Discrete Optimization - ADM2

Lecturer
Prof. Dr. Tom McCormick

Assistant
Svenja Griesbach and Sarah Morell

Notes
Simon Cyrani

Version
git: d34131e
compiled: Monday 2nd May, 2022 21:40

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.

Contents

Summary of lectures

Part I

Lecture notes

Lecture 1
Di 19 Apr 2022

1 Test

I didn't make any notes.

Lecture 2
Di 19 Apr 2022

2 Modelling using IP

$$
\begin{aligned}
\text{s.t.} \quad & \lambda_1 \leqslant y_1 \\
& \lambda_i \leqslant y_{i-1} + y_i, && i > 1 \\
& \lambda_k \leqslant y_{k-1} \\
& \sum_{i=1}^{k-1} y_i = 1 \\
& y_i \in \mathbb{B}
\end{aligned}
$$

This allows λ_{i-1}, λ_i to be positive and rest negative. [1].

Now, given an IP Q could be formulated by *many* P's.

$$
\begin{aligned}
Q :& \{0000, 1000, 0100, 0010, 0110, 0101, 0011\} \\
P_1 =& \{x \in \mathbb{R}^4 \mid 93x_1 + 49x_2 + 37x_3 + 29x_4 \leqslant 111\} \\
P_2 =& \{x \in \mathbb{R}^4 \mid 2x_1 + x_2 + x_3 + x_4 \leqslant 2\} \\
P_3 =& \{x \in \mathbb{R}^4 \mid 2x_1 + x_2 + x_3 + x_4 \leqslant 2, \\
& x_1 + x_2 \leqslant 1, \\
& x_1 + x_3 \leqslant 1, \\
& x_1 + x_4 \leqslant 1\}
\end{aligned}
$$

Then, $P_3 \subsetneq P_2 \subsetneq P_1$.

Facility location

Consider following boolean variables:

$$
\begin{aligned}
m : \quad & y_i = \begin{cases} 1, & \text{open warehouse } i, \\ 0, & \text{else} \end{cases} \\
n : \quad & x_{ij} = \begin{cases} 1, & \text{if serve store } j \text{ from warehouse } i, \\ 0, & \text{else} \end{cases}
\end{aligned}
$$

[1] Ch 1: Nemhauser Wolsey

Complexity

Question: Easy vs. hard?
(some recap of big-$\mathcal{O}$-notation and $\mathcal{P}$ vs. $\mathcal{NP}$)

Lecture 3
Di 26 Apr 2022

3 Hardness

We cheat and restrict hardness to *decision problems*, that is, problems that can be answered by "Yes" or "No" only.

Example 3.1. Possible decision problems could be:

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

Question 3.2. 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 3.3. 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 3.4. 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 3.5. 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 3.6. 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! $\square$

Corollary 3.7. If we have an optimal oracle, then one can solve decision version in oracle-polynomial time using the optimal oracle.

Conclusion 3.8. Optimization and decision version differ only by a polynomial factor of complexity. Therefore, either both are easy or both are hard.

Definition 3.9 (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 3.10 (**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 3.11. 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 3.12. 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 3.13. $\mathbf{P} \subseteq \mathbf{NP}$

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 3.14. $\mathsf{LP} \in \mathbf{NP}$, 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 3.15 (Satisfiability problem, SAT)**.** Consider n logical variables $v_1, ..., v_n$, allowing also the negated literals $\overline{v_i}$. Additionally, we have m clauses $C_1, ..., C_m$, 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 3.16. 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 3.17. SAT $\in$ **NP**

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

Theorem 3.18 (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. □

Lecture 4
Do 28 Apr 2022

This means SAT is the hardest problem in **NP**.

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

Example 3.19. Translating from Example 3.16:

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 3.20 (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 3.21 (**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**.

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 3.22. 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. It was previously shown this encodes exactly the logic formula. □

insert reference lec01

Definition 3.23 (3SAT)**.** We define 3SAT as a variant of SAT where we only allow clauses with exactly 3 literals, e.g. $|C_j| = 3$.

Theorem 3.24. 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 3.25 (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 3.26. 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:

Consider an instance of 3SAT and construct a graph as shown in *Figure* 1, 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.

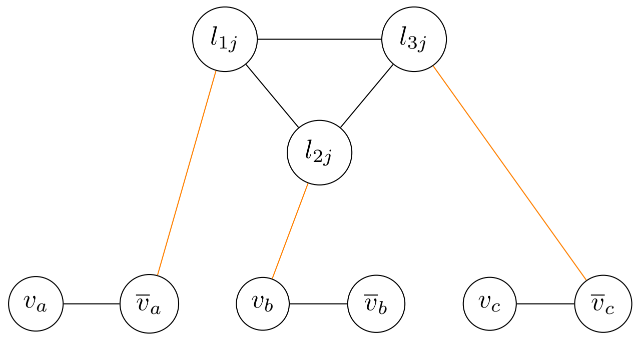

Figure 1: Schema of how to use triangle "gadgets" for a single clause C_j

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 3.27. NC in bipartite graphs is in **P**.

Definition 3.28 (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 3.29. 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 3.30. 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 3.31 (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 3.32. PART $\in$ **NPC**

Proof Sketch. We can show [Ber18, Ch. 15.5]:

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

Remark 3.33. Still, PART has a pseudopolynomial algorithm using dynamic programming.

Definition 3.34. If a (numerical) problem is only **NP**-complete if it is dependent on the size of the numbers (e.g. exponentially in count of numbers), we call it **weakly NP-complete**. Otherwise, we call it **strongly NP-complete**.

Definition 3.35 (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 3.36. 3SAT is strongly **NP**-complete.

Proof. □

proof 3part NPC

Remark 3.37. Only weakly **NP**-complete problems could have pseudopolynomial algorithms (except **P** = **NP**).

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:

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

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:

$$
\begin{aligned}
\text{min} \quad & z + \sum_{i=2}^{n} c_i x_i \\
\text{s.t.} \quad & Ax = b \\
& l \leqslant x_1 \leqslant u \\
& x_2, ..., x_n \leqslant 0 \\
& z \geqslant c_1x_1 \\
& z \geqslant c_1p + c_2x_1 - c_2p
\end{aligned}
$$

Index

Literature

[Ber18] Jens Vygen Bernhard Korte. *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.