0), x_1) g(x_0, \pi_0(x_0), x_1)$ is the same. Note the appearance of the discount factor in the recursive relation. We therefore get the following linear equations on the objective value:

Bellman equations:

The equations $V_{\pi_0}(x_0) = E_{\pi_0}(x_0) + \delta \sum_{x_1 \in X} p(x_0, \pi_0(x_0), x_1) V_{\pi_0}(x_1)$ over variables $V_{\pi_0}(x)$ for $x \in X$ for a linear system of equaitons called the **Bellman equations**. They determine the objective value for a fixed stationary policy π_0, a discount factor $\delta \in [0, 1[$ uniquely, since this system is invertible.

Proof: The system has form $(I - \delta P)V = E$. By recalling the identity $I = (I - A)\left(I + A + A^2 + ...\right)$ for matrices such that $I + A + A^2 + ...$ converges, we seek the convergence for $A = \delta P$. To see it, note that all entries of all P^n are in $[0, 1]$, which can be see from induction: it's true for $n = 1$ and in a product $P \times A$, the fact that P's lines are probability distributions allows us to get $(P \times A)_{ij} \leqslant \max_k(A_{kj})$, so that for the step $A = P^n$, we get the inductive step.
Therefore, for $A = \delta P$, then entries of $I + A + A^2 + ...$ can be bounded by $1 + \delta + \delta^2 = \dfrac{1}{1 - \delta}$, so that this positive series converges.

Value iteration:
Theoretically, we could solve such a system for all $O\left(|X|^{\max_{x \in X} |U(x)|}\right)$ possible stationary policies, and find the one with the smallest cost for a given starting point. However, we present a different approach.

35.5 Adversarial machine learning

"AdvML" books and papers

35.6 Solutions

36 Games, Mechanisms, Markets

36.1 A game on graphs: cops and robbers

The game of cops and robbers (with k cops) is played by two players on a graph G. In the first round, the 1st player, the police inspector, places k cops on nodes of the graph (possibly the same node). Then, in the second round, the 2nd player, the robber, chooses a vertex of the graph to start on. Next, cops and robbers alternated taking moves, each being able to move all their pieces from vertex to vertex along edges of the graph. The police inspector wins if there is a turn in which he moves one of his cops on a vertex occupied by the robber during his move. Otherwise, the robber wins, meaning that the robber has a strategy that will allow him to evade cops forever. Since there are $(k+1)^{|V|}$ configurations of cops and robbers on the graph, a sequence of moves by the players will eventually return to a visited configuration. So if the police inspector has no strategy to catch the robber in less then $(k+1)^{|V|}$ turns, then the robber wins.

Ex.CRTC: Show that on a tree, $k = 1$ cops is enough for them to win, whereas on a cycle of size $\geqslant 4$ and for $k = 1$ the robber wins.

As the exercise suggests, we're interested in how many cops are necessary, given G, for the cops to win. This number can't exceed $|V|$, as the cops could cover all vertices and win immediately.

Cop number:

For a graph G, the **cop-number** $c(G)$ is the smallest k such that the robber has no winning strategy for the game on G.

We'll now develop a polynomial time algorithm that decides if $c(G) \leqslant k$ for a fixed k. The algorithm is based on dynamic programming and checks if the robber has a winning strategy. The idea is that the strategy only depends on the current configuration of cops and the robber, so that for a given configuration of cops, the robber has a set of vertices from which it has an indefinite evasion strategy. We'll set up a dynamic programming type equation for these sets of vertices by considering their interaction with a change in configuration by the cops.
We consider the "strong power graph" $sp(G)$ of G which will represent the game configurations in a compact form: $V(sp(G))$ is made of k-tuples of vertices of $V(G)$, which represent the positions occupied by the cops. We add an edge between $(u_1, ...u_k), (v_1, ...v_k) \in V(sp(G))$ of the power graph if the cops can move from one configuration to the next, meaning that $(u_i, v_i) \in E(G)$ for all i. We now investigate the positions $f(u) \subseteq V(G)$ for cop configuration $u \in V(sp(G))$ from which the robber can evade the cops indefinitely, if he starts at these vertices on turn 2.

This function f has certain properties, if $c(G) > k$. For starters $\emptyset \neq f(u) \subseteq V(G)$ for all configurations $u \in V(sp(G))$, as otherwise the cops could place themselves in configuration u for which $\emptyset = f(u)$ in turn 1 so that the robber can't win from any starting vertex, meaning that the cops will win, so that $c(G) \leqslant k$. Next, note that $f(u) \subseteq V(G) \backslash \overline{\delta_G}(\{u_1, ..., u_k\})$, meaning that the winning starting vertices of the robber aren't adjacent or occupied by a cop (so $\overline{\delta_G}$ denotes the closed neighbourhood of a set of vertices in G): otherwise, the cops would catch the robber on turn 3. Finally we set up the dynamic programming equation: note that if the cops can move from configuration u to configuration v, then the robber should be able to move to a vertex of $f(v)$, to stay safe, as he can play the indefinite evasion strategy from a vertex of $f(v)$. This translates to $f(u) \subseteq \overline{\delta_G}(f(v))$ for all $(u, v) \in E(sp(G))$: if the robber

is in $f(u)$ and has an indefinite evasion strategy, then when the cops move from u to v, the robber moves according to the strategy to a vertex from which he can keep evading indefinitely, which is therefore in $f(v)$.

Conversely, a function f satisfying $\emptyset \neq f(u) \subseteq V(G)\backslash\overline{\delta_G}(\{u_1, ..., u_k\})$ and $f(u) \subseteq \overline{\delta_G}(f(v))$ for all $u \in V(sp(G))$ and $(u, v) \in E(sp(G))$ can be used by the robber to evade the cops indefinitely.
Indeed, If cops start with configuration $u^{(1)}$ on turn 1, then the robber can start on any vertex in $f\left(u^{(1)}\right) \neq \emptyset$. Induction then shows that the robber can move to the safe vertices in his turn, at each turn, as when the cops move from configuration $u^{(n)}$ to $u^{(n+1)}$, then the robber at $f\left(u^{(n)}\right)$ can move to a vertex of $f\left(u^{(n+1)}\right) \neq \emptyset$, as $f\left(u^{(n)}\right) \subseteq \overline{\delta_G}\left(f\left(u^{(n+1)}\right)\right)$. If the cops ever caught the robber on turn n, then on turn $n-1$, when the robber was at a vertex of $f\left(u^{(n-1)}\right)$, a cop at one of the vertices $u_1^{(n-1)}, ..., u_k^{(n-1)}$ must have been in the neighbourhood $\overline{\delta_G}\left(f\left(u^{(n-1)}\right)\right)$ so as to move to the robbers vertex on the next turn, so that $f\left(u^{(n)}\right) \cap \overline{\delta_G}\left(u_1^{(n-1)}, ..., u_k^{(n-1)}\right) \neq \emptyset$, contradicting $f(u) \subseteq V(G)\backslash\overline{\delta_G}(u_1, ..., u_k)$. So this is a winning strategy for the robber.

Now that we've set up a dynamic programming type equation for a winning strategy of the robber, we need to solve it, showing that $c(G) > k$, or prove that it has no solutions, showing that $c(G) \leqslant k$.
We determine this with an algorithm that constructs a sequence f_i of functions that refine the winning positions for a cop configuration u. We start with $f_0(u) = V(G)\backslash\overline{\delta_G}(u_1, ..., u_k)$, which are the winning starting vertices for a 3 turn game. We construct f_{i+1} from f_i by restraining the sets of winning starting vertices according to one move by the cops. That is, we set $f_{i+1}(u) = f_i(u) \cap \overline{\delta_G}(f_i(v))$ if the cops can move from u to v, so that by starting in $f_{i+1}(u)$, the robber can move to $f_i(v)$ for a move to any v the cops could make, ensuring the robber will survive the next turn too. More precisely, we loop through $(u, v) \in E(sp(G))$ and update $f_{i+1}(u) = f_i(u) \cap \overline{\delta_G}(f_i(v))$ and $f_{i+1}(v) = f_i(v) \cap \overline{\delta_G}(f_i(u))$.
By construction, $f_{i+1}(u) \supseteq f_i(u)$ at each iteration, so that the sequence f_i will eventually stabilize, as one can take only finitely many subsets for finitely many configurations u. At stabilization, we'll have $f_i(u) = f_i(u) \cap \overline{\delta_G}(f_i(v))$, meaning $f_i(u) \subseteq \overline{\delta_G}(f_i(v))$, for all $(u, v) \in E(sp(G))$. So the output verifies the dynamic programming equation. The inclusions $f_{i+1}(u) \supseteq f_i(u)$ and the initial $f_0(u) = V(G)\backslash\overline{\delta_G}(u_1, ..., u_k)$ imply the condition $f_i(u) \subseteq V(G)\backslash\overline{\delta_G}(u_1, ..., u_k)$ at stabilization. Only the condition $f_i(u) \neq \emptyset$ remains for the stabilization to represent a winning strategy. We can check this in our algorithm once stabilization has occurred. In the positive case, we have our desired function, and $c(G) > k$. If we encounter a configuration u for which $f_i(u) = \emptyset$, then we can prove that this implies that no f satisfying the characterisation exists, so that $c(G) \leqslant k$.
Indeed, if a function with the desired properties g exists, then $g(u) \subseteq f_0(u)$ and we'll show inductively that $g(u) \subseteq f_i(u)$: therefore if a configuration u for which $f_i(u) = \emptyset$ is encountered, then g can't exist as $\emptyset \neq g(u) \subseteq f_i(u) = \emptyset$. In the updates, $f_{i+1}(u) = f_i(u) \cap \overline{\delta_G}(f_i(v))$ and by induction $g(u) \subseteq f_i(u)$ and $g(v) \subseteq f_i(v)$, so that $g(u) \subseteq \overline{\delta_G}(g(v)) \subseteq \overline{\delta_G}(f_i(v))$ (the first inclusion is due to the characterisation), hence $g(u) \subseteq f_{i+1}(u)$.

What is the running time of our algorithm ? The updates from f_i to f_{i+1} require looping over the edges of $E(sp(G))$ for which there are at most $\binom{|V(sp(G))|}{2}$, where $|V(sp(G))| = |V(G)|^k$. Computing intersections and neighbourhoods can be done in times $|V(G)|$ and $|E(G)| \leqslant \binom{|V(G)|}{2}$ respectively. So this loop is in $O\left(|V|^{2k+2}\right)$. We perform this loop until f_i stabilizes, which doesn't happen when a $f_i(u)$ decreases by at least a vertex among at most $|V|$, for one of the $|V(G)|^k$ configurations u. So stabilisation must occur in $O\left(|V|^{k+1}\right)$. Therefore the whose procedure is in $O\left(|V|^{3k+3}\right)$, as this dominates initialisa-

tion of f_0 and the checks form emptiness at the end.

The fact that this runtime is exponential in k prevents this algorithm from being used as routine in a binary search on k for the cop number of a graph. In fact, we'll show that it's unlikely that a polynomial time algorithm for this task exists, as finding the cop number is NP-complete.

TO DO:chap 5 of cops and robbers for NP completeness.

36.2 A game on graphs: Shannon's switching game

This game is played on a connected graph G by two players Red and Blue. The players alternatively take turns and color an edge of the graph that hasn't yet been colored, or skip their turn. If at some point the graph spanned by the red edges contains all vertices and is connected, then Red wins the game. Otherwise, if all edges are colored and the graph induced by the red edges isn't connected, Blue wins.

A necessary condition for Red to win is to be able to color a spanning tree: if the graph induced by the red edges spans all vertices and is connected, it must contain a spanning tree. However, if Red attempts to color tree T, it's possible for Blue to color an edge of T and prevent Red from obtaining the desired tree. In that case, Red will have to switch the tree intended to be colored.

Assume Red want s to color a spanning tree T_1 of G. If Blue colors an edge $e \in T_1$, then Red can still fix things by coloring an edge a connecting the two components of $T_1 \backslash e$ in the next turn, if such an edge exists. In that case, the tree Red will intend to color will be $T_1 \backslash e \cup a$, and a at least is already colored. Under what conditions does such an edge a always exist, leading to a winning strategy for Red ?

At this stage, one may think of the tree exchange property (hence the name "switching game"). If there are edge disjoint spanning trees T_1 and T_2 in G, then if Blue colors $e \in T_1$, there must be a unique edge $a \in T_2$ connecting the components of $T_1 \backslash e$. For Red to win, the edges of T_2 should remain uncolored by Blue.

To get a clearer picture, we'll contract the red edges at each turn: if Red has a strategy to find a spanning tree in the contracted graph, it will also have that strategy in the initial graph, as it can color the edge to be contracted, and color one (of the two possible) edges in the initial graph corresponding to an edge to be colored in the strategy on the contracted graph.
We'll show that Red has a winning strategy if G has two edge-disjoint uncolored spanning trees, by induction on the number of vertices, and by contracting edges at each step. If T_1 and T_2 are these spanning trees, then if Blue goes first, we disjoint cases on the edge colored by Blue. If blue colors $e \in T_1$, then Red colors $a \in T_2$ connecting the components of $T_1 \backslash e$. When contracting a, the image of $T_1 \backslash e \cup a$ is still a spanning tree, as possible cycles would have been in the initial tree. So is the image of T_2 and the image trees are disjoint, as the only common edge is the one that was contracted. Red can do the same thing when Blue colors an edge $e \in T_2$, following the same arguments with swapped indices.

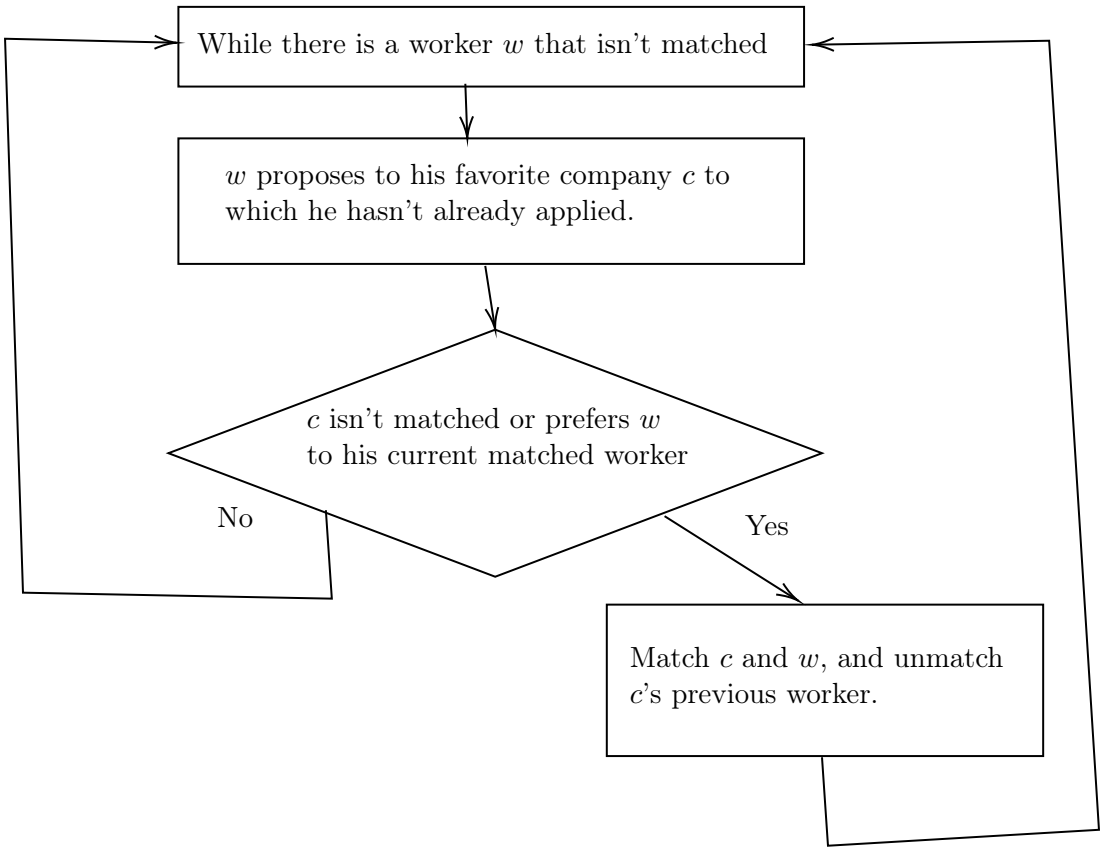
We also have to consider the case in which Blue chooses an edge in none of the trees, or skips turn. In the first case, we note that such an edge f closes a unique cycle in T_1 and that we can replace T_1 by $T_1 \backslash f' \cup f$ where $f' \in T_1$ was an edge on that cycle. The spanning tree $T_1 \backslash f' \cup f$ is still disjoint from T_2, but now Blue has colored an edge of it, so we can apply the same reasoning as before. In the second case, Red can color any edge in $e \in T_1$, but we have to modify T_2 to proceed, as e closes a unique cycle in T_2, so that contracting e creates a cycle in the image of T_2. But we can fix this by just deleting (from our objective tree) any edge of the cycle in the image of T_2, so that the contracted graph still has two edge-disjoint spanning trees.

The base case of this induction is for K_4, as graphs with fewer edges or vertices can't have edge-disjoint spanning trees (we have to satisfy $2(|V|-1) \leqslant \binom{|V|}{2}$). These cases, can be handle with case disjunctions and enumerations.
Note that checking if a graph has two edge disjoint spanning trees can be done with matroid partitioning.

36.3 Stable matchings

In the stable matching problem, we're given n companies and n workers, which we hope to match 1-to-1. The twist is that both the companies and the workers have preferences on their choices: each company c has ranked the the workers w by a strict order, and vice-versa. How do we get a matching that accounts for these preferences ?
One idea is to avoid situations that are inefficient. The case we want to avoid is that of matching companies c and c' and workers w and w' respectively, when both c has preference $w' >_c w$ (it prefers w' to w) and w' has preference $c >_{w'} c'$. If this happens, our matching system would be circumvented by w' and c, leaving us with more possibly dissatisfied matchings. A matching in which this doesn't occurs is called a **stable matching**.

The Gale-Shapely algorithm:
The Gale-Shapely algorithm simulates in a sense the real world. We let the workers apply for their favorite companies, and the companies stick to a worker as long as no prefered one applies:



We make two remarks: over the course of execution, the companies that a worker applies to gets worse in his ranking, and the workers a company is matched with gets better in their ranking.
From the workers perspective, this can be seen from the second box from the top of the flowchart, and for the companies, it's the disjunction box. This will ensure termination, as each worker can change company at most n times, increasing in preferences, and since there are n workers, we have at most n^2 steps. Next, note that all workers are matched at termination, due to the exit condition of the while loop, and since there are $n$companies as well, it's a perfect matching.

Is the final matching a stable one ? Assume for contradiction that there is matching of companies c and c' and workers w and w' respectively, when both c has preference $w' >_c w$ and w' has preference $c >_{w'} c'$. Since w' is matched with c' at termination, this company is the last one he applied to. When he

applied, c already was matched to a preferred worker $w'' >_c w'$, since it rejected w' (flowchart disjunction). Since for companies, the workers get better, and company c ended up with w, we must have $w >_c w''$. By transitivity $w >_c w'$, which we assumed the opposite of.

36.4 Equilibria

So far, we searched an optimal solution in problems in which we could modify all parameters. Game theory studies optimization problems in which other agents may choose parameters that affect our objective value.

Game theory concepts:

We consider m agents/players i, each with a set of parameters (sometimes called a strategy) $p_i \in P_i$ to choose (where P_i is a set from which parameters are chosen from, for example $\mathbb{R}^{d_i}$) and a **utility function** (or payoff) $u_i(p_1, ..., p_m)$ to maximise.

Classic examples are two-player matrix games. Here $m = 2$, player 1 chooses action $i \in [k] = P_1$ and player 2 action $j \in [r] = P_2$, and they receive payoffs $u_1(i, j) = a_{ij}$ and $u_2(i, j) = b_{ij}$ respectively.
In its randomised version, the players choose a probability distribution and receive expected payoffs under the product distribution. Then $P_1 = \left\{ p_1, ..., p_k \in \mathbb{R} : \sum_{i \in [k]} p_i \right\}$ and $P_2 = \left\{ q_1, ..., q_r \in \mathbb{R} : \sum_{i \in [r]} q_i \right\}$ and $u_1(p_1, ..., p_k, q_1, ..., q_r) = \sum_{ij} p_i q_j a_{ij}$ and $u_2(p_1, ..., p_k, q_1, ..., q_r) = \sum_{ij} p_i q_j b_{ij}$.

A first context to be studied is when players give initial parameters and update these parameters sequentially in a greedy way.

More game theory concepts:

A **best reply** of agent i to given choices $(p_j)_{j \in [m] \setminus i}$ of the other agents is a solution to $\max(u_i(p_1, ..., s, ..., p_m))$ st. $s \in P_i$. Ignoring the dynamics of iterated best reply, we prefer studying the equilibria that should arise from them: **Nash equilibria**. Such an equilibrium is achieved in a choice of parameters such that for all agent, their current parameter is a best reply, so that there is no incentive to change parameters. Formally, $\left(p_j^*\right)_{j \in [m]}$ is a Nash equilibrium if for all $i \in [m]$, p_i^* solves $\max\left(u_i\left(p_1^*, ..., s, ..., p_m^*\right)\right)$ st. $s \in P_i$.

For example for the deterministic matrix game with $A = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}$ and $B = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}$, there is a unique Nash equilibrium in slot $(2, 2)$, for $A = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$ and $B = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$, there are no Nash equilibria, and for $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ and $B = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ there are two Nash equilibria in slots in slot $(1, 1)$ and $(2, 2)$, as can be checked by looking for better replies for each player at all the slots.

For deterministic actions, we can characterise and find the Nash equilibria of a two player matrix game rather simply. Indeed, the pair of actions $i \in [k]$ and $j \in [r]$ for a Nash equilibrium when, taking the first players viewpoint, $a_{tj} \leqslant a_{ij}$ for all $t \in [k]$, and, taking the second players viewpoint, $b_{it} \leqslant b_{ij}$ for all $t \in [r]$. This is a direct translation of what it means to have mutual best responses. We can find all equilibria, if any, by computing the maximum entries (and their indices) on the columns of A and the rows of B in $O(kr)$ comparisons and then checking if the maximum on column j is attained for i and the maximum on row i is attained for j, for all such pairs.

For randomised actions, things get much more difficult. How do we even compute best responses ?

If player 2 play strategy q, then the expected payoff of player 1 when playing p is p^tAq. Finding a best response is maximising $(Aq)^tp$ on the distribution simplex $\begin{cases}\sum_{i\in[k]} p_i = 1 \\ p \geqslant 0\end{cases}$. We know from linear programming (or with a little thought) that this maximum will be a convex combination of vertices of the simplex on which the objective is maximum, or said differently, the optimal distribution will have its support in the indices for which $(Aq)_i$ is maximum.

Formally, there is a $u \in \mathbb{R}$ such that $Aq \leqslant u.1$ and such that $p_i > 0 \Rightarrow (Aq)_i = u$. Now, in a Nash equilibrium, this should also hold for player 2, so that there is a $v \in \mathbb{R}$ such that $Bp \leqslant v.1$ and such that $q_i > 0 \Rightarrow (Bp)_i = v$. Conversely, if we can find variables satisfying these conditions, then the distributions form Nash equilibria, since for player 1, any other strategy p' yields objective $p'^tAq \leqslant u\left(\sum_{i\in[k]} p'_i\right) = u = \sum_{i\in supp(p)} up_i = \sum_{i\in supp(p)} (Aq)_i p_i = p^tAq$, so p is a best reply, and likewise for player 2.

We can solve the problem of finding such distributions with an MIP-feasibility problem, by modifying it a bit. For reasons that will become relevant later, we note that the game on $A' = sA + T$ and $B' = sB + T$ for $T = \begin{pmatrix} t & \dots & t \\ \vdots & & \vdots \\ t & \dots & t \end{pmatrix}$ and $s > 0$ has the same best responses as in the original game, as $p^tA'q = sp^tA'q + t$, so that the maximising distributions are the same.
Therefore, by adding T with $t > -\min_i j(a_{ij}, b_{ij})$ and multiplying by $s = \dfrac{1}{\max_{ij}(a_{ij} + t, b_{ij} + t)} > 0$ we can wolg assum that A and B have entries in $[0, 1]$.

To handle the support, we introduce binary indicator variables $x_i, y_i \in \{0, 1\}$ that indicate if i is in the support or not. This can be done with constraints $p_i \leqslant x_i$ and $q_i \leqslant y_i$, which doesn't restrain the possible distributions, as $p_i, q_i \leqslant 1$. We then have $p_i > 0 \Rightarrow x_i = 1$ and $q_i > 0 \Rightarrow y_i = 1$.
Next, we add constraints $0 \leqslant u - A_{i*}q \leqslant 1 - x_i$ and $0 \leqslant v - B_{i*}p \leqslant 1 - y_i$, so that $p_i > 0 \Rightarrow (Aq)_i = u$ and likewise for player 2. Together with the simplex constraints, this MIP-feasibility problem's solution, if any, are Nash equilibria.

The converse is what required the (non-binding) assumption that A and B have entries in $[0, 1]$.
Indeed, for a Nash equilibrium (p, q), the simplex constraints are sarisfied, by setting $x_i = \begin{cases}1 : p_i > 0 \\ 0 : else\end{cases}$ and $y_i = \begin{cases}1 : q_i > 0 \\ 0 : else\end{cases}$ the indicator constraints are satisfied, and finally by setting $u = \max_i((Aq)_i)$ and $v = \max_i((Bp)_i)$, the constraints $0 \leqslant u - A_{i*}q \leqslant 1 - x_i$ and $0 \leqslant v - B_{i*}p \leqslant 1 - y_i$ are satisfied. To see the latter, note that $u = \max_i((Aq)_i) \geqslant (Aq)_i$ and since the entries of A are in $[0, 1]$, $0 \leqslant (Aq)_i \leqslant 1$ (bound a_{ij} and sum distribution) and in particular $u = \max_i((Aq)_i) \in [0, 1]$ so that $u - A_{i*}q \leqslant u \leqslant 1$. Now $0 \leqslant u - A_{i*}q \leqslant 1 - x_i$ when $x_i = 0$, but otherwise, by the Nash equilibrium, we know that $(Aq)_i = u$, so the constraint is still valid.

There are more direct, but still possibly exponential and more confusing algorithms for finding Nash equilibria for two player randomised matrix games, such as the Lemke-Howson algorithm.

We may also consider a different form of best responses:

Security level:

n a two player matrix game, we consider the case in which players hope to maximise the expected worst case outcome. By playing p, the first player expects to get a payoff among the entries of p^tA. If the player's don't know the adversaries strategy, they may be inclined to maximise the worst case outcome. We call the value $\max_p \min_i \left(p^tA\right)$ the first players **security level**.

The securitiy level can be comuted by solving the following LP: $\max v$ st. $\begin{cases} p^tA \geqslant v.1^t \\ p \geqslant 0, 1^tp = 1 \end{cases}$ for player 1 and $\max v$ st. $\begin{cases} Bq \geqslant v.1 \\ q \geqslant 0, 1^tq = 1 \end{cases}$ for player 2. Security levels have the advantage of being easy to compute, compared for Nash equilibria, and easy to generalize to n-player games as well. Another advantage is that security levels are unique, while Nash equilibria aren't.

Another question that arises is that of cooperation. When is it reasonable for players in a n-player game to team up and form coalitions ? In a coalition, a group of players try to as a single player. We consider can consider multiple models for this phenomenon.

In a first scenario, players may "trade" payoffs in the sense that in a coalition of players $S \subseteq [n]$, a coalition contract is given, and for outcome $p_1, ...p_n$, player i receives $f_i \sum_{j \in S} u_j(p_1, ...p_n)$, where the split is given by distribution $f \geqslant 0, 1^tf = 1$. This assumes that utilities can be transferred without additional costs, for example if $u_i = u_j$ for all players i, j.
Since maximising $f_i \sum_{j \in S} u_j(p_1, ...p_n)$ is equivalent to maximising $\sum_{j \in S} u_j(p_1, ...p_n)$, all players have the incentive to maximise this quantity, regardless of the contract f.

In the case of finite sets of actions, one can compute the Nash equilibria by considering the coalition to be a single player, playing actions of form $(p_i)_{i \in S}$ and getting payoffs $\sum_{j \in S} u_j(p_1, ...p_n)$.
We now consider the security level of this coalition. If player $i \in S$ plays distribution $(p_{i,a})_a$ over actions a available to i, then for a set of actions a_k for $k \in [n] \backslash S$, the coalition gets $\left(\prod_{i \in S} p_{i,a_i}\right) \left(\sum_{j \in S} u_j(a_1, ..., a_n)\right)$, assuming that the randomisation for the players are independent. Finding the security level involves solving a non-linear program: $\max v$ st. $\begin{cases} \left(\prod_{i \in S} p_{i,a_i}\right) \left(\sum_{j \in S} u_j(a_1, ..., a_n)\right) \geqslant v \\ p_i \geqslant 0, 1^tp_i = 1 \end{cases}$. In this case, when the randomisations for the players are independent one cannot consider the coalition as acting as a single player, since not all probability distributions over the combinations of actions can be achieved by products. For example, in a coalition of two players each have two actions, the distribution given by $\begin{pmatrix} 0 & 1/2 \\ 1/4 & 1/4 \end{pmatrix}$ can't be achieved as a product since if it where, $p_1q_1 = 0$, so that at least on of $p_1q_2 = 1/2$ or $p_2q_1 = 1/4$ should actually be 0. So the players have to randomise on the combinations of actions for them to truly be considered as a single player.

So at this stage, each coalition S with contract f_S has a security level of v_S, independently of whether the players not in the coalition form coalitions themselves, as the security level is a worst case bound over all combinations of adversarial actions. We can then determine if and which coalitions form assuming that players maximise their security $f_{S,i}v_S$. We can then imagine the following scenario, in which players i choose to declare wanting to build coalition $S \subseteq [n]$, where $i \in S$, in a negotiation phase. If all players in S declare wanting to build S, then each get payoff $f_{S,i}v_S$ in the action phase. Otherwise, then get payoff $f_{\{i\},i}v_{\{i\}}$ for playing without cooperating in the action phase. We're then dealing with a n-player matrix game where actions are choices of declarations $S \subseteq [n]$, and payoffs are $f_{S,i}v_S$ or $f_{\{i\},i}v_{\{i\}}$. The Nash equilibria of this game correspond to declarations of coalitions which are best responses to the declarations of the other players.

We now consider the case in which players may not share payoffs. By randomising collectively, we can maximise the minimum security of the players by solving $\max v$ st. $\begin{cases} p_{(a_i)_{i\in S}} u_j(a_1, ..., a_n) \geqslant v \\ p \geqslant 0, 1^t p = 1 \end{cases}$ where the first constraints range over the all players j and all actions of non-colluding players a_k for $k \in [n] \backslash S$. We then get the same scenario as before, except that the payoffs of the delcaration game are now v_S instead of $f_{S,i}v_S$.

Finally, we remark that despite solving LPs as a routine, we have constraints exponential in the number of players (the producs of actions and the fact that for the declaration games, the action space is one made of subsets).

36.5 Repeated games

Playful into, chap 10

36.6 A game on graphs: network congestion game

Network congestion game:

In a **network congestion game** on a directed graph D, n players i choose a path P_i from $s_i \in V$ to $t_i \in V$. For such choices, we denote by $x_e(P_1, ..., P_n)$ the number of players for which the edge e is on their path, so $|\{i : e \in P_i\}|$. The cost of using edge e when $x_e(P_1, ..., P_n)$ players are using it is $c_e(x_e(P_1, ..., P_n))$, where $c_e(0) = 0$ and c_e is increasing. This models congestion, as one can see by picturing the costs as traveling time: the more people want to use the same road, the slower they'll drive on it. The cost for each player i is $u_i(P_1, ..., P_n) = \sum_{e \in P_i} c_e(x_e(P_1, ..., P_n))$, which they desire to minimise.

When studying the question of the existence and the convergence to Nash equilibria, the notion of **potential** is useful. A potential is a function $p(P_1, ..., P_n)$ that will strictly decrease when *any* player switches to a better response: if there is an i so that $u_i(P_1, ..., P'_i, ..., P_n) < u_i(P_1, ..., P_i, ..., P_n)$, then $p(P_1, ..., P'_i, ..., P_n) < p(P_1, ..., P_i, ..., P_n)$. The point of a potential is that it prevents cycling in the action/parameter/strategy space: if a sequence of better replies by arbitrary players would lead back to the initial state, we'd contradict the strict deacrisingness of the potential. In particular, when the action space is finite (which is the case here, as they're only finitely many s-t-paths), this implies that any sequence of better replies must eventually arrive at a state in which no better replies exist, providing existence and the convergence to Nash equilibria.

Potentials represent a quantity that is optimised whenever any player optimises. Writing $s = (P_1, ..., P_i, ..., P_n)$ and $s' = (P_1, ..., P_i, ..., P_n)$, we have $u_i(s') - u_i(s) = \sum_{e \in P'_i \setminus P_i} c_e(x_e(s) + 1) - \sum_{e \in P_i \setminus P'_i} c_e(x_e(s)) < 0$, since the number of players using edges in $P'_i \setminus P_i$ increased by the player i and the number of players using edges in $P'_i \cap P_i$ stayed the same as i kept them. Our goal is to use this to find a quantity depending on this but not on the player i. We can for example consider the quantity $p(s) = \sum_{e \in E} c_e(x_e(s))$, which is the total cost without multiplicity. Note that $p(s') - p(s) = \sum_{e \in P'_i \setminus P_i} (c_e(x_e(s) + 1) - c_e(x_e(s))) + \sum_{e \in P_i \setminus P'_i} c_e(x_e(s) - 1) - c_e(x_e(s)))$, so it may fail to be a potential. To avoid having to swap costs on the edges, we can consider $p(s) = \sum_{e \in E} \sum_{i \leqslant x_e(s)} c_e(i)$, for which $p(s') - p(s) = u_i(s') - u_i(s)$, as costs to swap out where already in the potential of the previous states potential. This is Rosenthals potential.

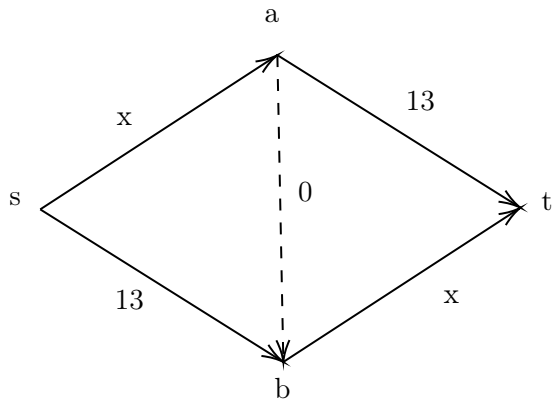
Note that when $p(s)$ is minimum, s is a Nash equilibrium, as the existence of better replies would contradict minimality, due to p decreasing under them. This provides a way of computing one (a priori not unique) Nash equilibrium for network congestion games: we minimise the potential.

We show how this can be done in our case when all destinations $t_i = t$ are the same.
To find the minimum of $p(s) = \sum_{e \in E} \sum_{i \leqslant x_e(s)} c_e(i)$, we create a flow network on D as follows. We replace each edge by n (number of players) of them, forming a directed multigraph, and subdiving each to return to a directed simple graph. Each edge has capacity 1, and we given one of the subdivided edges cost $c_e(i)$ for $i \leqslant n$, for each of the multiedges we created intermediately. We solve a minimum cost flow problem on this graph with demands 1 and -1 at the s_i and t_i respectively (cumulated if the nodes are equal; in our context, there is one sink t with demand $-n$), and 0 elsewhere. A crucial note is that the minimum flow

will be integral. Then, the cost of this flow will be $\sum_{e \in E} \sum_{i \leqslant x_e(s)} c_e(i)$, because it must use the edges of least weight for each multiplied and subdivided edge e (otherwise, we could shift the flow to the cheaper edges, on a bundle of multiplied and subdivided edges, to get a cheaper flow), so that the coast at each bundle of multiplied and subdivided edges corresponding to initial edge e is $\sum_{i \leqslant x_e(s)} c_e(i)$.

To recover the paths from this flow, we use a gaph search (ex.: DFS) from root s_n in the subgraph of the initial graph induced by the edges e for which the corresponding bundle of multiplied and subdivided edges carries flow. There must be a path to t to be found this way, otherwise the connected component of s_n has net non-zero flow, which is impossible. We then modify the flow by adjusting demands on s_n (decrease to 0) and t (decrease by 1), and set the flow along edges corresponding to those of the path to 0. We apply this recursively, until all paths are found.
If the t_i were different, it's possible that the minimum cost flow follows paths that lead to the wrong destinations, so these aren't feasible moves for the players.

We conclude this section with **Braess Paradox**. Consider the two player congestion game with linear costs, where cost coefficients are show next to the edges:



For the network without the dotted edge, the Nash equilibria is when the players choose different paths among (s, a, t) and (s, b, t), each having cost $x + 13$. If we add the dotted edge however, a new path of cost $2x$ appears as an option. One might think that adding this edge would help both players in lowering both their costs, as they don't have to pay costs on it. This is true for small x, as for $x = 2$, if both players take path (s, a, b, t), the cost for each is $2 \times 2 + 2 \times 2 = 8$, which beats all other combinations, which would have cost at least 13, since they would have to use one of the corresponding edges.

However, for $x = 5$, the case when both players take path (s, a, b, t) is still the unique Nash equilibrium: if the players used path separate paths (s, a, t) and (s, b, t) for 18 each, then each player has a better response of taking path (s, a, b, t) for $3 \times 5 = 15 < 18$; if one player uses (s, a, b, t) then the other has the choice between using it for 20, or using a different path for $2 \times 5 + 13 = 23$. Now however, the cost for each player at equilibrium has increased over that in the network without the dotted edge. This shows that in a game theoretic context, "help" can leave all players worse off then before, depending on the context.

36.7 Auctions and mechanism design

Fair division of a cake:
Consider the following situation: a cake is to be cut into n piece for n people, where each person hopes to maximise their piece. We want to avoid letting a third "impartial" party cutting the pieces as that party could be bribed by one of the people to give them a disproportionately sized piece or simply make bad cuts. We'll therefore design a mechanism that lets people do the cutting and even the do attribution of pieces, and will result (assuming none of the people collude) in equal piece for everyone.
We will design our mechanism recursively on the number of people n. For two people, we "divide" power into the legislative and executive: one player cuts the cake in two pieces, the other attributes the pieces. If the first player makes unequal halves, which we represent by f_1 and f_2, the fractions of cake, then $\min(f_1, f_2) < \max(f_1, f_2)$ and the second player can choose $\max(f_1, f_2)$ for himself. Since both players want to maximise their fraction of cake, the second one will choose the bigger piece and the first, knowing this, will maximise $\min(f_1, f_2)$ st. $f_1 + f_2 = 1$, which is attained uniquely in $f_1 = f_2 = \frac{1}{2}$.
For the general case, assume that we have a mechanism $M(n-1, f)$ for fair division of a fraction f of cake among $n-1$ players. To design $M(n, f)$, we chose a player (at random) to make n cuts to get n pieces of a fraction f of cake. Then, we choose another player (at random) that will select the piece of the player that made the cuts among the n pieces. Once the piece, say f_i of cake, is chosen, we group the rest of the pieces together and allocate them by $M(n-1, f-f_i)$.
The player that made the cuts will get the piece $\min_i(f_i)$: indeed, by induction, $M(n-1, f-f_i)$ will result in equally sized pieces of size $\frac{f-f_i}{n-1}$. This is maximised if $f_i = \min_i(f_i)$ so that the player choosing the piece will get the best result for himself in particular by choosing $f_i = \min_i(f_i)$ for the player that made the cut. Thus the player making the cut seeks to maximise $\min_i(f_i)$ st. $\sum f_i = f$. If he made unequal pieces, then there must be a piece f_j such that $f_j < \frac{f}{n}$ (otherwise by summing $f_j \geqslant \frac{f}{n}$, with at least one of the inequalities strict since not all piece are equal, we'd get $\sum f_i > f$). So $\min_i(f_i) \leqslant \frac{f}{n}$ with strict inequality if the pieces are unequal. Thus, for his own benefit, the player making the cut will make equally sized pieces.

Introducing auctions: first and second price sealed bid 1-item auctions
We consider the following auction. One item is to be auctioned to a set of n bidders, where bidder i has valuation (prohibitive price) $v_i \in \mathbb{R}$ for the item, and places bid b_i one the item. Bidders don't know the other bidders bids before placing theirs, and may bid only once.
Our perspective is that of the auction house that hope to receive the most from the item. In a first attempt at designing the auction, we could give the item to the highest bidder, which we ask to pay the bid (which they must then do). This is a "first price" auction, who's outcome is difficult to predict.
We assume that bidder i gets value 0 from not getting the item and value $v_i - b_i$ when getting the item. If we denote by $B_i = \max_{j \neq i}(b_i)$ the highest bid among the competitors of i, the i's value is $\begin{cases} 0 : b_i \leqslant B_i \\ v_i - b_i : b_i > B_i \end{cases}$.
So if the bidder overbids, in the sense that $v_i < b_i$, and gets the item, he's worse off then by not getting the item.

Even if i doesn't know B_i, we can study best responses to B_i. If $v_i \leqslant B_i$, then setting bids with $b_i > B_i$ results in value $v_i - b_i < v_i - B_i \leqslant 0$, so that bids of form $b_i \leqslant B_i$ would have been better, and each of that form is equally good. If $v_i > B_i$, then bids $b_i > B_i$ yield value $v_i - b_i > v_i - B_i > 0$, so they are better then bids $b_i \leqslant B_i$. However, they're not equally good. As a bidder, we'd hope to bid as close as our closest competitor, to get the item and simultaneously save money. In fact, any bid $b_i > B_i$ si dominated by a

smaller bid in the same interval, which has an unattained infinimmum. Hence, there isn't a undominated strategy for this first price auction.

This makes first price auctions rather unpredicatble. It is sometimes preferable for a auction house to make good predictions then to sell their objects at high prices. This is why the second price auctions, that are more predictable and that we'll now devellop, are popular inpractice.
In second price auctions, we let a winning bidder pay the price of his closest competitor, in the sens that the payoff is now $\begin{cases} 0 : b_i \leqslant B_i \\ v_i - B_i : b_i > B_i \end{cases}$. In that context, in the case of $v_i > B_i$, bids $b_i > B_i$ yield value $v_i - B_i > 0$, so they are still better then bids $b_i \leqslant B_i$. Now, they are also equally good. Hence, all responses of form $\begin{cases} b_i \leqslant B_i : v_i \leqslant B_i \\ b_i > B_i : v_i > B_i \end{cases}$ are best responses to B_i. In fact, bidding one owns valuation $b_i = v_i$ is a best response to any B_i. Even further, if we bid $b_i \neq v_i$, then there is a B_i, so a certain auction scenario, in which $v_i \leqslant B_i$ and $b_i > B_i$ (in case $b_i > v_i$) or $v_i > B_i$ and $b_i \leqslant B_i$ (in case $b_i > v_i$), such as $B_i = \frac{b_i + v_i}{2}$, in which we get negative payoff, whereas biding v_i always provides positive payoff. So betting one's valuation is the safest response. This phenomenon has a name:

DSIC Auctions:

An auction is **dominant-strategy incentive-compatible** (DSIC) if betting one's valuation is a undominated strategy.

Sponsored search auctions and Myerson's lemma:
We consider the following auction model. A TV channel is auctioning off the k time slots in the commercial break between two TV series to n advertisers. We'll model the situation with advertiser i having valuation $v_i a_j$ for slot j, where v_i is the valuation per relevance of of advertiser i, and a_j is the relevance of the time slot. For example, early slots may be less valuable as later ones, as spectators may leave for a toilet break at the beginning of the advertisement.
To allocate time slots, we'll ask the advertizers i to send bid b_i. The mechanism we'll design will be a little complex, so that bids aren't what advertizers will have to pay in the end. Our allocation rule is simple: give the best slots to the highest bidder, in the sense that we sort the bids so that wlog $b_1 \geqslant ... \geqslant b_n$, give slot j_1 attaining $\max_{j \in [k]}(a_j)$ to bidder 1, then slot j_2 attaining $\max_{j \in [k] \setminus j_1}(a_j)$ to bidder 2, and so on.

We'll ask bidder i to pay $p_i(b_1, ..., b_n)$, so that his utility will be $v_i a_{\pi(i)} - p_i(b_1, ..., b_n)$, where π maps the bidder to the slot we allocated them. We'll see that with our mechanism, we won't have p_i attaining the prohibitive value $v_i a_{\pi(i)}$. The goal of our auction is to be DSIC. The question is how to set the p_i so that the auction has this property.

We'll start by framing the problem in a more abstract context in order to devellop Myerson's lemma.
We define $m_i(b_1, ..., b_n)$ to be the relevance $a_{\pi(i)}$ of the slot attributed to i. For the case that there are more advertisers then slots, we'll assume that $a_i \geqslant 0$ and we'll add slots of value $a_i = 0$ until we can assume $k = n$. Also, we'll renumber solts, so that we can assume $a_1 \geqslant ... \geqslant a_k$, so that bidder i gets slot with relevance a_i.
To study strategic behavior, we're interested in $t \mapsto m_i(b_1..., t, ..., b_n) = m_i(t, b_{-i})$ where t is the ith argument, fixing the other bids b_{-i}. That is, we will seek to find the best response to b_{-i}. We still assume $b_1 \geqslant ... \geqslant b_n$ (without i), which leaves $m_i(t, b_{-i})$ unaffected. We can now represent $m_i(t, b_{-i})$ as a piece-

wise constant increasing function of t. Indeed, $m_i(t, b_{-i}) = \begin{cases} a_i : b_i \leqslant t < b_{i-1} \\ a_1 : t > b_1 \\ a_n : t \leqslant b_n \end{cases}$ with our renumbering of bidders and slots. Our goal is for the best response, which will maximise $v_i m_i(t, b_{-i}) - p_i(t, b_{-i})$, to be v_i. We seek p_i that provides this property.

We will reason along an analysis-sythesis argument. We will try to find properties of a desired p_i that are so restrictive, that we can construct a actual p_i solving the problem from the conditions, aka. we seek a set of necessary conditions, that together are sufficient.
Since the allocation m and price p are independent of the valuation v, yet we expect $v_i m_i(t, b_{-i}) - p_i(t, b_{-i})$ to be maximised at v_i, we have, for two potential valuations $0 \leqslant y < z$, the identities $y m_i(y, b_{-i}) - p_i(y, b_{-i}) \geqslant y m_i(z, b_{-i}) - p_i(z, b_{-i})$ (case where y is the valuation) and $z m_i(y, b_{-i}) - p_i(y, b_{-i}) \leqslant z m_i(z, b_{-i}) - p_i(z, b_{-i})$ (case where z is the valuation). This provides by grouping common terms $y[m_i(y, b_{-i}) - m_i(z, b_{-i})] \geqslant p_i(y, b_{-i}) - p_i(z, b_{-i}) \geqslant z[m_i(y, b_{-i}) - m_i(z, b_{-i})]$. In particular, we must have $m_i(y, b_{-i}) \leqslant m_i(z, b_{-i})$, or we'd contradict $y < z$, which is just the fact that m is increasing.
Now if y and z are in the same interval $[b_j, b_{j-1}[$ (or the extreme cases), the the values of m are equal and the sandwich inequality implies that $p_i(y, b_{-i}) = p_i(z, b_{-i})$. So p must be piece-wise constant, with same discontinuity points as m. To see what happens to p at the b_j ($j \neq i$), we'll take the limit $y \xrightarrow[y<b_j]{} b_j$ (so think $z = b_j$ and $y \in [b_{j+1}, b_j[$ due to our ordering) in the inequality, where we see that for the value w_j taken by p_i on interval $[b_j, b_{j-1}[$, we have $w_{j+1} - w_j = b_j(a_{j+1} - a_j)$.
So if we now find one value of the piece-wise constant p_i, we know the entire function since we now know the jumps it makes at the discontinuities.

We will now turn to the synthesis part of our reasoning. We will tak one of the candidates for p_i we just discribed, and will show that the auction is DSIC for it. Let's pick the one with $p_i(0, b_{-i}) = 0$. Since we know the jumps, we now define $p_i(t, b_{-i}) = \begin{cases} w_i : b_i \leqslant t < b_{i-1} \\ w_1 : t > b_1 \\ w_n : t \leqslant b_n \end{cases}$ where $w_j = w_1 + \sum_{k=1}^{j} b_k(a_{k+1} - a_k)$ and, in order to satisfy $w_n = 0$ (assuming bids are positive, so that in the end we really do get $p_i(0, b_{-i}) = 0$), $w_1 = -\sum_{k=1}^{n} b_k(a_{k+1} - a_k)$, with $a_{n+1} = 0$. So finally, we have form $w_j = \sum_{k=j+1}^{n} b_k(a_k - a_{k+1})$.

We will now show that for this p_i, the auction is DSIC, in the sense that $v_i m_i(t, b_{-i}) - p_i(t, b_{-i})$ is maximised for v_i.
We have $v_i m_i(t, b_{-i}) - p_i(t, b_{-i}) = \begin{cases} v_i a_j + \sum_{k=j+1}^{n} b_k(a_{k+1} - a_k) : b_j \leqslant t < b_{j-1} \\ v_i a_1 + \sum_{k=j+1}^{n} b_k(a_{k+1} - a_k) : t > b_1 \\ v_i a_n : t \leqslant b_n \end{cases}$.

In the case that $v_i \in [b_q, b_{q-1}[$, then for $j > q$, we have $v_i a_j + \sum_{k=j+1}^{n} b_k(a_{k+1} - a_k) \leqslant v_i \left(a_j + \sum_{k=j+1}^{n} (a_{k+1} - a_k) \right)$,

FIX: the indices, translating Roughgarden's "visual proof", fixing the whole proof actually.

Combinatorial auctions (single minded case):
In a combinatorial auction, n bidders seek to obtain bundles among m items. Their valuation of a bundle $S \subseteq [m]$ of items is $v_i(S)$, where i denotes the bidder and $v_i : 2^{[m]} \to \mathbb{R}_+$ is a function that may not be linear. For example, if the auction consists of items "coffee" and "cup", then one can imagine a valuation of 1 for the set of both items, and of 0 for just one or none of the items, as getting the coffee without the cup or the cup without the coffee is pointless. The goal of the auction is to find a collection of at most n

pairwise disjoint subsets of items to be given to the corresponding bidders.

Combinatorial auctions are hard to perform. We'll focus on the case of **single minded** valuations that satisfy $v_i(S) = \begin{cases} v_i^* : S_i^* \subseteq S \\ 0 : else \end{cases}$, for a desired set S_i^* and a valuation $v_i^* > 0$. For this case, we'll study the auction consisting of bidders reporting (b_i, S_i), where b_i is the bid and S_i is the bundle of items bidded for, and the auctioneer allocating either bundles S_i or $\emptyset$ to bidders i, asking for price b_i if the set S_i is allocated. We get a first hardness result in the following:

NP-completeness:

For the context described, finding the revenue maximising allocation is NP-complete.

Proof: We'll make a reduction to the independent set problem, in which we ask if a graph has an independent set (a set of vertices with no edges between them) of size at least k. We consider $|V|$ bidders, bidder i representing vertex i of the graph, and edges E as items. We let $v_i^* = 1$ and $S_i^* = \delta(i)$ and consider the instance in which bidders bid truhfully, in the sense $b = v^*$. When we run our black-box allocation, we have the S_i so as to maximise revenue $\sum_{i \in A} b_i = |A|$ (truthful bids) where A is the set of bidders having been allocated the set they bidded for, then the $S_i^* \subseteq S_i$ for $i \in A$ must be pairwise disjoint, so that the vertices represented by A form an independent set, as they have no common edges in their neighbourhoods. Hence, if we can find an allocation of revenue at least k, then this corresponds to an independent set of size at least k. The problem is also NP, as we can check disjointness and compute the revenue in polynomial time.

In the single minded context, we can investigate the following greedy allocation. We rank the bidders in terms of their "bid per item", in the sense that $\frac{b_1}{\sqrt{|S_1|}} \geqslant ... \geqslant \frac{b_n}{\sqrt{|S_n|}}$ after possible renumbering of bidders. Ignore the square roots for now: their only purpose will be to make nice derivations of bounds on the quality of the allocation. We then allocate S_1 to bidder 1, and then iteratively, at iteration i with set W_{i-1} of bidders having recieved the set they asked for so far (so $W_0 = \{1\}$), we check if $S_i \cap (\cup_{j \in W_i} S_j) = \emptyset$, in which case we can and do give i his desired bundle S_i. We the set $W_i = W_{i-1} \cup i$. Otherwise, we pass to the next bidder, letting $W_i = W_{i-1}$.

In the single minded context, we can investigate the following greedy allocation. We rank the bidders in terms of their "bid per item", in the sense that $\frac{b_1}{\sqrt{|S_1|}} \geqslant ... \geqslant \frac{b_n}{\sqrt{|S_n|}}$ after possible renumbering of bidders. Ignore the square roots for now: their only purpose will be to make nice derivations of bounds on the quality of the allocation. We then allocate S_1 to bidder 1, and then iteratively, at iteration i with set W_{i-1} of bidders having recieved the set they asked for so far (so $W_0 = \{1\}$), we check if $S_i \cap (\cup_{j \in W_i} S_j) = \emptyset$, in which case we can and do give i his desired bundle S_i. We the set $W_i = W_{i-1} \cup i$. Otherwise, we pass to the next bidder, letting $W_i = W_{i-1}$.

Inspired by the second-price auction concept, we'll ask a winner $i \in W_n$ to pay price $p_i = \frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i|}$, where $j > i$ is the smallest index such that $S_k \cap S_j = \emptyset$ for all $k < j, k \neq i$, but $S_i \cap S_j \neq \emptyset$, or $p_i = 0$ if such an index doesn't exist. For the $i \notin W_n$ that where allocated $\emptyset$, we ask to pay $p_i = 0$.

First, we check that taking part in the auction can only yield positive utility under our rules and certain assumptions. Assuming the reasonable (and necessary for the algorithm to work) $S_i^* \neq \emptyset$, so that $v_i(\emptyset) = 0$, we see that for $i \notin W_n$, bidder i has utility $v_i(\emptyset) - p_i = 0 - 0 = 0 \geqslant 0$. If $i \in W_n$, then if $b_i \leqslant v_i^*$ and

$S_i^* \subseteq S_i$, then the utility is $v_i(S_i) - \frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i|} \geqslant v_i^* - b_i \geqslant 0$, since $\frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i|} \leqslant b_i$ as $j > i$. Bidding for a set that doesn't contain S_i^* will yield utility $0 - \frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i|} \leqslant 0$, assuming positive bids, and bidding higher then ones valuation v_i^* may also result in loss.

We now investigate best responses in this context. We'll first show that given the other bidders bids, bidder i is better off (in the large sense) bidding (v_i^*, S_i^*) then (b_i, S_i), for any S_i and b_i. We disjoin cases.

If S_i doesn't contain S_i^*, the utility of bidding (b_i, S_i) is either 0 if $i \notin W_n$, or $0 - \frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i|} \leqslant 0$ if $i \in W_n$. By bidding (v_i^*, S_i^*) instead, two outcomes can occur. If $i \notin W_n$, utility is still 0 and in particular better then the outcomes of bidding (b_i, S_i). If $i \in W_n$, then the utility is $v_i(S_i^*) - \frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i^*|} \geqslant v_i^* - v_i^* = 0$ since $i \in W_n$, so that $\frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i^*|} \leqslant v_i^*$ must hold. So in this case (v_i^*, S_i^*) is better. In the cases that the index j is inexistent and the price is set to 0, the inequalities we derived still hold.

Next, if $S_i^* \subseteq S_i$, things get interesting. The utility of bidding (b_i, S_i) is either 0 if $i \notin W_n$, or $v_i^* - \frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i|}$ if $i \in W_n$. We saw that bidding (v_i^*, S_i^*) yields positive utility in all cases, so that the utility of bidding (b_i, S_i) for $i \notin W_n$ is dominated for this case.

In the case that $i \in W_n$ when bidding (b_i, S_i), then by bidding (b_i, S_i^*), we'd also end up winning in the sense that $i \in W_n$. First, i is possibly considered early on in the algorithm, as $\frac{b_i}{\sqrt{|S_j^*|}} \geqslant \frac{b_i}{\sqrt{|S_j|}}$. Now, since if at iteration i, we had $S_i \cap (\cup_{j \in W_i} S_j) = \emptyset$, then also $S_i^* \cap (\cup_{j \in W_i} S_j) = \emptyset$ as $S_i^* \subseteq S_i$, and for the pervious $W_{k<i}$ too, so that i is selected in this case too.

We'll now show that bidding (b_i, S_i^*) would have been better. We can directly deduce this from $v_i^* - \frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i|} \leqslant v_i^* - \frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i^*|}$ (by positivity and $S_i^* \subseteq S_i$), as it's unclear if bidding (b_i, S_i^*) yields the same index j selected by our algorithm. This happens to be true, and we'll prove it. Assume for contradiction that there is an index r with $i < r < j$ such that $S_k \cap S_r = \emptyset$ for all $k < r, k \neq i$, but $S_i^* \cap S_r \neq \emptyset$. Then in particular $i < r$ and $S_k \cap S_r = \emptyset$ for all $k < r, k \neq i$, but $S_i \cap S_r \neq \emptyset$, since $(S_i^* \cap S_r) \subseteq (S_i \cap S_r)$, as $S_i^* \subseteq S_i$. So r would contradict the minimality of j since $r < j$ and it has the properties defining j.

Therefore, if r is the index selected for iteration i when we bet (b_i, S_i^*), we must have $r \geqslant j$, or r non-existent, in which case the price is 0 and the utility is better. In case $r \geqslant j$, we have $\frac{b_i}{\sqrt{|S_i^*|}} \geqslant \frac{b_i}{\sqrt{|S_i|}} \geqslant \frac{b_j}{\sqrt{|S_j|}} \geqslant \frac{b_r}{\sqrt{|S_r|}}$, so that $v_i^* - \frac{b_r}{\sqrt{|S_r|}}\sqrt{|S_i^*|} \geqslant v_i^* - \frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i|}$, which means that bidding (b_i, S_i^*) is better. Finally, we'll show that bidding (v_i^*, S_i^*) is better then bidding (b_i, S_i^*). As we metioned early on, we only have to consider the case here we bid (b_i, S_i^*) and win, so that $i \in W_n$. If $b_i \leqslant v_i^*$, then $\frac{v_i^*}{\sqrt{|S_i^*|}} \geqslant \frac{b_i}{\sqrt{|S_i^*|}}$ and i is considered earlier on in the algorithm. Since the bundle S_i^* stays the same, i will be added to the winner set in both cases at its iteration, as its bundle shares no items with any predecessing bundle. So the the utility of (v_i^*, S_i^*) is $v_i^* - \frac{b_r}{\sqrt{|S_r|}}\sqrt{|S_i^*|}$, where r is the selected index. One can show by similar

arguments to our previous ones that $r = j$ where j was the index for (b_i, S_i^*), this time even more directly, as the bundle stays the same. Hence, in the case that $b_i \leqslant v_i^*$, bidding (v_i^*, S_i^*) instead of (b_i, S_i^*) yields same utility, hece the former is better in the large sense.

And you'll be as pleased to hear as I was, that after this long case disjunction, the final case $b_i > v_i^*$ may not hold. In this case, the "bid per item" ratio decreases, and it's now possible that i isn't selected anymore, as a bidder that is now a predecessor of i may have had, for example, the same desired bundle as i, and in general a bundle with an item common to that of i, preventing i from being taken afterwards. The resulting utility is 0. In parallel, it's still possible that $v_i^* - \frac{b_j}{\sqrt{|S_j|}}\sqrt{|S_i^*|} > 0$, depending on b_j. In such a case, bidding (b_i, S_i^*) would have been better then (v_i^*, S_i^*).
FIX: if it can be fixed... might want to read the actual paper cause the proof of the AGT textbook page 274 is trash.

We have seen that in this type of auction, truthful bidding is an undominated strategy.
What can we say about the welfare of the auction ?

Quality ratio of the greedy allocation:

For an optimal allocation maxiximising welfare $OPT = \sum_{i \in A} v_i^*$, where A is the set of bidders recieving aset containing their desired set, we have $OPT \leqslant \sqrt{m} \sum_{i \in W_n} v_i^*$.

Proof: We consider for $i \in W_n$ the sets $A_i = \{j \in A : j \geqslant i, S_j \cap S_i \neq \emptyset\}$, consisting of the elemts of A greater then i that won't be taken by the greedy algorithm, as they appear after i and their bundles share items. We have $A \subseteq (\cup_{i \in W_n} A_i)$, since for the $i \in A \cap W_n$, we have $i \in A_i$ and for the $j \in A \backslash W_n$, the algorithm must have skipt them due to sharing an item with the bundle of some $i < j$ of W_n, so that $j \in A_i$.

We'll soon show that $\sum_{j \in A_i} v_j^* \leqslant \sqrt{m} v_i^*$ for $i \in W_n$. Then $\sum_{i \in A} v_i^* \leqslant \sum_{j \in \cup_{i \in W_n} A_i} \leqslant \sum_{i \in W_n} \sum_{j \in A_i} v_j^*$ by inclusion and multiple accounting. So this will provide $\sum_{i \in A} v_i^* \leqslant \sqrt{m} \sum_{i \in W_n} v_i^*$, the desired result.

By the ordering, we have $\frac{v_j^*}{\sqrt{|S_j^*|}} \leqslant \frac{v_i^*}{\sqrt{|S_i^*|}}$ for all $j \in A_i$, so that $\sum_{j \in A_i} v_j^* \leqslant \frac{v_i^*}{\sqrt{|S_i^*|}} \sum_{j \in A_i} \sqrt{|S_j^*|}$. Now the square roots will come into play. We can bound this further with Cauchy-Schwarz on $\sum_{j \in A_i} \sqrt{|S_j^*|}$ see as a dot-product of $(1, ..., 1)$ with $\left(\sqrt{|S_j^*|}, ...\right)$ to get $\sum_{j \in A_i} \sqrt{|S_j^*|} \leqslant \left(\sqrt{\sum_{j \in A_i} |S_j^*|}\right) \sqrt{|A_i|}$. Since the S_j^* for $j \in A_i \subseteq A$ form disjoint subsets of $[m]$, we have bound $\sqrt{\sum_{j \in A_i} |S_j^*|} \leqslant \sqrt{m}$. Next, note that for each $j \in A_i$, we have $S_j \cap S_i \neq \emptyset$ so we can account for one element of S_i per j. Since the S_j for $j \in A_i \subseteq A$ form disjoint subsets, we'll never account the same element of S_i this way. So $|A_i| \leqslant |S_i|$. We can fnially put the bounds together to get $\sum_{j \in A_i} v_j^* \leqslant \frac{v_i^*}{\sqrt{|S_i^*|}} \sum_{j \in A_i} \sqrt{|S_j^*|} \leqslant \frac{v_i^*}{\sqrt{|S_i^*|}} \sqrt{m} \sqrt{|S_i^*|} = \sqrt{m} v_i^*$.

36.8 Markets

Microeconomics Reny, chap 4 and 5

36.9 Solutions

Ex.CRTC:
For a tree, one cop is enough, as the cop can move towards the component that the robber is in, when deleting the cop-vertex temporarily. The robber can't leave this component without passing through the cop-vertex, thereby getting caught. The size of the component the robber is in will decrease by at least 1 at each turn of the cop, so that after finitely many turns, the cop catches the robber.
On a cycle of size $\geqslant 4$, the robber can start on a vertex not in the neighbourhood of the cop and move in the same direction as the cop at each turn to maintain the distance between them, and evade the cop indefinitely.