Discrete Optimization - ADM2

Lecturer
Prof. Dr. Tom McCormick

Teaching Assistants
Svenja Griesbach and Sarah Morell

Student Assistant
Christos Pavlidis

Notes
Simon Cyrani

Version
git: afc835a
compiled: Saturday 21$^{\text{st}}$ May, 2022 11:43

Abstract

The following lecture notes are my personal (and therefore unofficial) write-up for 'Discrete Optimization' aka 'ADM II', which took place in summer semester 2022 at Technische Universität Berlin. I do not guarantee correctness, completeness, or anything else. Importantly, note that I willfully changed some specific notations, reordered some material, and left out parts that I didn't found worth typing down.

If you miss something, feel free to contribute in the repository!

Contents

Summary of lectures

Part I

Lecture notes

Lecture 1
Di 19 Apr 2022

1 Introduction

In ADM1 we often already worked with Integer Programming and just assumed everything is fine. In this course however, we want to find out how Integer Programming actually works, why it is generally "hard", and under which circumstances it is "easy".

Definition 1.1 (Flavors of IP)**.** First of all, we want to define different variants of **Integer Programming**:

- **Pure Integer Programming** assumes *all* variables are integer.
- **Mixed Integer Programming** also allows some variables to be real.
- **Binary Integer Programming**, also called 0-1-Integer-Programming, restricts the integer variables to $\mathbb{B} := \{0, 1\}$. Mixed variants are also possible.

Question 1.2. Why is IP harder than LP? Naively, one would assume this should *not* be the case, because our search space is smaller (at most countably infinite)!

Let's solve the IP in ADM1-style - suppose

$$\begin{aligned} \max_x \quad & w^T x \\ \text{s.t.} \quad & x \in Q = \{x \in \mathbb{Z}^n : Ax \leqslant b\} \end{aligned}$$

For simplicity, assume Q is bounded. Then the set of feasible points in Q is finite, and therefore we can consider the polytope

$$\operatorname{conv}(Q) = \{x \in \mathbb{R}^n \mid A'x \leqslant b'\}$$

for suitable A' and b'. Notice all vertices must be in Q and thus are integral. As a consequence, it is sufficient to solve the LP

$$\begin{aligned} \max_x \quad & w^T x \\ \text{s.t.} \quad & A'x \leqslant b'. \end{aligned}$$

Warning. Computing A', b' is non-trivial! In fact, computing the **integer hull** $\operatorname{conv}(Q)$ is what makes IP hard.

1.1 IP is "hard"

We can gather more evidence that IP must be hard.

Theorem 1.3. Every logical statement can be expressed with integer programming.

Proof. It suffices to show that for variables $x_1, x_2, y \in \mathbb{B}$ we can find IPs that model $\wedge, \vee, \neg$.

- $y = x_1 \wedge x_2$ can be modeled as

$$\begin{aligned} y &\leqslant x_1 \\ y &\leqslant x_2 \\ y &\geqslant x_1 + x_2 - 1 \end{aligned}$$

- $y := x_1 \vee x_2$ can be modeled as

$$\begin{aligned} y &\geqslant x_1 \\ y &\geqslant x_2 \\ y &\leqslant x_1 + x_2 \end{aligned}$$

- $y := \neg x$ can be modeled as

$$y = 1 - x$$

□

Theorem 1.4. IP can model (finite) unions of polyhedra, e.g. non-convex problems.

Proof. Consider

$$P := \bigcup_{i=1}^{k} \underbrace{\{x \mid A_i x \leqslant b_i\}}_{P_i}$$

and introduce auxiliary binary variables

$$y_i := \begin{cases} 1, & \text{if } x \in P_i, \\ 0, & \text{if we don't care.} \end{cases}$$

Now, assume $M \in \mathbb{R}$ large enough (**Big-M method**), such that

$$P_i \subseteq \overline{P} := \{x \mid A_i x \leqslant b_i + M \cdot \mathbf{1}\}$$

for all i. Thus, we can construct following IP:

$$\begin{aligned}\min_x \quad & w^T x \\ \text{s.t.} \quad & A_i x \leqslant b_i + (1-y_i)M \cdot \mathbf{1}, \quad i = 1, \dots, k, \\ & \sum_i y_i = 1\end{aligned}$$

This forces exactly one y_i to 1, resulting that $x \in P_i$ and $x \in \overline{P}$ is sufficient. We call this method **righthandside Big-M**, short RHS. Further information can be found in [Wol99, Ch. 1]. ☐

Note. It's also possible to handle M as a symbolic value, but this makes other things more complicated.

Problem. Finding M big enough can make LP hard to solve, because of matrix-inversions getting numerically unstable.

Alternative proof of Theorem 1.4. Assume that we can bound each $x \in P_i$ by u_i, i.e. $x \leqslant u_i$ (note that this is basically a hidden big-M!). Now we can disaggregate x for each P_i as its own private $x_i \in \mathbb{R}^k$, and analogously introduce $y_i \in \mathbb{B}$ to restrict ourselves to one polyhedron:

$$\begin{aligned}\min_x \quad & w^T x \\ \text{s.t.} \quad & A_i x_i \leqslant y_i b_i, \quad i = 1, \dots, k, \\ & x_i \leqslant y_i u_i, \quad i = 1, \dots, k, \\ & \sum_{i=1}^n y_i = 1, \\ & \sum_{i=1}^n x_i = x, \\ & x, x_i \geqslant 0\end{aligned}$$

Again, exactly one y_i is equal to 1, forcing the other x_j to be equal to 0, and thus setting x to x_i. We call this method **upper bound on x Big-M**, short UBX. ☐

Remark 1.5. In general, it cannot be said if RHS or UBX is better. Even though RHS only introduces n new variables as opposed to UBX's nk variables, UBX's disaggregated formulation often is *tighter* in the sense that the LP relaxation is closer to the integer hull.

Lecture 2
Do 21 Apr 2022

Theorem 1.6. IP can approximate any objective function infinitely good.

Proof. First, consider a **piecewise linear** objective function f with (not necessarily equidistant) breakpoints $a_1, \ldots, a_k$.

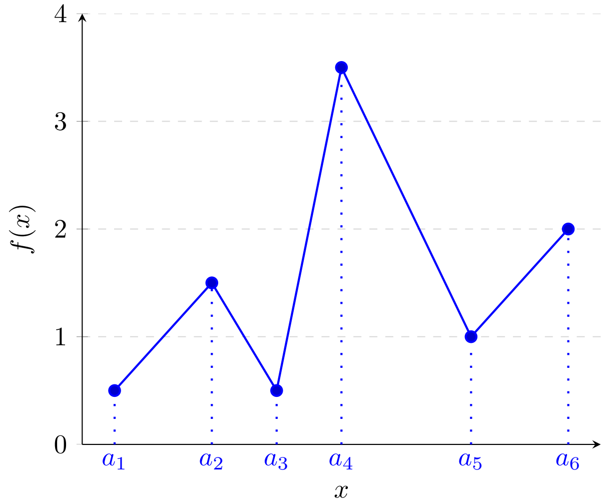


Figure 1: A piecewise linear function

Defining the intervals $I_i := [a_i, a_{i+1}]$ for each segment of f, we can introduce binary variables y_i such that

$$y_i = \begin{cases} 1, & \text{if } x \in I_i, \\ 0, & \text{otherwise} \end{cases}$$

Note that for $x \in I_i$, x is a convex combination of a_i, a_{i+1}. Therefore, we can express x as a linear combination of all breakpoints a_i, with the additional constraint that all except 2 scalars must be 0. By linearity, this also holds for $f(x)$. Translating into IP:

$$\begin{aligned} \min_{\lambda} \quad & \sum_i \lambda_i f(a_i) \\ \text{s.t.} \quad & \lambda_1 \leqslant y_1, \\ & \lambda_k \leqslant y_{k-1}, \\ & \lambda_i \leqslant y_{i-1} + y_i, \quad i = 2, \ldots, k-1, \\ & \sum_i y_i = 1 \end{aligned}$$

One can show this already suffices to model any cost function: For suitable choices of breakpoints we can approximate any function by piecewise linear functions. Details can be found in [Orl93, Ch. 14] or [Wol99, Ch. 1]. ☐

Conclusion 1.7. Summarizing, following facts that hold for IP, but not LP, deliver an intuition why IP should be hard:

1. Consider a polyhedron $P = \{x \in \mathbb{R}^n \mid Ax = b\}$ and its integer hull Q. Even

though P can be "smooth" (e.g. cube), its integer hull can look more like a "disco ball" with many facets.

2. In real life, many problems can be modeled with decision variables. IP can handle this, see Theorem 1.3.
3. Additionally, IP also can handle non-convex problems, see Theorem 1.4 and Theorem 1.6.

1.2 Representation of IPs

For a given IP, a set Q of feasible points can be formulated by many polyhedra P.

Example 1.8. Consider

$$Q := \{0000, 1000, 0100, 0010, 0110, 0101, 0011\} \subseteq \mathbb{Z}^4.$$

Then we can give following representations P_i such that P_i is an integer hull of Q:

$$\begin{aligned} P_1 &= \{x \in \mathbb{R}^4 \mid 93x_1 + 49x_2 + 37x_3 + 29x_4 \leqslant 111\} \\ P_2 &= \{x \in \mathbb{R}^4 \mid \quad 2x_1 + x_2 + x_3 + x_4 \leqslant 2\} \\ P_3 &= \{x \in \mathbb{R}^4 \mid \quad 2x_1 + x_2 + x_3 + x_4 \leqslant 2, \\ &\qquad x_1 + x_2 \leqslant 1, \\ &\qquad x_1 + x_3 \leqslant 1, \\ &\qquad x_1 + x_4 \leqslant 1\} \end{aligned}$$

One can show that $P_3 \subsetneq P_2 \subsetneq P_1$.

Example 1.9. A real-life example is the problem of placing facilities. Given n stores and m warehouses, decide which warehouse should be build at all, and which should deliver which store (for some cost function). Let $y_i \in \mathbb{B}$ denote if warehouse i should be opened, and x_{ij} if warehouse i should serve store j. Then

$$\begin{aligned} P_1 &:= \{x \mid \forall i : \sum_j x_{ij} \leqslant m y_i\}, \\ P_2 &:= \{x \mid \ \forall i, j : x_{ij} \leqslant y_i\} \end{aligned}$$

both represent the condition to only serve stores from warehouses that are opened. Notice that $P_2 \subsetneq P_1$ is tighter, but has $n \cdot m$ instead of n constraints.

1.3 Complexity

In order to define hardness, it is useful to define complexity first. We can use **big-$\mathcal{O}$** for this. During the rest of the lecture, we had a recap on this. For details, refer to canonical sources.

Lecture 3
Di 26 Apr 2022

2 Hardness

Prior, we only gave intuition why IP is "harder" than LP. In order to analyze IP more thoroughly, we now want to work towards a formal definition of hardness of problems. Basically, there are two types of problems for now:

Definition 2.1 (Problem types)**.** We differentiate between

- **decision problems**, which can be answered by *Yes* or *No* only, and
- **optimization problems**, which seeks for a numerical value minimizing a certain (cost) function.

For the beginning, we can cheat and restrict ourselves to decision problems.

Example 2.2. Possible decision problems could be:

1. Does there exist a Hamiltonian cycle?
2. Is the LP feasible?

Question 2.3. How do we model an optimization problem as an decision problem?

Answer. We can simply introduce a parameter z which we use as a bound for the value we want to optimize.

Example 2.4. Possible reformulations of optimization problems are therefore:

1. Does there exist a feasible x with $c^T x \leqslant z$?
2. Does there exist a spanning tree with cost less than z?
3. Is there a clique with size less than z?

Definition 2.5. A **clique** C is a subset of nodes V of a graph $G = (V, E)$ s.t. for all $i, j \in C$ it must hold true that $(i, j) \in E$.

Theorem 2.6. When we model an optimization problem as a decision problem, then there exist a oracle-polynomial way to solve the optimization problem using the decision problem as an oracle.

Algorithm 1: Oracle-polynomial algorithm for max-clique

Use binary search to find z^* in $\mathcal{O}(\log n)$
$G' \leftarrow G = (V, E)$
for $i = 1, \ldots, n$ **do**
  $G'' \leftarrow G'$, but remove all edges incident to node i
  if *Call of decision oracle on* G'', z^* *is true* **then**
    $G' \leftarrow G''$
  end
end

Theorem 2.7. Final $\overline{G} := G'$ is a max-clique.

Proof. The size of a max clique in G' never goes below z^*. Therefore, there exists a clique $C \subseteq \overline{G}$ with $|C| = z^*$. Suppose $\overline{G}$ has more than z^* nodes. Then $i \in \overline{G} \backslash C$. But then the algorithm would have deleted this node! □

Corollary 2.8. If we have an optimal oracle, then one can solve decision version in oracle-polynomial time using the optimal oracle.

Conclusion 2.9. Optimization and decision version differ only by a polynomial factor of complexity. Therefore, either both are easy or both are hard.

Definition 2.10 (Certificate)**.** Given an instance of any problem with size n, a **certificate** is a binary-encoded string that is generated by some algorithm specific to the problem, taking the instance as input. We say the certificate is a **succinct certificate**, if its *length* is polynomial in the input size n.

Definition 2.11 (**NP**)**.** We say a (decision) problem P lies in **NP**, if for all Yes-instances I there exists a succinct certificate C and a certificate checking algorithm A that confirms $A(I, C)$ in polynomial time.

Theorem 2.12. Max-clique lies in **NP**

Proof. We use our clique C directly as the certificate.

Algorithm 2: Certificate checking for max-clique

```
if |C| < z then
    return NO
end
else
    for i, j ∈ C do
        if (i, j) ∉ E then
            return NO
        end
    end
end
return YES
```

□

Remark 2.13. Note that we don't care for No-instances! In order to verify them we would need to list all $\binom{n}{z}$ subsets (for max-clique), which is *not* polynomial.

Theorem 2.14. $\mathbf{P} \subseteq \mathbf{NP}$

Proof. Let $P \in \mathbf{P}$. Then there exists a polynomial algorithm A. Record the steps of A on an instance I and use this as a polynomial certificate. □

Theorem 2.15. $\mathsf{LP} \in \mathbf{NP}$, using decision variant if there is any feasible x.

Proof. If feasible, there exists a basic feasible solution x^*. We verify by checking $Bx^* = b$. One can show that x^* has polynomial bits. □

Let's also have a look at the canonical **NP** problem:

Definition 2.16 (Satisfiability problem, SAT)**.** Consider n logical variables $v_1, \ldots, v_n$, allowing also the negated literals $\overline{v_i}$. Additionally, we have m clauses $C_1, \ldots, C_m$, which are subsets of the literals. Determining if there is an assignment such that the overall clause is true (i.e. each subclause has at least one true literal) is known as the **satisfiability problem**, for short SAT.

Example 2.17. A few examples:

1. $(v_1 \vee v_2 \vee v_3) \wedge (\overline{v_1} \vee \overline{v_2} \vee \overline{v_3})$
   This instance is true for $v = (110)$.
2. $(v_1 \vee v_2) \wedge (\overline{v_1} \vee v_2) \wedge (\overline{v_2} \vee v_3) \wedge (\overline{v_3} \vee \overline{v_4})$
   One can check that this instance is always false.

Theorem 2.18. SAT $\in$ **NP**

Proof. The satisfiability truth assignment is a succinct certificate. □

Theorem 2.19 (Cook)**.** If $P \in$ **NP**, then P has an oracle-polynomial algorithm with SAT as an oracle.

Proof. Suppose $P \in$ **NP**, then P has a non-deterministic Turing Machine with polynomial size. Encode the Turing Machine as a logical formula such that it is true iff P is a Yes-instance. □

Lecture 4
Do 28 Apr 2022

We remind ourselves that IP can formulate logic, and therefore can encode SAT formulas.

Example 2.20. Translating from Example 2.17:

1. $$\begin{aligned} x_1 + x_2 + x_3 &\geqslant 1 \\ (1-x_1) + (1-x_2) + (1-x_3) &\geqslant 1 \\ x &\in \mathbb{B}^3 \end{aligned}$$

2. $$\begin{aligned} x_1 + x_2 &\geqslant 1 \\ (1-x_1) + x_2 &\geqslant 1 \\ (1-x_2) + x_3 &\geqslant 1 \\ (1-x_2) + (1-x_3) &\geqslant 1 \\ x &\in \mathbb{B}^3 \end{aligned}$$

Definition 2.21 (Reduction)**.** We say $P \propto Q$ ("P **reduces to** Q") if there exists a polynomial algorithm A such that

1. for all instances $I \in P$, $A(I)$ is element of Q,
2. I is **Yes-preserving**, e.g. I is Yes-instance of P iff $A(I)$ is Yes-instance of Q.

Definition 2.22 (**NP**-complete)**.** A problem P is **NP-complete**, if

1. $P \in$ **NP**, and
2. for all $Q \in$ **NP** it holds that $Q \propto P$.

We call the set of all **NP**-complete problems **NPC**.

Corollary 2.23. Using our new definition, it follows immediately from Theorem 2.19 that SAT $\in$ **NPC**.

Proof Strategy. In order to show a problem P is **NP**-complete we first describe a way to construct a succinct certificate, and state an algorithm that describes how we use the certificate to verify a Yes-instance is indeed a Yes-instance.

After that, we find a suitable problem Q, which is known to be **NP**-complete, and try to proof $Q \propto P$. We do this by converting each instance of Q into an instance of P in polynomial time, and verify that the conversion is Yes-preserving.

Theorem 2.24. SAT is as hard as 0-1-IP

Proof. We know 0-1-IP $\in$ **NP**, and therefore 0-1-IP $\propto$ SAT. It remains to show SAT $\propto$ 0-1-IP: Let $I \in$ SAT with clauses $c_j = l_1, ..., l_k$. We convert each clause to the inequality $l_1 + ... + l_k \geqslant 1$ for binary l. As shown in Theorem 1.3, this encodes exactly the logic formula. □

Definition 2.25 (3SAT)**.** We define 3SAT as a variant of SAT where we only allow clauses with exactly 3 literals, e.g. $|C_j| = 3$.

Theorem 2.26. 3SAT $\in$ **NPC**

Proof. 3SAT $\in$ **NP** follows directly from SAT $\in$ **NP**. It remains to show SAT $\propto$ 3SAT. Consider clause $C_j = (l_1 \vee ... \vee l_k)$ for $k > 3$. Add $k-3$ new variables $y_{2,j}, ..., y_{k-2,j}$ and replace C_j with

$$(l_1 \vee l_2 \vee y_{2,j}) \wedge (\overline{y}_{2,j} \vee l_3 \vee y_{3,j}) \wedge ... \wedge (\overline{y}_{k-2,j} \vee l_{k-1} \vee l_k)$$

One can figure out via proof tables and induction that this is indeed Yes-preserving. □

Definition 2.27 (Node cover, NC)**.** Given graph $G = (N, E)$, we say $C \subseteq N$ is a **node cover** if for every edge in E at least one of the nodes is in N. We define NC as the decision problem if there is a node cover of at most size z.

Theorem 2.28. NC $\in$ **NPC**

Proof. We can easily check if for a given C, it is indeed a node cover in polynomial time. Therefore NC $\in$ **NPC**. We want to reduce from 3SAT:

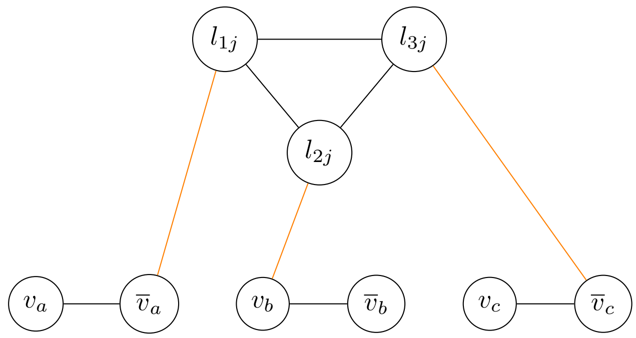


Figure 2: Schema of how to use triangle "gadgets" for a single clause C_j

Consider an instance of 3SAT and construct a graph as shown in *Figure* 2, e.g. for each variable v_i construct an edge between nodes v_i and $\overline{v}_i$, and for each clause C_j construct a triangle l_{1j}, l_{2j}, l_{3j}. Now, connect each node of the triangle with the corresponding literal in the clause (the orange edges). Using this construction, we want to proove that there is a node cover of size $n + 2m$ iff the 3SAT instance is valid.

Suppose the 3SAT instance is feasible. We use the n nodes of the feasible labeling corresponding to the literals. Now, because the labelling is valid, at least one orange edge per triangle must be covered, by construction. Therefore, we can choose 2 additional nodes per triangle that cover the triangle and the remaining orange edges.

On the other side, suppose there is a node cover of size (at most) $n+2m$. Analoguous, each triangle must have at least 2 chosen nodes to cover each edge, and each literal-pair at least 1 node, meaning our bounds must actually be exact to not overshoot $n + 2m$. Therefore, the node cover represents a valid truth assignment, which is also a valid labelling, because each clause has a remaining orange edge, which is covered by one of the literals.

Therefore, our reduction is Yes-preserving. □

Remark 2.29. NC in bipartite graphs is in **P**.

Definition 2.30 (Independent set, IS)**.** For a graph $G = (N, E)$ we call $S \subseteq N$ a **independent set** (or **stable set**) if no edge has both nodes in S. The decision problem, called IS, if there is a independent set of size at least z.

Theorem 2.31. IS $\in$ **NPC**

Proof. IS $\in$ **NP** trivial. We can also easily show that C is a node cover iff $N \backslash C$ is stable. □

Theorem 2.32. CLIQUE $\in$ **NPC**

Proof. CLIQUE $\in$ **NP** trivial. We can also easily show that C is a clique in $G = (N, E)$ iff C is stable in $(N, \overline{E})$. □

Definition 2.33 (Partition, PART)**.** Given $a_1, ..., a_n \in \mathbb{Z}^+$. The decision problem if there is a set $S \subseteq \{1, ..., n\}$ such that

$$\sum_{i \in S} a_i = \sum_{i \notin S} a_i$$

is called **partition problem**, PART.

Theorem 2.34. PART $\in$ **NPC**

Proof Sketch. We can show [Vyg18, Ch. 15.5]:

$$\mathsf{SAT} \propto \text{3-dim match} \propto \text{subset sum} \propto \mathsf{PART}$$

Remark 2.35. Still, PART has a pseudopolynomial algorithm using dynamic programming.

Definition 2.36. If a (numerical) problem is only **NP**-complete if it depends on the size of the numbers (e.g. polynomial in the unary bit model), we call it **weakly NP-complete**. Otherwise, we call it **strongly NP-complete**.

Definition 2.37 (3-partition, 3PART)**.** Given the numbers $a_1, ..., a_{3k} \in \mathbb{Z}$. The problem, if we can partition these numbers in sets of 3 such that every set has the same value, is called **3-Partition**, or 3PART.

Theorem 2.38. 3PART is strongly **NP**-complete.

Remark 2.39. Only weakly **NP**-complete problems could have pseudopolynomial algorithms (except $\mathbf{P} = \mathbf{NP}$).

Lecture 5
Di 03 May 2022

Definition 2.40 (**NP**-hard)**.** Consider an optimization problem P. Formally, we can't have $P \in \mathbf{NP}$, but because of Theorem 2.6 we can introduce the notion to call P **NP-hard**, if its decision variant is **NP**-complete.

Definition 2.41 (co−**NP**)**.** We say $P \in \mathsf{co}-\mathbf{NP}$, if we have a succinct certificate for verifying No-instances.

Example 2.42. Given a matrix A. We call it totally unimodular, if every square submatrix has determinant 0 or 1. Deciding if A is totally unimodular is in co−**NP**, because giving a failing submatrix as a succinct certificate is easy.

Theorem 2.43. The decision version of LP is in co−**NP**.

Proof. The answer to the decision problem is No iff

1. the system is infeasible, or
2. the system is feasible, but the optimal cost is larger than the z we want.

Because both can be decided with the tools we have in polynomial time, LP is indeed in co−**NP**. □

Definition 2.44 (co−**NP**-complete)**.** Analoguous to Definition 2.22, we can also define **co−NP-completeness**, or co−**NPC**, for the "most difficult" problem in co−**NP**.

Remark 2.45. It holds that $\mathsf{co}-\mathbf{NPC} \cap \mathbf{NPC} = \emptyset$, except $\mathbf{P} = \mathbf{NP}$. See [Vyg18].

Open Question 2.46. Is $\mathbf{P} = \mathbf{NP}$?

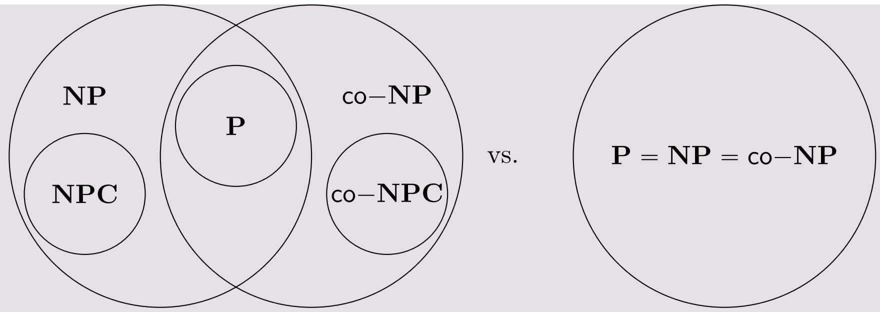


Figure 3: Landscape of problems depending if $\mathbf{P} = \mathbf{NP}$

3 Complexity differences between IP and LP

Some 0-1-IPs are easy, such as bipartite matching, or the assignment problem, even though IP $\in$ **NPC**. In the following section, we want to give intuition, why there are different complexities in IP.

3.1 Optimization vs. Separation

Recall (Ellipsoid Algorithm)**.** As discussed in ADM1, the **ellipsoid method** can be used to determine feasibility of LPs in polynomial time. One could also call the ellipsoid method a fancy "n-dimensional binary search". A rough draft how the algorithm worked:

1. Reduce optimization version to decision version and introduce bound $L = mn \cdot \log(\text{max abs. data})$
2. Volume-based argument: If the LP is feasible, there is a solution within the centered cube with length 2^L.
3. Volume is zero: Perturb the problem to $Ax \leqslant b + 2^{-L}$, which maintains feasibility, but now has positive volume.

For details, refer to the slides from ADM1.

Definition 3.1. Given a polyhedron P with an associated cost function. We want to introduce two distinct problem types:

- The **optimization problem** OPT denotes the problem of finding an optimal $x^* \in P$.
- The **separation problem** SEP denotes the problem of deciding if $x \in P$, or else stating a separating hyperplane.

Remark 3.2. The key step of the ellipsoid method is to find a hyperplane that separates the current x from the considered polyhedron. Especially, if $\mathsf{SEP} \in \mathbf{P}$, then $\mathsf{OPT} \in \mathbf{P}$. The converse can also be proven.

Definition 3.3. Let $\mathcal{Q}$ be the class of full-dimensional polytopes with 0 inside. We define the **polar** Q^* for $Q \in \mathcal{Q}$ as

$$Q^* := \{y \in \mathbb{R}^n \mid \forall x \in Q \colon y^T x \leqslant 1\}.$$

Theorem 3.4. Considering this class of polytopes, one can proove [Vyg18, Ch. 4, Thm. 4.22]:

1. Q^* is also a full-dimensional polytope with 0.
2. $(Q^*)^* = Q$
3. v is a vertex of Q iff $v^T y \leqslant 1$ is a facet of Q^*

Theorem 3.5. Suppose we can solve OPT on $\mathcal{Q} \in \mathbf{P}$ with algorithm A. Then we can use A as an oracle to solve SEP on $\mathcal{Q}^*$ in polynomial time.

Proof. Suppose $Q^* \in \mathcal{Q}^*$, and we want to separate y^0. Use A to solve OPT on Q with objective function $\max(y^0)^T x$ to get $x^* \in Q$. This yields two cases:

- $(y^0)^T x^* \leqslant 1$: Then this holds for all $x \in Q$, and thus $y^0 \in Q^*$ by definition.
- $(y^0)^T x^* > 1$: Consider hyperplane $(x^*)^T y$. From $x^* \in Q$ it follows that for all $y \in Q^*$, that $(x^*)^T y \leqslant 1$, but $(x^*)^T y^0 > 1$. Thus, we found a separating hyperplane.

□

Theorem 3.6. $\mathsf{SEP} \in \mathbf{P}$ for $\mathcal{Q}$ iff $\mathsf{OPT} \in \mathbf{P}$ for $\mathcal{Q}$

Proof. Using what we proven so far:

$$\begin{aligned} &\mathsf{OPT} \in \mathbf{P} \text{ for } \mathcal{Q} \overset{3.5}{\Longrightarrow} \mathsf{SEP} \in \mathbf{P} \text{ for } \mathcal{Q}^* \\ \overset{\text{Ellips.}}{\Longrightarrow} &\mathsf{OPT} \in \mathbf{P} \text{ for } \mathcal{Q}^* \overset{3.5}{\Longrightarrow} \mathsf{SEP} \in \mathbf{P} \text{ for } (\mathcal{Q}^*)^* = \mathcal{Q} \\ \overset{\text{Ellips.}}{\Longrightarrow} &\mathsf{OPT} \in \mathbf{P} \text{ for } \mathcal{Q} \end{aligned}$$

□

Conclusion 3.7. There is a close relationship between OPT and SEP:

$$\underbrace{\boxed{\text{Complexity}}}_{\mathsf{OPT}} \iff \underbrace{\boxed{\text{Integrality}}}_{\mathsf{SEP}}$$

Lecture 6
Do 05 May 2022

Theorem 3.8 (Minkowski)**.** For a polyhedron P it holds $x \in P$ iff there exist vertices $v_1, ..., v_k$ and rays $r_1, ..., r_l$, such that

$$\sum_i \lambda v_i + \sum_j \mu_j r_j = x$$
$$\sum_i \lambda_i = 1$$
$$\lambda, \mu \geqslant 0.$$

Minkowski's Theorem is also known as **Resolution Theorem**.

Proof. See ADM1. □

Definition 3.9. We have different variants of representing a polyhedron P:

- The **H-representation** (from "hyperplane") is given by $P = \{x \mid Ax \leqslant b\}$.
- The **V-representation** (from "vertex") is given by Theorem 3.8.

add ref HW?

Conclusion 3.10. Depending on the representation we have, we have different ways to solve OPT and SEP:

	H-representation	V-representation
OPT	LP Simplex/Ellipsoid	Brute Force
SEP	Brute Force	LP (Homework)

Example 3.11. Consider the n-cube $C^n := \{x \in \mathbb{R}^n \mid -1 \leqslant x_i \leqslant 1\}$. It has $2n$ facets, but 2^n vertices.

Now, consider the polar of C^n, which can be shown to be the n-octahedron O^n. Remember the intuition, that the polar exchanges vertices with facets. Indeed it holds that now, we have 2^n facets, but only $2n$ vertices.

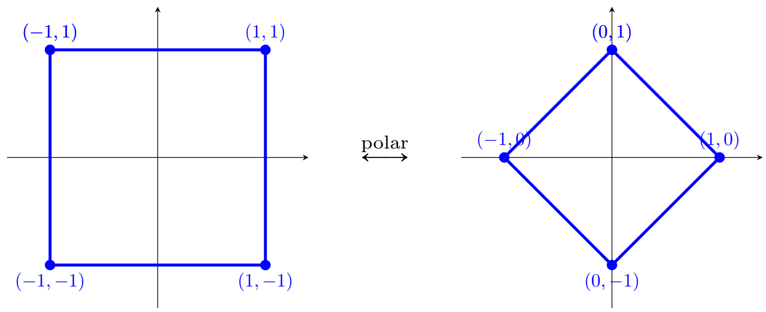


Figure 4: 2-cube vs. 2-octahedron

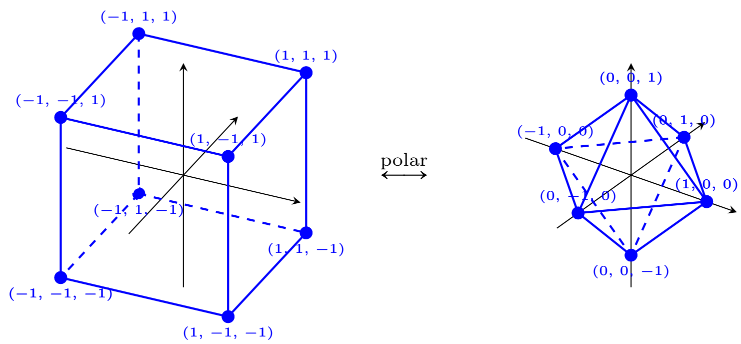


Figure 5: 3-cube vs. 3-octahedron

Information. Polymake is a tool for converting programmatically between H-representation and V-representation.

3.2 Certificate construction

Question 3.12. Consider the problem of finding a solution x to a linear or integer system. How do we construct succinct certificates of feasibility and infeasibility?

In order to answer this question, we will have to look at different kinds of systems each on their own. For the following theorems, we will consider two different systems every time, and show that the solutions can be used as the certificate we seek for.

Theorem 3.13. Exactly one of the following systems is feasible:

$$Ax = b \qquad \text{vs.} \qquad \begin{aligned} y^T A &= 0 \\ y^T b &= 1 \end{aligned}$$

In particular, the left system delivers a certificate of feasibility, whereas the right a certificate of infeasibility.

Proof. Suppose both are feasible. Then we have solutions y^0, x^0, and can construct following contradiction:

$$\begin{aligned} Ax^0 &= b \\ \Leftrightarrow \quad \underbrace{(y^0)^T A}_{0} x^0 &= (y^0)^T b = 1 \end{aligned}$$

It remains to prove at least one system is feasible. We can use **Gaussian Elimination** for that: Gaussian Elimination either yields a solution x^0 we can use as a succinct certificate for feasibility, or determine it is infeasible by yielding the row multiplier y^0 in order to generate $0^T x = 1$ as a succinct certificate of infeasibility. □

Warning. Gaussian Elimination is *not* polynomial in its natural variant because of numbers generated during the algorithm. Nonetheless, if careful and using certain tricks, Gaussian Elimination is polynomial.

Remark 3.14. Theorems stating that exactly one of two systems have a solution are called **Theorem of the Alternative**.

Theorem 3.15 (Farkas Lemma)**.** Exactly one of the following systems is feasible:

$$Ax \leqslant b \qquad \text{vs.} \qquad \begin{aligned} y^T A &= 0 \\ y^T b &< 0 \\ y &\geqslant 0 \end{aligned}$$

This is also known as **Farkas Lemma**. In particular, the left system delivers a certificate of feasibility, whereas the right a certificate of infeasibility.

Proof. Suppose both are feasible. Analoguous to previous proof we can see the contradiction:

$$\begin{aligned} Ax^0 &\leqslant b \\ \Leftrightarrow \quad \underbrace{(y^0)^T A}_{0} x^0 &\leqslant (y^0)^T b < 0 \end{aligned}$$

At least one system is feasible, which we can see by using the Ellipsoid Method to get a feasible x as a feasibility certificate for $Ax \leqslant b$. Otherwise, $Ax \leqslant b$ is infeasible. In that case we can use Phase 1 of the Simplex Method to generate $0^T x \leqslant z$ (for some $z \in \mathbb{Z}^-$) and use the extracted row multipliers y as an infeasibility certificate. Note that this y solves the right system. □

Conclusion 3.16. We already knew from Theorem 2.15 and Theorem 2.43, that previous feasibility problems both lie in $\mathbf{NP} \cap \mathsf{co}-\mathbf{NP}$. Thus, we found that Gaussian Elimination is the suspected polynomial algorithm.

Definition 3.17 (Diophantine equations)**.** An equation of the form $Ax = b$, for $x \in \mathbb{Z}^n$, is called **diophantine equation**.

Theorem 3.18. Exactly one of following systems is feasible:

$$\begin{aligned} Ax &= b \\ x &\in \mathbb{Z}^n \end{aligned} \qquad \text{vs.} \qquad \begin{aligned} y^T A &\in \mathbb{Z}^n \\ y^T b &\notin \mathbb{Z} \end{aligned}$$

In particular, the left system delivers a certificate of feasibility, whereas the right a certificate of infeasibility.

Proof. Suppose both are feasible. Then

$$\begin{aligned} & Ax^0 = b \\ \Leftrightarrow \quad & \underbrace{(y^0)^T A}_{\mathbb{Z}^n} \underbrace{x^0}_{\mathbb{Z}^n} = \underbrace{(y^0)^T b}_{\notin \mathbb{Z}} \end{aligned}$$

We can use the **Hermite Normal Form** algorithm to show that at least one system is feasible. Note that the Hermite Normal Form can be calculated in polynomial time. □

Problem. IP is defined as finding $x \in \mathbb{Z}^n$ for $Ax \leqslant b$. We have already shown that $\mathsf{IP} \in$ **NPC** (see Theorem 2.24), meaning that certificates cannot be calculated in polynomial time (unless $\mathbf{P} = \mathbf{NP}$).

Conclusion 3.19. Summing everything up, we can summarize our findings for calculation of feasibility certificates in following table:

	continuous	integer
$=$	Gaussian Elim./Phase 1	Hermite Normal Form
$\leqslant$	Linear Programming	not possible

The problem with integer inequality systems is its missing duality, e.g. there is no way of generating succinct certificates for verifying infeasibility, making it impossible to use the Theorem of the Alternative.

Remark 3.20. If we have an LP in standard form, we can also formulate a Theorem of the Alternative using Farkas Lemma:

$$\begin{aligned} Ax &= b \\ x &\geqslant 0 \end{aligned} \quad \Longleftrightarrow \quad \begin{aligned} Ax &\leqslant b \\ -Ax &\leqslant -b \\ -x &\leqslant 0 \end{aligned}$$

$$\overset{3.15}{\text{vs.}}$$

$$\begin{aligned} (y^1)^T A - (y^2)^T A - (y^3)^T &= 0 \\ (y^1)^T b - (y^2)^T b &< 0 \\ y^1, y^2, y^3 &\geqslant 0 \end{aligned} \quad \Longleftrightarrow \quad \begin{aligned} y^T A &\geqslant 0 \\ y^T b &< 0 \\ y &\text{ free} \end{aligned}$$

Note for the last equivalence that we used $y = y^1 - y^2$ with $y^1, y^2 \geqslant 0$, and interpreted y^3 as slack.

Theorem 3.21 (Gourdan)**.** Consider $Ax < 0$.

write Gourdan

Consider an LP with lower and upper bounds:

$$\begin{aligned} \min \quad & c^T x \\ Ax &= b \\ l \leqslant x &\leqslant u \end{aligned}$$

Decompose:

$$\begin{aligned} \min \quad & c^T x \\ Ax &= b \\ x &\geqslant l \\ -x &\geqslant u \\ x &\text{ free} \end{aligned}$$

Dualize:

$$\begin{aligned} \max \quad & b^T y + l^T \lambda - u^T \mu \\ y^T A + \lambda^T - \mu^T &= c^T \\ \lambda, \mu &\geqslant 0 \\ y &\text{ free} \end{aligned}$$

Rewrite

Lecture 7
Di 10 May 2022

Content lec07

Fact 3.22 (Convex duality)**.** If term j of the primal objective is $f_j(x_j)$ for convex f_j, then term j of the dual objective is $-f_j^\bullet(y_j)$, such that y_j is the dual of x_j.

We're using $f^\bullet$ as the **convex conjugate** of f, which has the property that $(f^\bullet)^\bullet = f$.

4 Approaches to IP

Even though IP is **NP**-complete, we still want to solve them as they model many real-world problems. For certain cases, though, we can use tricks to make calculation easier:

1. If the IP only has integer vertices for all b, we can just use LP.
2. If the IP only has integer vertices for a single useful b, we can at least use LP for this b, and might derive useful information anyway.
3. We could get a direct combinatorial algorithm that doesn't use LP. referring to duality SEP OPT
4. For a *fixed* (small) dimension, we can solve IP in polynomial time.
5. If we are only interested in *good* solutions, we could use approximation algorithms and heuristics.
6. Alternatively, solve the relaxed LP and round to an IP solution.
7. We can relax "bad" constraints and variables.
8. Just use Cutting Planes.

4.1 Integer-optimal solutions in LP

Recall. In ADM1 we proved that there are combinatorial problems that can be solved using LP nonetheless, e.g. Max-Flow, Min-Cut, Bipartite Matching, Min-Cost-Flow etc.

Question 4.1. Why do exactly these problems have the property of integer vertices?

Given an optimal vertex solution $x^* = (x_B^*, 0) = (B^{-1}b, 0)$ with basis B to an LP. By **Cramer's Rule**, it holds for all $j \in B$, that

$$x_j^* = \frac{\overbrace{\det(B_1, B_2, ..., b_j, ..., B_n)}^{\text{integer}}}{\det(B)}.$$

Thus, if $|\det(B)| = 1$, then x^* is integer.

Definition 4.2 (Totally unimodular)**.** A matrix A is **totally unimodular**, if for all square submatrices B of A it holds that $\det(B) \in \{-1, 0, 1\}$.

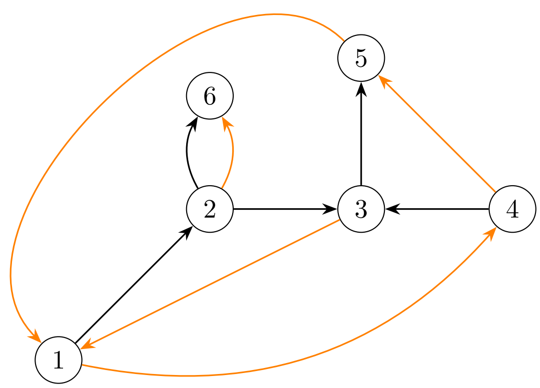
Note. Obiviously, A itself must consist only of $\{-1, 0, 1\}$ entries in order to be totally unimodular.

Theorem 4.3. Given A is totally unimodular. Then all optimal vertices x^* are integer for all righthandside b's. Conversely, if all vertices of $\{x \mid Ax \leqslant b, x \geqslant 0\}$ are integer for all righthandside b, then A is totally unimodular.

Proof. See [Wol99, Thm 2.5 III 1.2]. □

Definition 4.4. Let $G = (N, A)$ be a directed graph, T a spanning tree of G, and $S \subseteq A$. We construct a matrix $D \in \mathbb{R}^{|N-1|,|S|}$, such that every column corresponds to an arc $(u, v) \in S$, and every row to an arc in T. Consider the undirected (unique) path from u to v in T. We set in each column every entry to 1, where we used the arc as supposed, to -1, if we used the arc backwards, and 0 otherwise. Then D is called a **tree-path**, or **network matrix**.

Example 4.5. Given following graph:



Then a network matrix of this graph is given by

$$\begin{array}{c|c} & \begin{array}{ccccc} 1 \to 4 & 2 \to 6 & 3 \to 1 & 4 \to 5 & 5 \to 1 \end{array} \\ \begin{array}{c} 1 \to 2 \\ 2 \to 3 \\ 3 \to 6 \\ 3 \to 5 \\ 4 \to 3 \end{array} & \begin{pmatrix} 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Theorem 4.6. Any network matrix M is totally unimodular.

Proof. Every submatrix of a network matrix M is again a network matrix. Thus it suffices to show that every square network matrix M_s has $\det(M_s) \in \{-1, 0, 1\}$. We prove by induction over dimension d of M_s.

For $d = 1$ this is clear. Thus, consider the statement true for some d. Let node l be a leaf of the spanning tree T, and consider row $l \to k$. Using a case distinction:

- 0 arcs in S hit l. Then row $l \to k$ is 0, and thus $\det(M_s) = 0$.
- Exactly 1 arc in S hits l. Then row $l \to k$ is a unit vector, and we
- Otherwise, there are at least 2 arcs in S that hit l.

□

finish proof

Corollary 4.7. A node-arc incidence matrix of a directed graph is totally unimodular.

Proof. □

Corollary 4.8. A node-edge incidence matrix of a directed graph is totally unimodular.

Proof. □

Definition 4.9. A 0-1-matrix A has the **consecutive ones-property** if the 1's in each row do not have any 0's between them, i.e. 0001111100.

Corollary 4.10. A matrix A with consecutive ones-property is totally unimodular.

Proof. Suppose T is a line of connected nodes. For each row, construct an arc from the first 1 to the last 1. Then A is a network matrix for this graph. □

Note that there are totally unimodular matrices that aren't network matrices. Furthermore, it's also possible to combine totally unimodular matrices to get new ones.

Theorem 4.11. Let A_1, A_2 be two totally unimodular matrices. Then

$$\left(\begin{array}{c|c} A_1 & 0 \\ \hline 0 & A_2 \end{array}\right)$$

is also totally unimodular.

Construct new TU

Theorem 4.12 (Seymour)**.** The set of totally unimodular matrices is fully defined by

- all network matrices,
- two additional 5×5 matrices, and
- three different composition operations, e.g. Theorem 4.11.

Conclusion 4.13. Network problems are the easiest IPs.

Lecture 8
Do 12 May 2022
Content lec08

Definition 4.14. Given a graph $G = (N, E)$ and cost vector $c \in \mathbb{R}^E$. Finding a spanning tree T such that $c(T)$ is minimal is called **Minimal Spanning Tree** problem, short MST.

Recall. In ADM1 we learned that we can use **Kruskal's algorithm** to find a MST in polynomial time. Kruskal sorted the edges and added edges in this order if it wouldn't create a cycle.

On the other hand, there is also an LP formulation:

$$\begin{aligned} \min_x \quad & c^T x \\ \text{s.t.} \quad & x(\gamma(K)) \leqslant |K| - 1, \quad K \subsetneq N, \\ & x(E) = n - 1, \\ & x \geqslant 0 \end{aligned}$$

Tom doesn't like the LP formulation, though, because

1. "min" and "$\leqslant$" just feel wrong,
2. rather than $x(\gamma(K))$, we should use $x(S) \leqslant ?$ for $S \subseteq E$.

We can circumvent this problems with following LP, for $w = M - c$, and some magic function r:

$$\begin{aligned} \min_w \quad & w^T x \\ \text{s.t.} \quad & x(S) \leqslant r(S), \quad S \subseteq E, \\ & x \geqslant 0 \end{aligned}$$

Let's have a closer look at $r(S)$, and define it as the maximum number of edges we can choose in S without creating a cycle.

Let's also define $cc(S)$ as the number of connected components in (N, S).

Theorem 4.15. For a graph $G = (N, E)$ it holds that $r(S) = n - cc(S)$.

Proof. Let $C_1, ..., C_k$ be node sets of connected components of (N, S), meaning $cc(S) = k$. Note that $\sum_i |C_i| = n$. Furthermore, we can choose at most $|C_i| - 1$ acyclic edges per component C_i by choosing any spanning tree. Thus,

$$\begin{aligned} r(S) = &\text{ maximum acyclic edges in } (N, S) \\ = &\sum_i (\text{maximum acyclic edges in } C_i) \\ = &\sum_i |C_i| - 1 = n - k = n - cc(S). \end{aligned}$$

□

Example 4.16. Consider following graph:

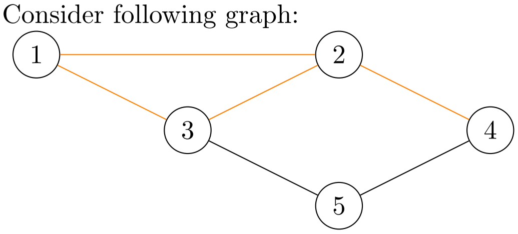


Figure 6: A graph with S colored orange

For the drawn S, it holds $cc(S) = 2$, namely $\{1, 2, 3, 4\}$ and $\{5\}$, and thus $r(S) = 5 - 2 = 3$.

We can construct the dual of our new LP:

$$\begin{aligned} \min_{y_S} \quad & r(S) y_S \\ \text{s.t.} \quad & \sum_{S : e \in S} y_S \geqslant w_e, \quad \forall e \in E, \\ & y \geqslant 0 \end{aligned}$$

Warning. Our original LP had 2^n constraints, which means our dual has 2^n variables. Even the data $r(S)$ cannot be written down in polynomial time!

This means we need to think about $r(S)$ a little bit more. Consider a set R and edge $e \in E$ with $R \subseteq S \subseteq S + e$. The so-called **marginal cost** of adding e to R is $r(R + e) - r(R) \in \{0, 1\}$. Same goes for S.

Definition 4.17. Let $r(S)$ be a generic function defined on $S \subseteq M$ of some set M. If the marginal costs are non-decreasing for $R \subseteq S$, if we add the same edge, i.e.

$$r(S + e) - r(S) \leqslant r(R + e) - r(R),$$

then we call r **submodular**.

Theorem 4.18. Our acyclic-$r(S)$ is submodular.

Proof. It suffices to show that $r(S+e) - r(S) = 1$ and $r(R+e) - r(R) = 0$ cannot happen. Note that the connected components of R are subsets of connected components of S. Thus, if we add an edge e that connects two connected components, it also connects two components of R. By Theorem 4.15, this is the only way $r(S)$ increases by one, but also forces $r(R)$ to increase. $\square$

There is also an equivalent definition of submodularity:

Theorem 4.19. A function $r : \mathcal{P}(M) \to \mathbb{R}$ is **submodular** iff for all $S, R \subseteq E$

$$r(S) + r(R) \geqslant r(S \cap R) + r(S \cup R).$$

Proof. See homework. $\square$

reference homework

Now let's try to derive Kruskal from our LP:

Theorem 4.20. Suppose x^*, y^* are optimal in their corresponding LP. Among the dual optimal solution let y^* be the one with $y^* = \sum_{S \subseteq E} (y_S)^2$ (notice the square!).

Then for $R, S \subseteq E$, if $y^*_R, y^*_S > 0$, it follows that either $R \subseteq S$ or $S \subseteq R$. We call this property **nested**.

Proof. Assume $y^*_R, y^*_S > 0$, but are not nested. Let $\varepsilon = \min(y^*_R, y^*_S 0) > 0$ and

$$y'_Q = \begin{cases} y^*_Q - \varepsilon, & \text{if } Q = R \vee Q = S, \\ y^*_Q + \varepsilon, & \text{if } Q = R \cup S \vee Q = R \cap S, \\ y^*_Q, & \text{otherwise.} \end{cases}$$

Then y' is still feasible, because the ε cancel out, since $R \cup S$ and $R \cap S$ must be new sets, and every edge is either in all four sets, no set, or exactly one of R, S and $R \cup S, R \cap S$ each. The choice of ε ensures $y' \geqslant 0$.

Now have a look how the objective function changes and consider the difference given by:

$$\varepsilon(\underbrace{r(R \cap S) + r(R \cup S) - r(R) - r(S)}_{\leqslant 0 \text{ by submodularity}})$$

We cannot get cheaper though, because we were already optimal. Therefore, y' is also an optimal dual solution, but

$$\sum_S (y'_S)^2 < \sum_S (y^*_S)^2$$

is a contradiction! (One can check this by tedious calculations.) $\square$

Consider $\mathcal{J} \coloneqq \{S \subseteq E \mid y_S^* > 0\}$, and build a chain

$$S_1 \subseteq \ldots \subseteq_m, |S_i| = i.$$

If this is not possible, add some $y_S^* = 0$ sets. Thus, using this S as a basis yields

$$\begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_m \end{array} \begin{array}{c} \begin{array}{cccccc} S_1 & S_2 & S_3 & \cdots\cdots & S_m \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & \cdots\cdots & 1 \\ & 1 & 1 & & 1 \\ & & 1 & & 1 \\ & & & \ddots & \vdots \\ & 0 & & & 1 \end{pmatrix} \end{array}$$

which has continuous ones property (and thus being totally unimodular by Corollary 4.10), and is an upper triangular matrix, meaning the resulting equation system is easy to solve.

Part II

Appendix

A Exercise sheets

1. exercise sheet

Exercise 1.1. Did on paper.

2. exercise sheet

Exercise 2.1. An correct ordering is given by:

$$
\begin{aligned}
\mathrm{O}\left(\varepsilon^{n}\right) &\subseteq \mathrm{O}\left(n^{\varepsilon-1}\right) \subseteq \mathrm{O}\left(n^{-\varepsilon}\right) \subseteq \mathrm{O}\left(\frac{\log n}{n^{\varepsilon}}\right) && (1)\\
&\subseteq \mathrm{O}\left(\frac{1}{\log n}\right) \subseteq \mathrm{O}\left(\frac{\log^{2} n}{\log n}\right) \subseteq \mathrm{O}\left(\frac{1}{\log^{2} n}\right) && (2)\\
&\subseteq \mathrm{O}\left(e^{\frac{1}{n}}\right) = \mathrm{O}\left(1\right) = \mathrm{O}\left(\left(1-\frac{1}{n}\right)^{n}\right) && (3)\\
&\subseteq \mathrm{O}\left(\log n\right) \subseteq \mathrm{O}\left(\frac{n^{\varepsilon}}{\log n}\right) \subseteq \mathrm{O}\left(n^{\varepsilon}\right) \subseteq \mathrm{O}\left(n^{\varepsilon}\log n\right) \subseteq \mathrm{O}\left(n^{1-\varepsilon}\right) && (4)\\
&\subseteq \mathrm{O}\left(\frac{n}{\log n}\right) \subseteq \mathrm{O}\left(n\log n\right) \subseteq \mathrm{O}\left(n^{2}\right) \subseteq \mathrm{O}\left(n^{2}\log n\right) \subseteq \mathrm{O}\left(n^{e}\right) && (5)\\
&\subseteq \mathrm{O}\left(n^{\log n}\right) \subseteq \mathrm{O}\left(e^{n}\right) \subseteq \mathrm{O}\left(\left(\log n\right)^{n}\right) \subseteq \mathrm{O}\left(n!\right) && (6)
\end{aligned}
$$

These can mostly achieved by the fact that $n^x \in \mathrm{O}(n^y)$ if $x \leqslant y$, and $(\log n) \cdot n^x \in \mathrm{O}(n^y)$ if $y > x$, otherwise the other way around. Additionally, it is often useful to consider the logarithm of the functions we compare, because it maintains monotonocity.

Exercise 2.2. Analoguous to the lecture we can introduce constraints, such that $y_{ij} = x_i \wedge x_j$:

$$
\begin{aligned}
y_{ij} &\leqslant x_i\\
y_{ij} &\leqslant x_j\\
y_{ij} &\geqslant x_i + x_j - 1\\
y_{ij} &\in [0, 1]
\end{aligned}
$$

Exercise 2.3. We can show that $f(x_1) = \max(c_1x_1, c_1p + c_2x_1 - c_2p)$ using a case distinction.

- $x_1 = p$: Trivial.
- $x_1 > p$: Consider $c_1 < c_2$. Multiplying by $x_1 - p$ (which is positive) and rearranging yields $c_1x_1 < c_1p + c_2x_1 - c_2p$.
- $x_1 < p$: Analoguous, but now $x_1 - p$ is negative, which reverses the inequality.

As shown in ADM1, the maximum of linear functions can be written as an LP by introducing a helper variable as follows:

$$\begin{aligned}
\min \quad & z + \sum_{i=2}^{n} c_i x_i \\
\text{s.t.} \quad & Ax = b \\
& l \leqslant x_1 \leqslant u \\
& x_2, ..., x_n \leqslant 0 \\
& z \geqslant c_1 x_1 \\
& z \geqslant c_1 p + c_2 x_1 - c_2 p
\end{aligned}$$

3. exercise sheet

Exercise 3.1.

1. We can show easily that $\mathsf{SPATH} \in \mathbf{P}$ by using the fact from ADM1, that breadth-first search started from s finds a shortest path to t in polynomial time. Therefore, if the shortest path has length $k^* \leqslant k$, we can return true, and false otherwise.

2. We first show $\mathsf{LPATH} \in \mathbf{NP}$: Suppose an instance of LPATH is true, then there is a path of at least length k. Therefore, we can simply use this path as a succinct certificate and verify in polynomial time that the path is indeed valid.

   It remains to show that we can reduce a **NP**-complete problem to LPATH. It suffices to show $\mathsf{UHAMPATH} \propto \mathsf{LPATH}$: Suppose we have an instance $((V, E), s, t)$ of $\mathsf{UHAMPATH}$. We can simply reduce it to the problem of finding a path of at least length $|V| - 1$ starting in s and ending in t, because every such path is indeed a hamiltonian path, because every vertex needs to be visited exactly once. Therefore, if there is a hamiltonian path, it is already a path of at least length $|V| - 1$. For the other direction, if there is a path of at least length $|V| - 1$, then it must visit every node exactly once in order to be a valid path.
   This shows that the reduction is Yes-preserving.

Exercise 3.2. If $\mathsf{DOUBLESAT}$ is true, then we can choose any two valid assignments as a succinct certificate and easily verify their correctness in polynomial time.

It remains to show $\mathsf{SAT} \propto \mathsf{DOUBLESAT}$: Starting from our SAT-instance, we can simply introduce two new variables a, b and a new clause $a \vee b$. This construction is Yes-preserving, because if the original instance is infeasible, the new instance still has no assignments. On the other hand, if there is a valid assignment in the original, then we now have at least 3 valid instances for different assignments of a and b.

Exercise 3.3. We notice that $a \vee b$ is equivalent to $\neg a \implies b$, and $\neg b \implies a$. By doing this for all clauses, we can construct a graph with the literals as vertices, and the implications as directed edges. Now, checking for each literal pair $l, \neg l$ if both can reach one another by a directed path suffices to show feasibility:

If previous condition holds true, then by logic it must hold that a feasible assignment satisfies $l \Leftrightarrow \neg l$, which is impossible. On the other hand, if this is never the case, then there must be a feasible solution:

We can construct this solution by iteratively setting either l or $\neg l$ to true, depending if $l \implies \neg l$ holds, and then also set every implied variable to true. If we would encounter a variable r which is already false, then $\neg r$ must be true, and therefore all further implications would need to be true by construction. Because $l \implies r$, also $\neg r \implies \neg l$, meaning that l would be already false - contradiction!
Therefore, our construction always works.

4. exercise sheet

Exercise 4.1. We're given an instance $(v^1, ..., v^k, x^0)$ as described. Iff $x_0 \in P$, then $x_0 \in \operatorname{conv}(v^1, \ldots, v^k)$. Thus there exist $\lambda_1, \ldots, \lambda_k$ such that

$$\sum_{i=1}^{k} \lambda_i v^i = x_0$$
$$\sum_{i=1}^{k} \lambda_i = 1,$$
$$\lambda \geqslant 0$$

which we can find by LP. Otherwise, the system is found infeasible, and we know there must exist c with $c^T x \geqslant c^T x^0$. We first show it suffices that this property holds for all vertices.

Suppose $c^T v^i \geqslant c^T x^0$. Let $x = \sum_i \lambda'_i v^i \in P$. Then

$$\begin{aligned} c^T x &= \sum_i \lambda'_i c^T v^i \\ &\geqslant \sum_i \lambda'_i c^T x^0 \\ &= c^T x^0. \end{aligned}$$

Note that we used $\lambda' \geqslant 0$ and $\sum_i \lambda'_i = 1$ in the second and third step.

Using

$$c^T v^i \geqslant c^t x^0 \Leftrightarrow (v^i - x^0)^T \cdot c \geqslant 0,$$

this means we can find c with following constraints via LP:

$$(v^i - x^0)^T \cdot c \geqslant 0, \quad i = 1, ..., k.$$

Exercise 4.2.

Index

Literature

[Orl93] Ravindra K. Ahuja; Thomas L. Magnanti; James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993. ISBN: 0-13-617549-X. URL: https://tu-berlin.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=TUB_ALMA_DS21506259970002884&context=L&vid=TUB&lang=de_DE&search_scope=TUB_ALL&adaptor=Local%20Search%20Engine&tab=tub_all&query=any,contains,network%20flows%20ahuja&offset=0.

[Vyg18] Bernhard Korte; Jens Vygen. *Kombinatorische Optimierung*. 3rd ed. Springer, 2018. ISBN: 978-3-662-57690-8. URL: https://link.springer.com/book/10.1007/978-3-662-57691-5.

[Wol99] George Nemhauser; Laurence Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1999. ISBN: 978-0-471-82819-8. URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9781118627372.