Introduction to integer programming II

Martin Branda

Charles University
Faculty of Mathematics and Physics
Department of Probability and Mathematical Statistics

COMPUTATIONAL ASPECTS OF OPTIMIZATION

Introduction to complexity theory

Wolsey (1998): Consider **decision problems** having YES–NO answers.
Optimization problem

$$\max_{x \in M} c^T x$$

can be replaced by (for some k integral)

Is there an $x \in M$ with value $c^T x \geq k$?

For a problem instance X, the **length of the input** $L(X)$ is the length of the binary representation of a standard representation of the instance.
Instance $X = \{c, M\}$, $X = \{c, M, k\}$.

Example: Knapsack decision problem

For an instance

$$X = \left\{ \sum_{i=1}^{n} c_i x_i \geq k,\ \sum_{i=1}^{n} a_i x_i \leq b,\ x \in \{0,1\}^n \right\},$$

the length of the input is

$$L(X) = \sum_{i=1}^{n} \lceil \log c_i \rceil + \sum_{i=1}^{n} \lceil \log a_i \rceil + \lceil \log b \rceil + \lceil \log k \rceil$$

Running time

Definition

- $f_A(X)$ is the **number of elementary calculations** required to run the algorithm A on the instance $X \in P$.
- **Running time of the algorithm** A

$$f_A^*(l) = \sup_X \{ f_A(X) : \ L(X) = l \}.$$

- An algorithm A is **polynomial** for a problem P if $f_A^*(l) = O(l^p)$ for some $p \in \mathbb{N}$.

Classes $\mathcal{NP}$ and $\mathcal{P}$

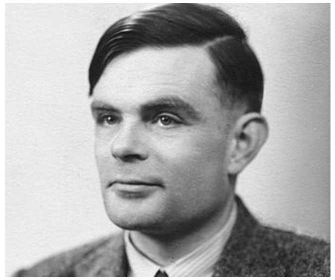
Definition

- $\mathcal{NP}$ (Nondeterministic Polynomial) is the class of decision problems with the property that: for any instance for which the answer is YES, there is a polynomial proof of the YES.
- $\mathcal{P}$ is the class of decision problems in $\mathcal{NP}$ for which there exists a polynomial algorithm.

$\mathcal{NP}$ may be equivalently defined as the set of decision problems that can be solved in polynomial time on a non-deterministic Turing machine[1].

[1]NTM writes symbols one at a time on an endless tape by strictly following a set of rules. It determines what action it should perform next according to its internal state and what symbol it currently sees. It may have a set of rules that prescribes more than one action for a given situation. The machine "branches" into many copies, each of which follows one of the possible transitions leading to a "computation tree".

Alan Turing

The Imitation Game (2014)

Polynomial reduction and the class $\mathcal{NPC}$

Definition

- If problems $P, Q \in \mathcal{NP}$, and if an instance of P can be converted in polynomial time to an instance of Q, then P is **polynomially reducible** to Q.
- $\mathcal{NPC}$, the class of $\mathcal{NP}$-**complete** problems, is the subset of problems $P \in \mathcal{NP}$ such that for all $Q \in \mathcal{NP}$, Q is polynomially reducible to P.

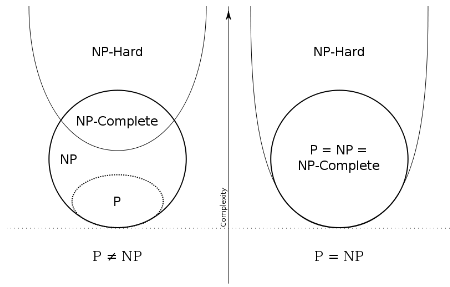
Proposition: Suppose that problems $P, Q \in \mathcal{NP}$.

- If $Q \in \mathcal{P}$ and P is polynomially reducible to Q, then $P \in \mathcal{P}$.
- If $P \in \mathcal{NPC}$ and P is polynomially reducible to Q, then $Q \in \mathcal{NPC}$.

Proposition: If $\mathcal{P} \cap \mathcal{NPC} \neq \emptyset$, then $\mathcal{P} = \mathcal{NPC}$.

Open question & Euler diagram

Is $\mathcal{P} = \mathcal{NP}$?



$\mathcal{NP}$-hard optimization problems

Definition

An optimization problem for which the decision problem lies in $\mathcal{NPC}$ is called $\mathcal{NP}$-hard.

Simplex algorithm

Klee–Minty (1972) example:

$$\begin{aligned} \max \ & \sum_{j=1}^{n} 10^{n-j} x_j \\ \text{s.t. } & 2 \sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \le 100^{i-1}, \ i = 1, \dots, n, \\ & x_j \ge 0, \ j = 1, \dots, n. \end{aligned} \tag{1}$$

Can be easily reformulated in the standard form. The Simplex algorithm takes $2^n - 1$ **pivot steps**, i.e. it is not polynomial in the worst case.

Branch-and-Bound

Basic idea: **DIVIDE AND RULE**

Let $M = M_1 \cup M_2 \cup \dots \cup M_r$ be a partitioning of the feasibility set and let

$$f_j = \min_{x \in M_j} f(x).$$

Then

$$\min_{x \in M} f(x) = \min_{j=1,\dots,r} f_j.$$

Branch-and-Bound

General principles:

- Solve LP problem without integrality only.
- Branch using additional constraints on integrality: $x_i \le \lfloor x_i^* \rfloor$, $x_i \ge \lfloor x_i^* \rfloor + 1$.
- Cut inperspective branches before solving (using bounds on the optimal value).

Branch-and-Bound

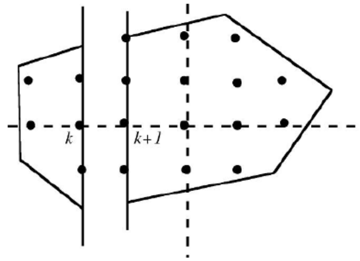
General principles:

- Solve only LP problems with relaxed integrality.
- **Branching**: if an optimal solution is not integral, e.g. $\hat{x}_i$, create and save two new problems with constraints $x_i \leq \lfloor \hat{x}_i \rfloor$, $x_i \geq \lceil \hat{x}_i \rceil$.
- **Bounding** ("different" cutting): save the objective value of the best integral solution and cut all problems in the queue created from the problems with higher optimal values[2].

Exact algorithm ..

[2] Branching cannot improve it.

Branch-and-Bound



P. Pedegral (2004)

Branch-and-Bound

0. $f_{min} = \infty$, $x_{min} = \cdot$, list of problems $P = \emptyset$
   Solve LP-relaxed problem and obtain f^*, x^*. If the solution is integral, STOP. If the problem is infeasible or unbounded, STOP.
1. **BRANCHING**: There is x_i^* basic non-integral variable such that $k < x_i^* < k+1$ for some $k \in \mathbb{N}$:
   - Add constraint $x_i \leq k$ to previous problem and put it into list P.
   - Add constraint $x_i \geq k+1$ to previous problem and put it into list P.
2. Take problem from P and solve it: f^*, x^*.
3.
   - If $f^* < f_{min}$ and x^* is non-integral, GO TO 1.
   - **BOUNDING**: If $f^* < f_{min}$ a x^* is integral, set $f_{min} = f^*$ a $x_{min} = x^*$, GO TO 4.
   - **BOUNDING**: If $f^* \geq f_{min}$, GO TO 4.
   - Problem is infeasible, GO TO 4.
4.
   - If $P \neq \emptyset$, GO TO 2.
   - If $P = \emptyset$ a $f_{min} = \infty$, integral solution does not exist.
   - If $P = \emptyset$ a $f_{min} < \infty$, optimal value and solution are f_{min}, x_{min}.

Better ...

2./3. Take problem from list P and solve it: f^*, x^*. If for the optimal value of the current problem holds $f^* \geq f_{min}$, then the branching is not necessary, since by solving the problems with added branching constraints we can only increase the optimal value and obtain the same f_{min}.

Branch-and-Bound

Algorithmic issues:

- **Problem selection from the list** P: FIFO, LIFO (depth-first search), problem with the smallest f^*.
- **Selection of the branching variable** x_i^*: the highest/smallest violation of integrality OR the highest/smallest coefficient in the objective function.

B&B – Example I

$$\begin{aligned} \min 4x_1 + 5x_2 & \\ x_1 + 4x_2 &\geq 5, \\ 3x_1 + 2x_2 &\geq 7, \\ x_1, x_2 &\in \mathbb{Z}_+. \end{aligned}$$

After two iterations of the dual SIMPLEX algorithm ...

			4	5	0	0
			x_1	x_2	x_3	x_4
5	x_2	8/10	0	1	-3/10	1/10
4	x_1	18/10	1	0	2/10	-4/10
		112/10	0	0	-7/10	-11/10

B&B – Example I

Branching means adding a cut of the form $x_1 \leq 1$, i.e.

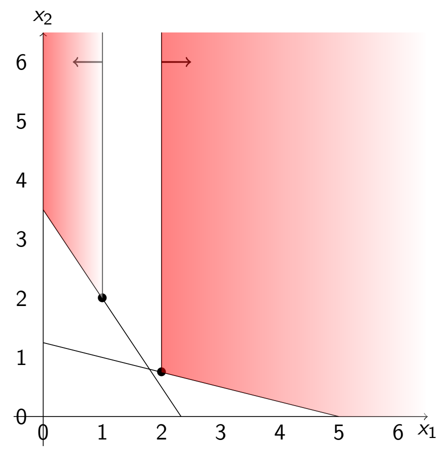
$$x_1 + x_5 = 1, \; x_5 \geq 0.$$

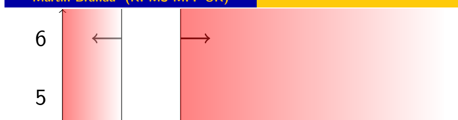
$(\alpha = (1, 0, 0, 0, 1), \; \alpha_B = (1, 0))$

			4	5	0	0	0
			x_1	x_2	x_3	x_4	x_5
5	x_2	8/10	0	1	-3/10	1/10	0
4	x_1	18/10	1	0	2/10	-4/10	0
0	x_5	-8/10	0	0	- 2/10	4/10	1
		112/10	0	0	-7/10	-11/10	0

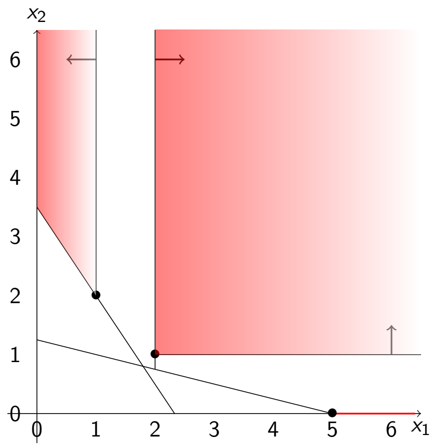
Dual feasible, primal infeasible – run the dual simplex ...

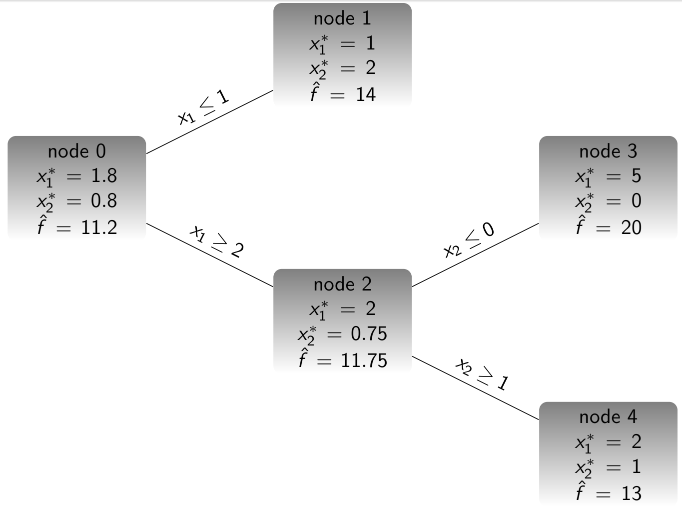
Branch-and-Bound





Branch-and-Bound

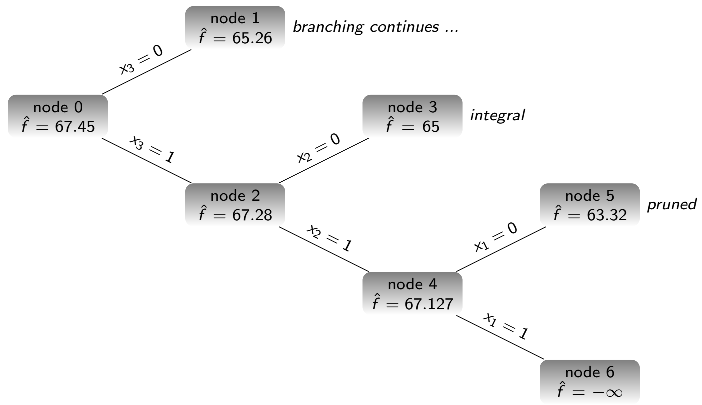




B&B – Example II

$$
\begin{aligned}
\max\ & 23x_1 + 19x_2 + 28x_3 + 14x_4 + 44x_5 \\
\text{s.t. } & 8x_1 + 7x_2 + 11x_3 + 6x_4 + 19x_5 \leq 25, \\
& x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}.
\end{aligned}
$$

B&B – Example II



Branch-and-Bound – remarks

- If you are able to get a **feasible solution** quickly, deliver it to the software (solver).
- **Branch-and-Cut**: add cuts at the beginning/during B&B.
- **Algorithm termination**: (Relative) difference between a lower and an upper bound – construct the upper bound (for minimization) using a feasible solution, lower bound …

Branch-and-Bound

Lower bound

Construction of the lower bound:

Let $M = M_1 \cup M_2 \cup \dots \cup M_r$ be a partitioning of the feasibility set and let

$$\underline{f}_j = \min_{x \in M_j} f(x)$$

be a lower bound for each subproblem. Then

$$\min_{x \in M} f(x) \geq \min_{j=1,\dots,r} \underline{f}_j$$

is a lower bound for the optimal value.

Duality

Set $S(b) = \{x \in \mathbb{Z}^n_+ : \; Ax = b\}$ and define the **value function**

$$z(b) = \min_{x \in S(b)} c^T x. \tag{2}$$

A dual function $F : \mathbb{R}^m \to \overline{\mathbb{R}}$

$$F(b) \leq z(b), \; \forall b \in \mathbb{R}^m. \tag{3}$$

A general form of **dual problem**

$$\max_F \{F(b) : \; \text{s.t. } F(b) \leq z(b), \; b \in \mathbb{R}^m, \; F : \mathbb{R}^m \to \mathbb{R}\}. \tag{4}$$

We call F a **weak dual** function if it is feasible, and **strong dual** if moreover $F(b) = z(b)$.

Duality

A function F is **subadditive** over a domain Θ if

$$F(\theta_1 + \theta_2) \leq F(\theta_1) + F(\theta_2)$$

for all $\theta_1 + \theta_2, \theta_1, \theta_2 \in \Theta$.

The value function z is subadditive over $\{b : S(b) \neq \emptyset\}$, since the sum of optimal x's is feasible for the problem with $b_1 + b_2$ r.h.s., i.e. $\hat{x}_1 + \hat{x}_2 \in S(b_1 + b_2)$.

Duality

If F is subadditive, then condition $F(Ax) \le c^T x$ for $x \in \mathbb{Z}_+^n$ is equivalent to $F(a_{\cdot j}) \le c_j$, $j = 1, \dots, m$.

This is true since $F(Ae_j) \le c^T e_j$ is the same as $F(a_{\cdot j}) \le c_j$.

On the other hand, if F is subadditive and $F(a_{\cdot j}) \le c_j$, $j = 1, \dots, m$ imply

$$F(Ax) \le \sum_{j=1}^{m} F(a_{\cdot j}) x_j \le \sum_{j=1}^{m} c_j x_j = c^T x.$$

Duality

If we set

$$\Gamma^m = \{F : \mathbb{R}^m \to \mathbb{R},\ F(0) = 0,\ F \text{ subadditive}\},$$

then we can write a **subadditive dual** independent of x:

$$\max_F \{F(b) :\ \text{s.t. } F(a_{\cdot j}) \le c_j,\ F \in \Gamma^m\}. \tag{5}$$

Weak and strong duality holds.

An easy feasible solution based on LP duality (= weak dual)

$$F_{LP}(b) = \max_y b^T y \text{ s.t. } A^T y \le c. \tag{6}$$

Duality

Complementary slackness condition: if $\hat{x}$ is an optimal solution for IP, and $\hat{F}$ is an optimal subadditive dual solution, then

$$(\hat{F}(a_{\cdot j}) - c_j)\hat{x}_j = 0,\ j = 1, \dots, m.$$

Knapsack problem

$$\begin{aligned} \max\ & \sum_{i=1}^{n} c_i x_i \\ \text{s.t. } & \sum_{i=1}^{n} a_i x_i \le b, \\ & x_i \in \{0, 1\}. \end{aligned}$$

Dynamic programming

Let a_i, b be positive integers.

$$f_r(\lambda) = \max \sum_{i=1}^{r} c_i x_i$$
$$\text{s.t.} \sum_{i=1}^{r} a_i x_i \le \lambda,$$
$$x_i \in \{0,1\}.$$

If

- $\hat{x}_r = 0$, then $f_r(\lambda) = f_{r-1}(\lambda)$,
- $\hat{x}_r = 1$ then $f_r(\lambda) = c_r + f_{r-1}(\lambda - a_r)$.

Thus we arrive at the recursion

$$f_r(\lambda) = \max \{f_{r-1}(\lambda), c_r + f_{r-1}(\lambda - a_r)\}.$$

Dynamic programming

0. Start with $f_1(\lambda) = 0$ for $0 \le \lambda < a_1$ and $f_1(\lambda) = \max\{0, c_1\}$ for $\lambda \ge a_1$.
1. Use the **forward recursion**
$$f_r(\lambda) = \max \{f_{r-1}(\lambda), c_r + f_{r-1}(\lambda - a_r)\}.$$
to successively calculate $f_2, \ldots, f_n$ for all $\lambda \in \{0, 1, \ldots, b\}$; $f_n(b)$ is the optimal value.
2. Keep indicator $p_r(\lambda) = 0$ if $f_r(\lambda) = f_{r-1}(\lambda)$, and $p_r(\lambda) = 1$ otherwise.
3. Obtain the optimal solution by a **backward recursion**: if $p_n(b) = 0$ then set $\hat{x}_n = 0$ and continue with $p_{n-1}(b)$, else (if $p_n(b) = 1$) set $\hat{x}_n = 1$ and continue with $p_{n-1}(b - a_n)$ ...

Knapsack problem

Values $a_1 = 4$, $a_2 = 6$, $a_3 = 7$, costs $c_1 = 4$, $c_2 = 5$, $c_3 = 11$, budget $b = 10$:

$$\max \sum_{i=1}^{3} c_i x_i$$
$$\text{s.t.} \sum_{i=1}^{3} a_i x_i \le 10,$$
$$x_i \in \{0,1\}.$$

Knapsack problem – Dynamic programming

$a_1 = 4$, $a_2 = 6$, $a_3 = 7$, $c_1 = 4$, $c_2 = 5$, $c_3 = 11$

	r/ λ	0	1	2	3	4	5	6	7	8	9	10
	1	0	0	0	0	4	4	4	4	4	4	4
f_r	2	0	0	0	0	4	4	5	5	5	5	9
	3	0	0	0	0	4	4	5	11	11	11	11
	1	0	0	0	0	1	1	1	1	1	1	1
p_r	2	0	0	0	0	0	0	1	1	1	1	1
	3	0	0	0	0	0	0	0	1	1	1	1

Other successful applications: Uncapacitated lot-sizing problem, Shortest path problem.

Literature

- V. Klee, G.J. Minty, (1972). How good is the simplex algorithm?. In Shisha, Oved. Inequalities III (Proceedings of the Third Symposium on Inequalities held at the University of California, Los Angeles, Calif., September 1–9, 1969). New York-London: Academic Press, 159–175.
- G.L. Nemhauser, L.A. Wolsey (1989). Integer Programming. Chapter VI in Handbooks in OR & MS, Vol. 1, G.L. Nemhauser et al. Eds.
- P. Pedegral (2004). Introduction to optimization, Springer-Verlag, New York.
- L.A. Wolsey (1998). Integer Programming. Wiley, New York.
- L.A. Wolsey, G.L. Nemhauser (1999). Integer and Combinatorial Optimization. Wiley, New York.
- Northwestern University Open Text Book on Process Optimization, available online: https://optimization.mccormick.northwestern.edu [2017-03-19]