NMEK436

Computational Aspects of Optimization

Course Notes

Martin Branda

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

Version: December 15, 2017

Contents

1 Linear programming

To investigate the simplex algorithms, we will consider a linear programming problem in the standard form

$$\begin{aligned} \min\ & c^T x \\ \text{s.t. } & Ax = b, \\ & x \geq 0, \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$. Assume that $h(A) = h(A|b) = m$. We denote the set of feasible solutions by

$$M = \{x \in \mathbb{R}^n : \ Ax = b,\ x \geq 0\}.$$

We know that the set can be decomposed as $M = K + P$, where

- **Convex polyhedron** P – uniquely determined by its vertices (convex hull)
- **Convex polyhedral cone** K – generated by extreme directions (positive hull)

The Direct method is based on evaluating all vertices and extreme directions, computing the values of the objective function and verifying the optimality condition.

One of these cases is valid:

1. $M = \emptyset$,
2. $M \neq \emptyset$: the problem is unbounded,
3. $M \neq \emptyset$: the problem has an optimal solution (at least one of the optimal solutions is a vertex).

1.1 Primal simplex algorithm

The simplex algorithm was introduces by George B. Dantzig (1914–2005).

Basis B = regular square submatrix of A, i.e. A can be divided into the basis and nonbasis part

$$A = (B|N).$$

We also consider $B = \{i_1, \ldots, i_m\}$ as the set of column indices which correspond to the basis. We split also the objective coefficients and the decision vector accordingly:

$$\begin{aligned} c^T &= (c_B^T, c_N^T), \\ x^T(B) &= (x_B^T(B), x_N^T(B)), \end{aligned}$$

where

$$x_B(B) = B^{-1}b,\ x_N(B) \equiv 0.$$

We consider

- feasible basis for which $x_B(B) \geq 0$,
- optimal basis corresponding to an optimal solution,
- basic solution(s).

The simplex algorithm can be represented by the simplex table:

			x^T
			c^T
c_B	$x_B(B)$	$B^{-1}b$	$B^{-1}A$
		$c_B^T B^{-1}b$	$c_B^T B^{-1}A - c^T$

In the table, we can identify

- feasibility condition:
$$B^{-1}b \geq 0,$$
- optimality condition:
$$c_B^T B^{-1}A - c^T \leq 0.$$

Simplex algorithm – a step: If the optimality condition is not fulfilled:

- Denote the criterion row by
$$\delta^T = c_B^T B^{-1}A - c^T.$$
- Find a positive element $\delta_i > 0$ and denote the corresponding column by
$$\rho = B^{-1}A_{\bullet,i},$$
where $A_{\bullet,i}$ is the $i-$th column of A.
- Minimize the ratios
$$\hat{u} = \arg\min \left\{ \frac{x_u(B)}{\rho_u} \, : \; \rho_u > 0, \; u \in B \right\}.$$
- Substitute $x_{\hat{u}}$ by x_i in the basic variables, i.e. $\hat{B} = B \setminus \{\hat{u}\} \cup \{i\}$.

Denote by $\hat{B}$ the new basis. Simplex algorithm is moving from one basic solution to another one. We can identify the direction and the step length in our steps. The direction is

$$\begin{aligned} \Delta_u &= -\rho_u, \; u \in B, \\ \Delta_i &= 1, \\ \Delta_j &= 0, \; j \notin B \cup \{i\}. \end{aligned}$$

If $\rho \leq 0$ ($\hat{u} = \emptyset$), then the problem is unbounded ($c^T x \to -\infty$). Otherwise, we can move from the current basic solution to another one as

$$x(\hat{B}) = x(B) + t\Delta,$$

where $0 \leq t = \frac{x_{\hat{u}}(B)}{\rho_{\hat{u}}}$. We should prove that the new solution is a feasible basic solution and that the objective value decreases

First, we show that the new solution is feasible:

$$\begin{aligned} x(\hat{B}) &\geq 0, \\ Ax(\hat{B}) &= Ax(B) + tA\Delta \\ &= Ax(B) - tB\rho + tA_{\bullet,i} \\ &= b - tBB^{-1}A_{\bullet,i} + tA_{\bullet,i} = b. \end{aligned}$$

Now, we obtain that the objective value decreases

$$\begin{aligned} c^T x(\hat{B}) &= c^T x(B) + tc^T\Delta \\ &= c^T x(B) - tc_B^T\rho + tc_i \\ &= c^T x(B) - t(c_B^T B^{-1}A_{\bullet,i} - c_i) \\ &= c^T x(B) - t\delta_i, \end{aligned}$$

where $\delta_i > 0$ is the element of the criterion row.

Finally note

- if $\rho \leq 0$, then $x(\hat{B})$ is feasible for all $t \geq 0$ and the objective value decreases in the direction Δ,
- otherwise the step length t is bounded by $\frac{x_{\hat{u}}(B)}{\rho_{\hat{u}}}$. In this case, the new basis $\hat{B}$ is regular, because we interchange one unit vector by another one using the column i with $\rho_{\hat{u}} > 0$ element (on the right position).

Realize that not all steps of the simplex algorithm are uniquely determined. Pivot rules are used for selecting the entering variable if there are several possibilities:

- **Largest coefficient** in the objective function
- **Largest decrease** of the objective function
- **Steepest edge** – choose an improving variable whose entering into the basis moves the current basic feasible solution in a direction closest to the direction of the vector c

  $$\max \frac{c^T(x_{new} - x_{old})}{\|x_{new} - x_{old}\|}.$$

  Computationally the most successful.
- **Blands's rule** – choose the improving variable with the smallest index, and if there are several possibilities for the leaving variable, also take the one with the smallest index (prevents cycling)

For details see Matoušek and Gärtner (2007).

Example 1.1. *Consider two table which differ by an objective coefficient.*

			3	-1	0	0
			x_1	x_2	x_3	x_4
0	x_3	2	-2	1	1	0
0	x_4	1	1	-2	0	1
		0	-3	1	0	0
-1	x_2	2	-2	1	1	0
4	x_4	5	-3	0	2	1
		-2	-1	0	-1	0

We moved in direction $\Delta^T = (0, 1, -1, 2)$*, i.e.*

$$(0, 2, 0, 5) = (0, 0, 2, 1) + t \cdot (0, 1, -1, 2),$$

where $t = 2$.

			-2	-1	0	0
			x_1	x_2	x_3	x_4
0	x_3	2	-2	1	1	0
0	x_4	1	1	-2	0	1
		0	2	1	0	0
-1	x_2	2	-2	1	1	0
0	x_4	5	-3	0	2	1
		-2	4	0	-1	0

This problem is unbounded in direction $\Delta^T = (1, 2, 0, 3)$.

1.2 Duality in linear programming

We review basis of duality in linear programming. Consider primal problem **Primal problem**

$$\begin{aligned} \text{(P)}\ \ \min\ & c^T x \\ \text{s.t. } & Ax \geq b, \\ & x \geq 0. \end{aligned}$$

and corresponding **dual problem**

$$\begin{aligned} \text{(D)}\ \ \max\ & b^T y \\ \text{s.t. } & A^T y \leq c, \\ & y \geq 0. \end{aligned}$$

Denote the sets of feasible solutions:

$$\begin{aligned} M &= \{x \in \mathbb{R}^n :\ \ Ax \geq b,\ x \geq 0\}, \\ N &= \{y \in \mathbb{R}^m :\ \ A^T y \leq c,\ y \geq 0\}. \end{aligned}$$

Weak duality theorem says

$$b^T y \leq c^T x,\ \forall x \in M, \forall y \in N.$$

Equality holds if and only if complementarity slackness conditions are fulfilled:

$$\begin{aligned} y^T(Ax - b) &= 0, \\ x^T(A^T y - c) &= 0. \end{aligned}$$

- Duality theorem: If $M \neq \emptyset$ and $N \neq \emptyset$, than the problems (P), (D) have optimal solutions.
- Strong duality theorem: The problem (P) has an optimal solution if and only if the dual problem (D) has an optimal solution. If one problem has an optimal solution, than the optimal values are equal.

1.2.1 Production planning

Optimize the production of the following products V_1, V_2, V_3 made from materials M_1, M_2.

	V_1	V_2	V_3	Constraints
M_1	1	0	2	54 kg
M_2	2	3	1	30 kg
Gain (\$/kg)	10	15	10	

Primal problem can be formulated as

$$
(P) \quad \begin{array}{rrcrcrcl}
\max & 10x_1 & + & 15x_2 & + & 10x_3 & & \\
\text{s.t.} & x_1 & & & + & 2x_3 & \leq & 54, \\
 & 2x_1 & + & 3x_2 & + & x_3 & \leq & 30, \\
 & x_1 & & & & & \geq & 0, \\
 & & & x_2 & & & \geq & 0, \\
 & & & & & x_3 & \geq & 0.
\end{array}
$$

The corresponding dual problem is

$$
(D) \quad \begin{array}{rrcrcl}
\min & 54y_1 & + & 30y_2 & & \\
\text{s.t.} & y_1 & + & 2y_2 & \geq & 10, \\
 & & & 3y_2 & \geq & 15, \\
 & 2y_1 & + & y_2 & \geq & 10, \\
 & y_1 & & & \geq & 0, \\
 & & & y_2 & \geq & 0.
\end{array}
$$

We can easily obtained an optimal solution of (D) $\hat{y} = \left(\frac{5}{2}, 5\right)^T$. Using the complementarity slackness conditions we obtain the optimal solution of the primal problem $\hat{x} = (0, 1, 27)^T$. The optimal values (gains) of (P) and (D) are 285.

- Both (P) constraints are fulfilled with equality, thus there in no material left.
- Dual variables are called **shadow prices** and represent the prices of sources (materials).
- **Sensitivity**: If we increase (P) r.h.s. by one, then the objective value increases by the shadow price.
- The first constraint of (D) is fulfilled with strict inequality with the difference 2.5 \$, called **reduced prices**, and the first product is not produced. The producer should increase the gain from V_1 by this amount to become profitable.

1.2.2 Transportation problem

We consider the following notation:

- x_{ij} – decision variable: amount transported from i to j,
- c_{ij} – costs for transported unit,
- a_i – capacity,
- b_j – demand.

We assume that $\sum_{i=1}^n a_i \geq \sum_{j=1}^m b_j$, i.e. the demand can be satisfied by the available capacity. (Sometimes $a_i, b_j \in \mathbb{N}$.)

Primal problem

$$\begin{aligned}
\min\ & \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij} \\
\text{s.t.}\ & \sum_{j=1}^m x_{ij} \leq a_i,\ i = 1, \dots, n, \\
& \sum_{i=1}^n x_{ij} \geq b_j,\ j = 1, \dots, m, \\
& x_{ij} \geq 0.
\end{aligned}$$

Dual problem

$$\begin{aligned}
\max\ & \sum_{i=1}^n a_i u_i + \sum_{j=1}^m b_j v_j \\
\text{s.t.}\ & u_i + v_j \leq c_{ij}, \\
& u_i \leq 0, \\
& v_j \geq 0.
\end{aligned}$$

Interpretation: $-u_i$ price for buying a unit of goods at i, v_j price for selling at j.

Competition between the transportation company (which minimizes the transportation costs) and an "agent" (who maximizes the earnings):

$$\sum_{i=1}^n a_i u_i + \sum_{j=1}^m b_j v_j \leq \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij}$$

Linear programming duality

Apply KKT optimality conditions to primal LP ... we will see relations with NLP duality.

1.3 Dual simplex algorithm

Linear programming duality **Primal problem** (standard form)

$$\begin{aligned}
\min\ & c^T x \\
\text{s.t.}\ & Ax = b, \\
& x \geq 0.
\end{aligned}$$

and corresponding **dual problem**

$$\begin{aligned} &\max b^T y \\ &\text{ s.t. } A^T y \le c, \\ &\quad y \in \mathbb{R}^m. \end{aligned}$$

Dual simplex algorithm works with

- **dual feasible basis** B and
- **basic dual solution** $y(B)$,

where

$$\begin{aligned} B^T y(B) &= c_B, \\ N^T y(B) &\le c_N. \end{aligned}$$

Primal feasibility $B^{-1}b \ge 0$ **is violated** until reaching the optimal solution.
Primal optimality condition is always fulfilled:

$$c_B^T B^{-1} A - c^T \le 0.$$

Using $A = (B|N)$, $c^T = (c_B^T, c_N^T)$, we have

$$\begin{aligned} c_B^T B^{-1} B - c_B^T &= 0, \\ c_B^T B^{-1} N - c_N^T &\le 0, \end{aligned}$$

Setting $\hat{y} = (B^{-1})^T c_B$

$$\begin{aligned} B^T \hat{y} &= c_B^T, \\ N^T \hat{y} &\le c_N^T. \end{aligned}$$

Thus, $\hat{y}$ is a basic dual solution.
Dual simplex algorithm – a step ... uses the same simplex table.

- Find index $u \in B$ such that $x_u(B) < 0$ and denote the corresponding row by

$$\tau^T = (B^{-1}A)_{u,\bullet}.$$

- Denote the criterion row by

$$\delta^T = c_B^T B^{-1} A - c^T \le 0.$$

- Minimize the ratios

$$\hat{\imath} = \arg\min \left\{ \frac{\delta_i}{\tau_i} : \ \tau_i < 0 \right\}.$$

- Substitute x_u by $x_{\hat{\imath}}$ in the basic variables, i.e. $\hat{B} = B \setminus \{u\} \cup \{\hat{\imath}\}$. We move to another **basic dual solution**.

Example – dual simplex algorithm

The problem is **dual nondegenerate** if for all dual feasible basis B it holds

$$\begin{aligned}(A^T y(B) - c)_j &= 0,\ j \in B,\\ (A^T y(B) - c)_j &< 0,\ j \notin B.\end{aligned}$$

If the problem is dual nondegenerate, then the dual simplex algorithm ends after finitely many steps.

$$\begin{aligned}\min 4x_1 + 5x_2 & & \\ x_1 + 4x_2 &\geq& 5,\\ 3x_1 + 2x_2 &\geq& 7,\\ x_1, x_2 &\geq& 0.\end{aligned}$$

Dual problem

$$\begin{aligned}\max\ & -5y_1 - 7y_2\\ \text{s.t.}\ & -y_1 - 3y_2 \leq 4\\ & -4y_1 - 2y_2 \leq 5\\ & y_1 \leq 0\\ & y_2 \leq 0.\end{aligned}$$

			4	5	0	0
			x_1	x_2	x_3	x_4
0	x_3	-5	-1	-4	1	0
0	x_4	-7	-3	-2	0	1
		0	-4	-5	0	0
0	x_3	-8/3	0	-10/3	1	-1/3
4	x_1	7/3	1	2/3	0	-1/3
		28/3	0	-7/3	0	-4/3
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

The last solution is primal and dual feasible, thus optimal.

A general step in the dual simplex algorithm

$$y(\hat{B}) = y(B) - t(B^{-1})^T_{\bullet,u}$$

i.e.

$$(0, -4/3) = (0, 0) - 4/3(0, 1),$$

which can be seen in the criterion row in the columns corresponding to the initial basis. Dual constraints 1 and 3 are then active.

1.4 Software tools for LP

Software tools for solving linear programming problems include

- Matlab
- Mathematica
- GAMS
- Cplex studio
- AIMMS
- ...
- R
- MS Excel
- ...

2 An introduction to Benders decomposition

Benders decomposition is a very useful principle in optimization which main idea is to decompose the original problem into two or more problems These problems are then solved iteratively where the solutions of one problem are used in the second one and vice versa. The Benders decomposition can be helpful for solving the problems from the following classes:

- linear programming,
- mixed-integer (non)linear programming,
- two-stage stochastic programming (called L-shaped algorithm),
- multistage stochastic programming (Nested Benders decomposition).

We will show how the Benders decomposition works on two-stage linear programming problems of the form

$$\begin{aligned} \min\ & c^T x + q^T y \\ \text{s.t. } & Ax = b, \\ & Tx + Wy = h, \\ & x \geq 0, \\ & y \geq 0. \end{aligned} \tag{1}$$

We impose the following assumption on the problem:
ASS. $\mathcal{B}_1 := \{x : \ Ax = b, x \geq 0\}$ is bounded and the problem has an optimal solution.

Benders decomposition relies on decomposing the problem into two (or more) parts which can be solved easily than the full problem. In our case, both problems will be linear. We define the recourse function (second-stage value function, slave problem)

$$f(x) \ = \ \min\{q^T y : \ Wy = h - Tx, \ y \geq 0\} \tag{2}$$

If for some x is $\{y : \ Wy = h - Tx, \ y \geq 0\} = \emptyset$, then we set $f(x) = \infty$. We can show that the recourse function is piecewise linear, convex, and bounded below .

Proof (outline): We will show the properties in several steps

- $f(x)$ is bounded below and piecewise linear (affine): There are finitely many optimal basis B chosen from W such that

  $$f(x) = q_B^T B^{-1}(h - Tx),$$

  where feasibility $B^{-1}(h - Tx) \geq 0$ is fulfilled for $x \in \mathcal{B}_1$. Optimality condition $q_B^T B^{-1} W - q \leq 0$ does not depend on x.

- $f(x)$ is convex: let $x_1, x_2 \in \mathcal{B}_1$ and y_1, y_2 be such that $f(x_1) = q^T y_1$ and $f(x_2) = q^T y_2$. For arbitrary $\lambda \in (0,1)$ and $x = \lambda x_1 + (1-\lambda)x_2$ we have

  $$\lambda y_1 + (1-\lambda) y_2 \in \{y : \ Wy = h - Tx, \ y \geq 0\},$$

  i.e. the convex combination of y's is feasible. Thus we have

  $$\begin{aligned} f(x) \ &= \ \min\{q^T y : \ Wy = h - Tx, \ y \geq 0\} && (3) \\ &\leq \ q^T(\lambda y_1 + (1-\lambda) y_2) = \lambda f(x_1) + (1-\lambda) f(x_2). && (4) \end{aligned}$$

We have an equivalent NLP problem to the original one (1)

$$\begin{aligned} \min\ & c^T x + f(x) \\ \text{s.t. } & Ax = b, \\ & x \geq 0. \end{aligned} \tag{5}$$

We can solve the master problem (first-stage problem) in an equivalent form

$$\begin{aligned} \min\ & c^T x + \theta \\ \text{s.t. } & Ax = b, \\ & f(x) \leq \theta, \\ & x \geq 0. \end{aligned} \tag{6}$$

We would like to approximate $f(x)$ from below by adding cuts (linear inequalities) leading back to a linear programming problem.

First, we will show how to construct the feasibility cuts. Solve the slave problem for a given $\hat{x}$

$$\begin{aligned} f(\hat{x}) &= \min\{q^T y :\ Wy = h - T\hat{x},\ y \geq 0\} && (7) \\ &= \max\{(h - T\hat{x})^T u :\ W^T u \leq q\}. && (8) \end{aligned}$$

If the dual problem is unbounded (primal is infeasible according to the LP duality), then there exists a growth direction $\tilde{u}$ such that $W^T\tilde{u} \leq 0$ and $(h - T\hat{x})^T\tilde{u} > 0$. For any feasible x there exists some $y \geq 0$ such that $Wy = h - Tx$. If we multiply it by $\tilde{u}$, we obtain

$$\tilde{u}^T(h - T\hat{x}) = \tilde{u}^T Wy \leq 0,$$

which has to hold for any feasible x, but is violated by $\hat{x}$. Thus by

$$\tilde{u}^T(h - Tx) \leq 0$$

the infeasible $\hat{x}$ is cut off. The last inequality is the feasibility cut.

If the dual problem is not unbounded, then we can derive the optimality cut. We know that there is an optimal solution $\hat{u}$ of the dual problem such that

$$f(\hat{x}) = (h - T\hat{x})^T\hat{u}.$$

For arbitrary x we have

$$\begin{aligned} f(x) &= \sup_u\{(h - Tx)^T u :\ W^T u \leq q\}, && (9) \\ &\geq (h - Tx)^T\hat{u}, && (10) \end{aligned}$$

because $\hat{u}$ is feasible for arbitrary x. From inequality $f(x) \leq \theta$ we have the optimality cut

$$\hat{u}^T(h - Tx) \leq \theta.$$

If this cut is fulfilled for actual $(\hat{x}, \hat{\theta})$, then we STOP the iterations, $\hat{x}$ is an optimal solution.

The cuts are added to the master problem (one in each iteration). We solve the master problem with cuts

$$\begin{aligned} \min\ & c^T x + \theta \\ \text{s.t. } & Ax = b, \\ & \tilde{u}_l^T (h - Tx) \le 0,\ l = 1, \dots, L, \\ & \tilde{u}_k^T (h - Tx) \le \theta,\ k = 1, \dots, K, \\ & x \ge 0. \end{aligned} \tag{11}$$

The algorithm is summarized in the following steps:

0. INIC: Set $\theta = -\infty$, $L = 0$, $K = 0$.
1. Solve the master problem to obtain $(\hat{x}, \hat{\theta})$.
2. For $\hat{x}$, solve the dual of the second-stage (recourse) problem to obtain
   - a direction of unbounded decrease (feasibility cut), $L = L + 1$,
   - or an optimal solution (optimality cut), $K = K + 1$.
3. STOP, if the current solution $(\hat{x}, \hat{\theta})$ fulfills the optimality cuts. Otherwise GO TO Step 1.

Convergence of the algorithm can be proven as follows. There are finitely many extreme directions that can generate the feasibility cuts and finitely many (dual) feasible basis which can produce the optimality cuts.

Let (x^*, θ^*) be an optimal solution of the reformulated original problem.

1. The feasibility set of the master problem (6) is always contained in the feasibility set of the master problem with cuts (11) (no feasible solutions are cut).
2. The optimal solution $(\hat{x}, \hat{\theta})$ obtained by the algorithm is feasible for the master problem (6), because
$$\hat{\theta} \ge (h - T\hat{x})^T \hat{u} = f(\hat{x}).$$

Thus, from 1. and 2. we obtain

$$c^T x^* + \theta^* \ge c^T \hat{x} + \hat{\theta} \ge c^T x^* + \theta^*.$$

For more details see Kall and Mayer (2005), Proposition 2.19.

Benders optimality cuts Kall and Mayer (2005)

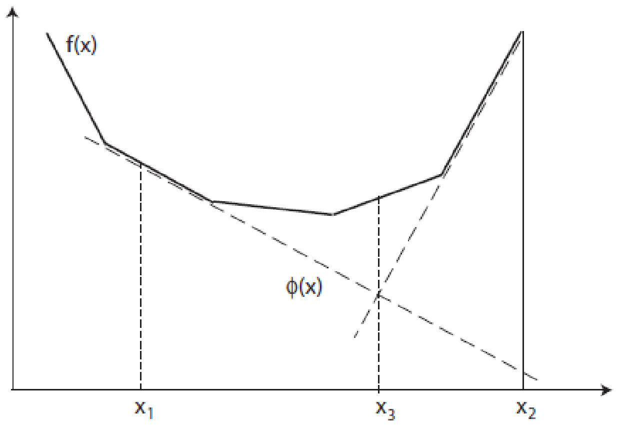
2.1 Example

We will demonstrate the steps of the Benders decomposition on the following LP problem:

$$\begin{aligned} \min\ & 2x + 2y_1 + 3y_2 \\ \text{s.t. } & x + y_1 + 2y_2 = 3, \\ & 3x + 2y_1 - y_2 = 4, \\ & x,\ y_1,\ y_2 \ge 0. \end{aligned} \tag{12}$$

ht

Figure 1: Benders cuts approximating the value function



Recourse function can be defined as:

$$
\begin{aligned}
f(x) = \min\ & 2y_1 + 3y_2 \\
\text{s.t. } & y_1 + 2y_2 = 3 - x, \\
& 2y_1 - y_2 = 4 - 3x, \\
& y_1,\ y_2 \geq 0,
\end{aligned} \tag{13}
$$

i.e. it is a real function of one variable. Now we will proceed in iterations.

Iteration 1: Set $\theta = -\infty$ and solve master problem

$$
\min_x 2x \text{ s.t. } x \geq 0. \tag{14}
$$

We obtained the optimal solution $\hat{x} = 0$. Now, solve the dual of the slave problem for $\hat{x} = 0$:

$$
\begin{aligned}
\max_u\ & (3 - x)u_1 + (4 - 3x)u_2 \\
\text{s.t. } & u_1 + 2u_2 \leq 2, \\
& 2u_1 - u_2 \leq 3.
\end{aligned} \tag{15}
$$

Optimal solution is $\hat{u} = (8/5, 1/5)$ with optimal value 28/5, thus no feasibility cut is necessary. We can construct an optimality cut

$$
(3 - x)8/5 + (4 - 3x)1/5 = 28/5 - 11/5x \leq \theta.
$$

Iteration 2: Add the optimality cut and solve

$$
\begin{aligned}
\min_{x,\theta}\ & 2x + \theta \\
\text{s.t. } & 28/5 - 11/5x \leq \theta, \\
& x \geq 0.
\end{aligned} \tag{16}
$$

Optimal solution $(\hat{x}, \hat{\theta}) = (2.5455, 0)$ with optimal value 5.0909. Solve the dual problem for $\hat{x} = 2.5455$:

$$
\begin{aligned}
\max_u\ & (3 - x)u_1 + (4 - 3x)u_2 \\
\text{s.t. } & u_1 + 2u_2 \leq 2, \\
& 2u_1 - u_2 \leq 3.
\end{aligned} \tag{17}
$$

Optimal solution is $\hat{u} = (1.5, 0)$ with optimal value 0.6818, thus no feasibility cut is necessary. We can construct an optimality cut

$$(3 - x)1.5 + (4 - 3x)0 = 4.5 - 1.5x \le \theta.$$

Iteration 3: Add the optimality cut and solve

$$\begin{aligned} \min_{x,\theta}\ & 2x + \theta \\ \text{s.t. } & 28/5 - 11/5x \le \theta, \\ & 4.5 - 1.5x \le \theta, \\ & x \ge 0. \end{aligned} \tag{18}$$

The iterations continue.

2.2 Applications

In this part, we will outline the applications of the Benders decomposition to more complicated problems.

2.2.1 Two-stage stochastic programming problems

Consider realization (q_s, h_s, T_s) of the random coefficients with probabilities $0 < p_s < 1$, $\sum_s p_s = 1$. Two-stage stochastic programming problem can be formulated as follows

$$\begin{aligned} \min\ & c^T x + \sum_{s=1}^{S} p_s q_s^T y_s \\ \text{s.t. } & Ax = b, \\ & \begin{array}{ccccccc} Wy_1 & & & & +T_1 x & = & h_1, \\ & Wy_2 & & & +T_2 x & = & h_2, \\ & & \ddots & & \vdots & \vdots & \vdots \\ & & & Wy_S & +T_S x & = & h_S, \end{array} \\ & x \ge 0,\ y_s \ge 0,\ s = 1, \dots, S. \end{aligned} \tag{19}$$

A modification of the Benders decomposition called “L-shaped algorithm”, considers one master and S second-stage problems wehere we can apply the dual approach to each of them.

2.2.2 Minimization of Conditional Value at Risk

If the distribution of asset returns R_i is discrete with realizations r_{is} and probabilities $p_s = 1/S$, then we can use linear programming reformulation of the minimization formula for the Conditional Value at Risk:

$$\begin{aligned}\min_{\xi, x_i}\ & \xi + \frac{1}{(1-\alpha)S}\sum_{s=1}^{S}[-\sum_{i=1}^{n} x_i r_{is} - \xi]_+,\\ \text{s.t. } & \sum_{i=1}^{n} x_i \overline{R}_i \geq r_0,\\ & \sum_{i=1}^{n} x_i = 1,\ x_i \geq 0,\end{aligned}$$

where $\overline{R}_i = 1/S\sum_{s=1}^{S} r_{is}$, $[\cdot]_+ = \max\{\cdot, 0\}$.

We consider the master problem

$$\begin{aligned}\min_{\xi, x_i}\ & \xi + \frac{1}{(1-\alpha)S}\sum_{s=1}^{S} f_s(x,\xi),\\ \text{s.t. } & \sum_{i=1}^{n} x_i \overline{R}_i \geq r_0,\ \sum_{i=1}^{n} x_i = 1,\ x_i \geq 0,\end{aligned}$$

and S second-stage problems

$$\begin{aligned}f_s(x,\xi) = \min_{y}\ & y,\\ \text{s.t. } & y \geq -\sum_{i=1}^{n} x_i r_{is} - \xi,\\ & y \geq 0.\end{aligned}$$

If we construct the dual problems, we observe that they can be solved very quickly...

3 Integer Linear Programming

In this chapter, we will focus on integer linear programming problems. We will discuss several real life problems, general properties and algorithms.

3.1 Motivation and applications

Consider a knapsack problem with the item values $a_1 = 4$, $a_2 = 6$, $a_3 = 7$, the costs $c_1 = 4$, $c_2 = 5$, $c_3 = 11$, and the knapsack capacity (budget) $b = 10$:

$$\begin{aligned} \max \ & \sum_{i=1}^{3} c_i x_i \\ \text{s.t.} \ & \sum_{i=1}^{3} a_i x_i \leq 10, \\ & x_i \in \{0, 1\}. \end{aligned}$$

Think about the following modifications

- equality $=$ instead of $\leq$ in the constraint,
- relaxation $0 \leq x_i \leq 1$ and rounding of the LP optimal solution instead of $x_i \in \{0, 1\}$,
- heuristic based on sorting the items using the ratio c_i/a_i.

Why is integrality of decision variables so important? There are many real-life (mixed-)integer programming problems (not always linear) which cannot be formulated without integer variables, for example

- Portfolio optimization – integer number of assets, fixed transaction costs,
- Scheduling – integer (binary) decision variables to assign a job to a machine,
- Vehicle Routing Problems (VRP) – binary decision variables which identify a successor of a node on the route.

In general, integer (binary) decision variables can be used for modelling of "logical relations", e.g.

- at least two constraints from three are fulfilled,
- if we buy this asset than the fixed transaction costs increase.

Below, we will formulate several (mixed-)integer problems which represent general classes of real-life problems.

3.1.1 Facility Location Problem

The basic facility location problems aims at minimizing the building costs for the facilities and at the same time minimization of the transportation costs to customers.

The following standard notation is used:

- i warehouses (facilities, branches),
- j customers,
- x_{ij} – sent (delivered, served) quantity,
- y_i – a warehouse is built,
- c_{ij} – unit supplying costs,
- f_i – fixed costs of building the warehouse,
- K_i – warehouse capacity,
- D_j – demand.

The problem is

$$\begin{aligned} \min_{x_{ij}, y_i} \; & \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} + \sum_{i} f_i y_i \\ \text{s.t.} \; & \sum_{j=1}^{m} x_{ij} \le K_i y_i, \;\; i = 1, \dots, n, \\ & \sum_{i=1}^{n} x_{ij} = D_j, \;\; j = 1, \dots, m, \\ & x_{ij} \ge 0, \;\; y_i \in \{0, 1\}. \end{aligned}$$

3.1.2 Scheduling to Minimize the Makespan

Scheduling deals with assigning of jobs to available machines to optimize various criteria under specific constraints. The basic problem formulated in this part aims at minimizing the makespan, i.e. the latest working time of machines. Later, we will discuss another class of scheduling problems called fixed interval scheduling.

We employ the notation:

- i machines,
- j jobs,
- y – machine makespan,
- x_{ij} – assignment variable of job j to machine i,
- t_{ij} – time necessary to process job j on machine i.

The problem is

$$\begin{aligned}
\min_{x_{ij},y}\ & y \\
\text{s.t.}\ & \sum_{i=1}^{m} x_{ij} = 1,\ j = 1,\dots,n, \\
& \sum_{j=1}^{n} t_{ij}x_{ij} \le y,\ i = 1,\dots,m, \\
& x_{ij} \in \{0,1\},\ y \ge 0.
\end{aligned} \tag{20}$$

3.1.3 Lot Sizing Problem

The main goal of the lot sizing problems (LSP) is to balance the production and storing (warehousing) costs. We will consider the basic variant where only one item is produced and the goal is to minimize the costs over a planning horizon. We distinguish two problems – the uncapacitated and capacitated where a maximal production capacity is each period is given.

We employ the notation:

- t – time period,
- x_t – production at period t,
- y_t – on/off decision at period t,
- s_t – inventory at the end of period t ($s_0 \ge 0$ fixed),
- D_t – (predicted) *expected* demand at period t,
- p_t – unit production costs at period t,
- f_t – setup costs at period t,
- h_t – inventory costs at period t,
- M – a large constant.
- C_t – production capacity at period t.

Uncapacitated single item LSP is

$$\begin{aligned}
\min_{x_t,y_t,s_t}\ & \sum_{t=1}^{T} (p_t x_t + f_t y_t + h_t s_t) \\
\text{s.t.}\ & s_{t-1} + x_t - D_t = s_t,\ t = 1,\dots,T, \\
& x_t \le M y_t, \\
& x_t, s_t \ge 0,\ y_t \in \{0,1\}.
\end{aligned} \tag{21}$$

A traditional assumption called "Wagner-Whitin costs" is that the production at one period together with the storage costs are greater or equal to the production costs at the following period, i.e.

$$p_{t+1} \le p_t + h_t.$$

Capacitated single item LSP is

$$\begin{aligned}\min_{x_t,y_t,s_t}\ & \sum_{t=1}^{T}(p_t x_t + f_t y_t + h_t s_t)\\ \text{s.t. } & s_{t-1} + x_t - D_t = s_t,\ t = 1,\dots,T,\\ & x_t \le C_t y_t,\\ & x_t, s_t \ge 0,\ y_t \in \{0,1\}.\end{aligned} \tag{22}$$

3.1.4 Unit Commitment Problem

The unit commitment problem (UCP) is usually used in the energy sector where the "units" are the power generators and the goal is optimize the energy production over several periods to satisfy the demand taking into account various production costs. Maybe the main difference between UCP and LSP is that in this case the storage is not possible and the whole production has to be "consumed".

We employ the notation:

- $i = 1,\dots,n$ units (power plants),
- $t = 1,\dots,T$ periods,
- y_{it} – on/off decision for unit i at period t,
- x_{it} – production level for unit i at period t,
- D_t – (predicted) *expected* demand at period t,
- p_i^{min}, p_i^{max} – minimal/maximal production capacity of unit i,
- c_{it} – variable production costs,
- f_{it} – (fixed) start-up costs.

The unit commitment problem can be formulated as

$$\begin{aligned}\min_{x_{it},y_{it}}\ & \sum_{i=1}^{n}\sum_{t=1}^{T}(c_{it}x_{it} + f_{it}y_{it})\\ \text{s.t. } & \sum_{i=1}^{n} x_{it} \ge D_t,\ t = 1,\dots,T,\\ & p_i^{min} y_{it} \le x_{it} \le p_i^{max} y_{it},\\ & x_{it} \ge 0,\ y_{it} \in \{0,1\}.\end{aligned} \tag{23}$$

3.2 Formulation and properties

General integer linear programming (ILP) can be formulated as

$$\min c^T x \tag{24}$$

$$Ax \ \ge\ b, \tag{25}$$

$$x \ \in\ \mathbb{Z}_+^n. \tag{26}$$

Assumption: all coefficients are integer (rational before multiplying by a proper constant). Set of feasible solution and its relaxation are denoted by

$$\begin{aligned} S &= \{x \in \mathbb{Z}^n_+ :\ Ax \geq b\}, & (27)\\ P &= \{x \in \mathbb{R}^n_+ :\ Ax \geq b\} & (28)\end{aligned}$$

Obviously $S \subseteq P$. Not so trivial that $S \subseteq \operatorname{conv}(S) \subseteq P$.

ILP with irrational data need not to fulfill the trichotomy known from linear programming. We will propose an example where the problem is feasible and the objective function is bounded but there is no optimal solution, cf. Škoda (2010):

$$\begin{aligned} &\max \sqrt{2}x - y\\ &\text{s.t. } \sqrt{2}x - y \leq 0,\\ &\quad x \geq 1,\\ &\quad x, y \in \mathbb{N}. \end{aligned} \tag{29}$$

For any feasible solution with the objective value $z = \sqrt{2}x^* - \lceil\sqrt{2}x^*\rceil$ we can construct a solution with a higher objective value as follows. Let $z = \sqrt{2}x^* - \lceil\sqrt{2}x^*\rceil$ be the optimal solution. Since $-1 < z < 0$, we can find $k \in \mathbb{N}$ such that $kz < -1$ and $(k-1)z > -1$. By setting $\epsilon = -1 - kz$ we get that $-1 < z < -\epsilon = 1 + kz < 0$. Then

$$\begin{aligned} &\sqrt{2}kx^* - \left\lceil\sqrt{2}kx^*\right\rceil\\ &= kz + k\left\lceil\sqrt{2}x^*\right\rceil - \left\lceil\sqrt{2}kx^*\right\rceil\\ &= -1 - \epsilon + k\left\lceil\sqrt{2}x^*\right\rceil - \left\lceil\sqrt{2}kx^*\right\rceil\\ &= k\left\lceil\sqrt{2}x^*\right\rceil - 1 - \epsilon - \left\lceil\left\lceil\sqrt{2}kx^*\right\rceil - 1 - \epsilon\right\rceil\\ &= -\epsilon > z. \end{aligned} \tag{30}$$

Note that $k\lceil\sqrt{2}x^*\rceil - 1$ is integral. Thus, we have obtained a solution with a higher objective value which is a contradiction.

Example 3.1. *Consider the integer set given by*

$$\begin{aligned} x_1 + 4x_2 &\geq 5, & (31)\\ 3x_1 + 2x_2 &\geq 7, & (32)\\ x_1, x_2 &\in \mathbb{Z}^n_+. & (33)\end{aligned}$$

Figure 2 contains the integer set, its relaxations and the convex envelope of the integer set.

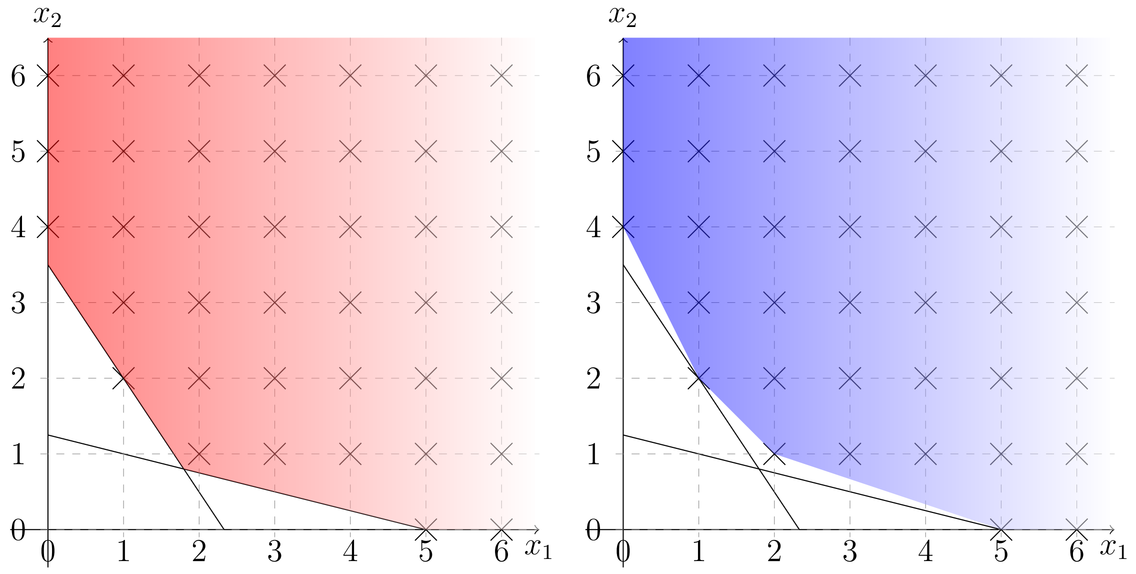
We claim without proof, see Wolsey (1998), that the integer linear programming problem

$$\min c^T x :\ x \in S. \tag{34}$$

is equivalent to

$$\min c^T x :\ x \in \operatorname{conv}(S), \tag{35}$$

Figure 2: Set of feasible solutions S (crosses), its relaxation P (red) and convex envelope conv(S) (blue)



which is a linear programming problem, because the convex envelope can be described by linear inequalities. However, explicit description of conv(S) is very difficult to construct – many constraints ("strong cuts") are necessary. There are some exceptions where the description is available. Under the LP-relaxation we understand the problem with the relax set

$$\min c^T x : \ x \in P. \tag{36}$$

Usually, the algorithms are starting with solving the LP-relaxation.

Note that often both integer and continuous decision variables appear in the problem leading to a Mixed-integer linear programming (MILP)

$$\begin{aligned} \min\ & c^T x + d^T y \\ \text{s.t. }\ & Ax + By \geq b \\ & x \in \mathbb{Z}^n_+,\ y \in \mathbb{R}^{n'}_+, \end{aligned}$$

which we do not consider in this introduction.

3.3 Algorithms

We will introduce two basic approaches:

- *Cutting Plane Method*
- *Branch-and-Bound*

There are methods which combine the previous algorithms, e.g., Branch-and-Cut which adds cuts to reduce the problems solved inside the Branch-and-Bound.

3.3.1 Cutting plane method

In this part, cutting plane method with a special on Gomory cuts is discussed. The algorithm can be outlined as follows:

1. Solve LP-relaxation using (primal or dual) SIMPLEX algorithm.
   - If the solution is integral – END, we have found an optimal solution,
   - otherwise continue with the next step.
2. Add a *Gomory cut* (...) and solve the resulting problem using DUAL SIMPLEX alg.

Example 3.2.

$$\min 4x_1 + 5x_2 \tag{37}$$
$$x_1 + 4x_2 \geq 5, \tag{38}$$
$$3x_1 + 2x_2 \geq 7, \tag{39}$$
$$x_1, x_2 \in \mathbb{Z}^n_+. \tag{40}$$

Use the dual simplex algorithm for the LP-relaxation. After two iterations we obtain

			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*

Now, we will show how to derive the Gomory cuts. There is a row in simplex table, which corresponds to a *non-integral solution* x_i in the form:

$$x_i + \sum_{j \in N} w_{ij} x_j = d_i, \tag{41}$$

where N denotes the set of non-basic variables; d_i is non-integral. We denote

$$w_{ij} = \lfloor w_{ij} \rfloor + f_{ij}, \tag{42}$$
$$d_i = \lfloor d_i \rfloor + f_i, \tag{43}$$

i.e. $0 \leq f_{ij}, f_i < 1$. The Gomory cut is then defined as

$$\sum_{j \in N} f_{ij} x_j \geq f_i. \tag{44}$$

Before adding to the simplex table, it is transformed into equality using a slack variable $s \geq 0$, i.e. we have

$$-\sum_{j \in N} f_{ij} x_j + s = -f_i.$$

In general, the cuts (including Gomory ones) has to have these two properties:

- Property 1: Current (non-integral) solution becomes infeasible (it is cut).
- Property 2: No feasible integral solution becomes infeasible (it is not cut).

Property 1 for the Gomory cuts: We express the constraints in the form

$$x_i + \sum_{j\in N}(\lfloor w_{ij}\rfloor + f_{ij})x_j = \lfloor d_i\rfloor + f_i, \tag{45}$$

$$x_i + \sum_{j\in N}\lfloor w_{ij}\rfloor x_j - \lfloor d_i\rfloor = f_i - \sum_{j\in N} f_{ij}x_j. \tag{46}$$

Current solution $x_j^* = 0$ pro $j \in N$ a $x_i^* = d_i$ is non-integral, i.e. $0 < x_i^* - \lfloor d_i\rfloor < 1$, thus

$$0 < x_i^* - \lfloor d_i\rfloor = f_i - \sum_{j\in N} f_{ij}x_j^* \tag{47}$$

and

$$\sum_{j\in N} f_{ij}x_j^* < f_i, \tag{48}$$

which is a contradiction with the Gomory cut.

Property 2 for the Gomory cuts: Consider an arbitrary integral feasible solution and rewrite the constraint as

$$x_i + \sum_{j\in N}\lfloor w_{ij}\rfloor x_j - \lfloor d_i\rfloor = f_i - \sum_{j\in N} f_{ij}x_j, \tag{49}$$

Left-hand side (LS) is integral, thus right-hand side (RS) is integral. Moreover, $f_i < 1$ a $\sum_{j\in N} f_{ij}x_j \geq 0$, thus RS is strictly lower than 1 and at the same time it is integral, thus lower or equal to 0, i.e. we obtain Gomory cut

$$f_i - \sum_{j\in N} f_{ij}x_j \leq 0. \tag{50}$$

Thus each integral solution fulfills it.

Very simple cuts are so called Dantzig cuts:

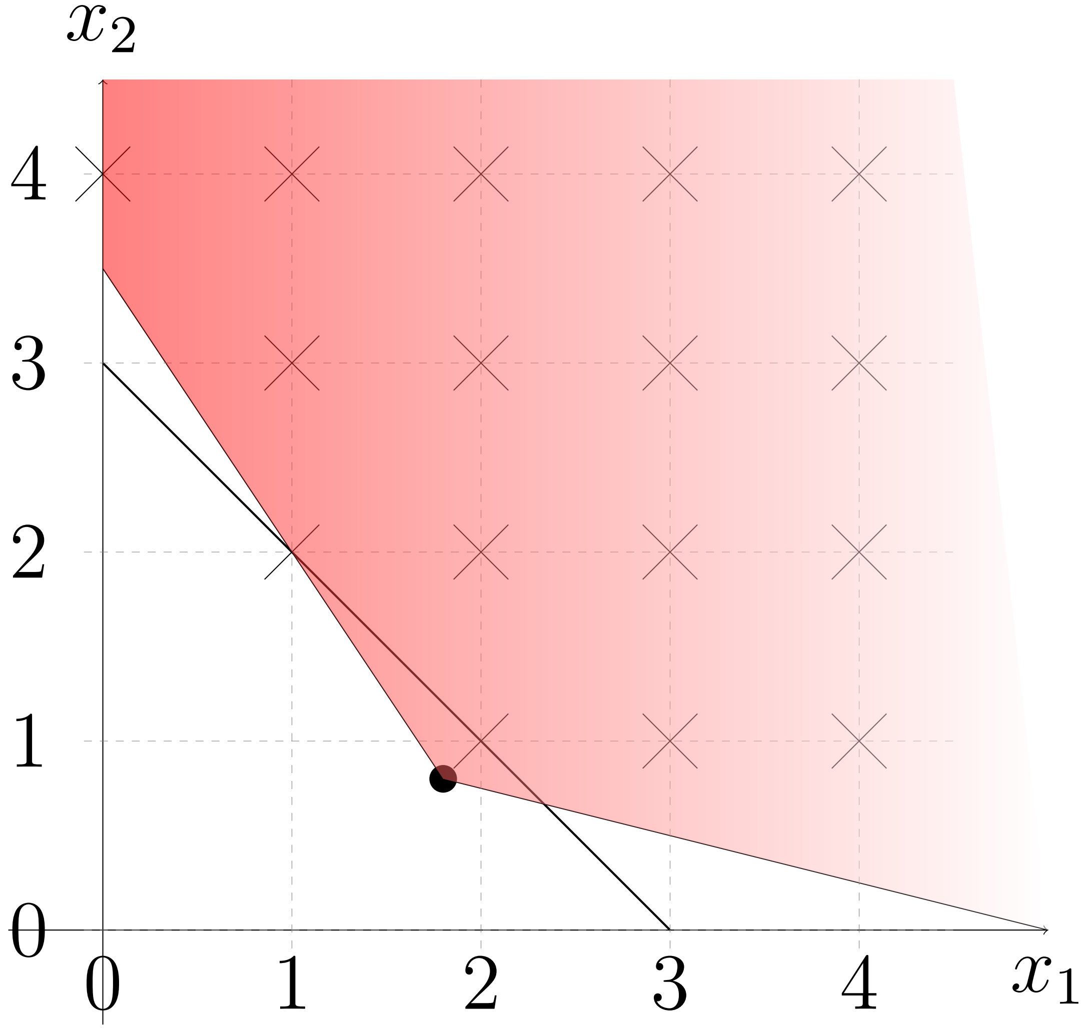
$$\sum_{j\in N} x_j \geq 1. \tag{51}$$

Remind that non-basic variables are equal to zero at the current solution.

Example 3.2 continues ... 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

Figure 3: Cutting plane method – Gomory cut



For example, x_1 is not integral, i.e. we can rewrite the correcponding row of the simplex table as:

$$\begin{aligned} x_1 + 2/10x_3 - 4/10x_4 &= 18/10, \\ x_1 + (0+2/10)x_3 + (-1+6/10)x_4 &= 1+8/10. \end{aligned}$$

The Gomory cut is then:

$$2/10x_3 + 6/10x_4 \geq 8/10.$$

Note that in the original space the cut has the form

$$x_1 + x_2 \geq 3,$$

which we can obtain from the simplex table by combining the rows using coefficients $(2, 2, -1)$. This combination causes that the coefficient with variables x_3, x_4 disappear.

Adding this cut, we obtain new simplex table:

			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	-6/10	1
		112/10	0	0	-7/10	-11/10	0

Dual simplex algorithm leads to the Gomory cut:

$$4/6x_3 + 1/6x_5 \geq 2/3.$$

After adding this cut and running the dual simplex algorithm, we obtain the optimal solution $(2, 1, 1, 1, 0, 0)$.

3.3.2 Branch-and-Bound

The basic idea of the branch-and-bound algorithm can be described as "divide and rule". If we decompose (partition) the original feasibility set into smaller ones

$$M = M_1 \cup M_2 \cup \cdots \cup M_r,$$

we can minimize the objective function on each of these sets easily to obtain

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

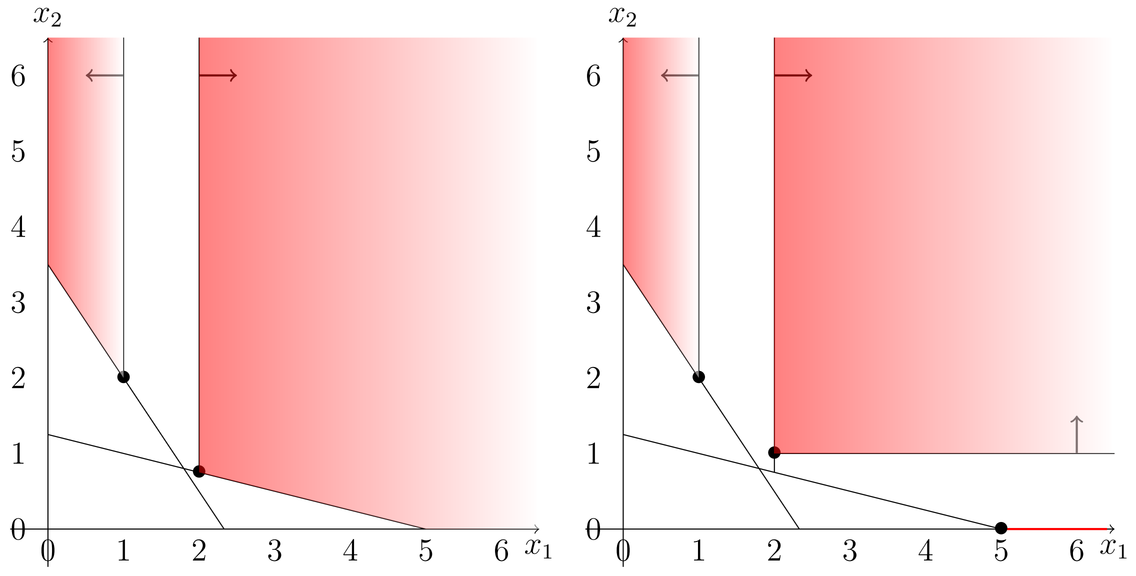
Then, it holds in general

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

General principles of branch-and-bound:

- 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 $x_i \leq \lfloor \hat{x}_i \rfloor$, $x_i \geq \lceil \hat{x}_i \rceil$.

Figure 4: Branch-and-bound: divide and rule of the set S



- *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[1]. Cut inperspective branches before solving (using a bound on the optimal value)

Exact algorithm ..
Branch-and-Bound algorithm:

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.

[1]Branching cannot improve it.

– If $P = \emptyset$ a $f_{min} < \infty$, optimal value and solution are f_{min}, x_{min}.

There is a possible improvement:

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}.

Algorithmic issues include steps which are not uniquely determined but should be clear before the algorithm run:

- *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.

Example

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

After two iterations of the dual SIMPLEX algorithm we obtain the optimal solution of the relaxed problem.

			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

We can see the both positive components of the optimal solution are nonitegral.

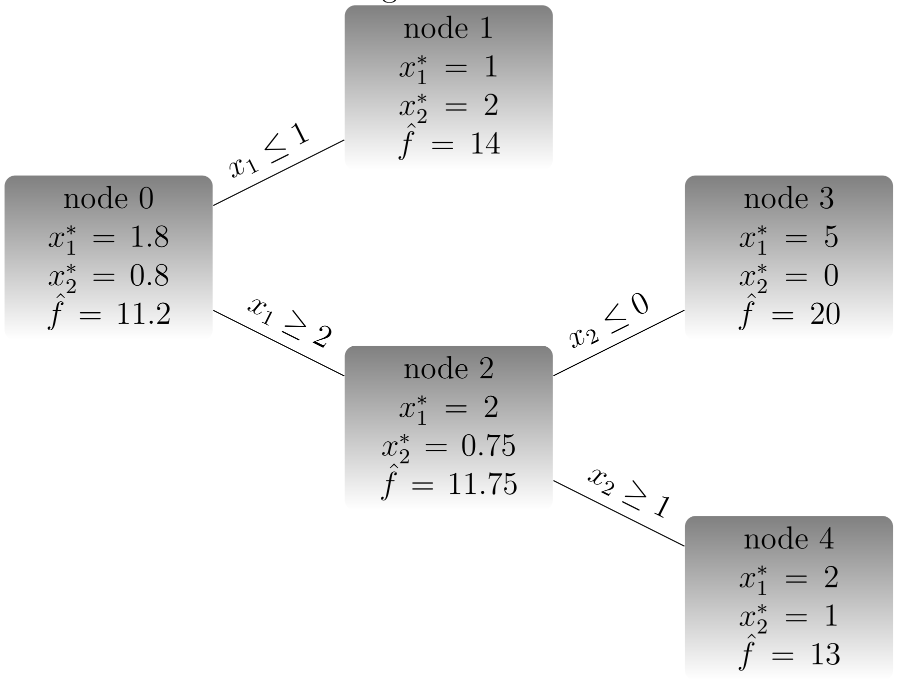
Branching based on the variable x_1 means adding a cut of the form $x_1 \leq 1$ to the table, i.e.

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

			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

The actual solution is dual feasible, but primal infeasible, thus we run iterations of the dual simplex algorithm. Figure 5 shows the iterations of the branch-and-bound algorithm as a binomial tree. Note that this example does not contain any bounding.

Figure 5: Branch-and-bound tree



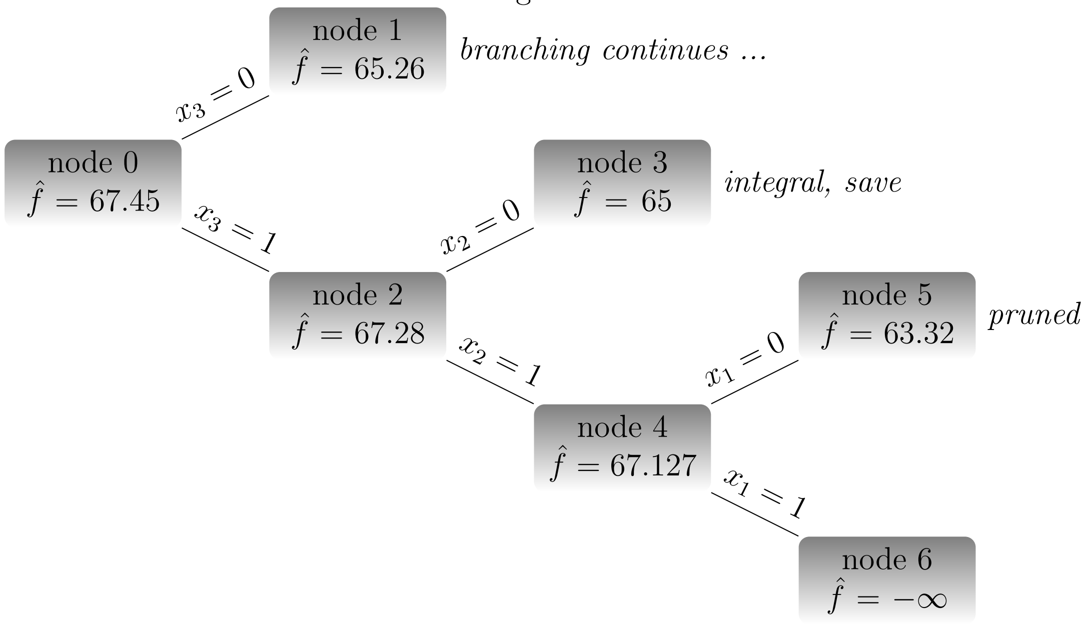
Example We consider problem with binomial variables only

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

For the tree, see Figure 6. If we solve the subproblems in the order how they are numbered, we will arrive at node 3 which provides an integral solution. This solution is saved and used to bound the splitting in node 5. Remind that we are solving a maximization problem, thus further splitting problem in node 5 cannot lead to better (higher) optimal value for an integral solution.

Remark 3.3.
- *If you are able to get a feasible solution quickly, deliver it to the software (solver).*
- *Algorithm termination: B&B usually does not end with an empty queue, but rather with a low (Relative) difference between a lower and an upper bound – construct the upper bound (for minimization) using a feasible solution, lower bound can be based on duality.*
- *Branch-and-Cut: add cuts at the beginning and possibly also during B&B.*

Figure 6: xxx



3.4 Dynamic programming

To introduce the dynamic programming algorithm for integer problems, we return to the knapsack problem

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

Let a_i, b be positive integers. Then we can formulate the value function considering first r items where the capacity serves as the state variable:

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

There are two possibilities:

- if item r is NOT added to the knapsack $\hat{x}_r = 0$, then $f_r(\lambda) = f_{r-1}(\lambda)$,
- if item r is added to the knapsack $\hat{x}_r = 1$ then $f_r(\lambda) = c_r + f_{r-1}(\lambda - a_r)$.

Thus, we arrive at the Bellman recursion for first r items and capacity λ

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

The dynamic programming algorithm can be summarized as follows:

0. Start with $f_1(\lambda) = 0$ for $0 \leq \lambda < a_1$ and $f_1(\lambda) = \max\{0, c_1\}$ for $\lambda \geq a_1$.
1. Use the *forward recursion*
$$f_r(\lambda) = \max\left\{f_{r-1}(\lambda), c_r + f_{r-1}(\lambda - a_r)\right\}.$$
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 $f_{n-1}(b)$, else (if $p_n(b) = 1$) set $\hat{x}_n = 1$ and continue with $f_{n-1}(b - a_n)$ ...

Example 3.4. *Consider the item values $a_1 = 4$, $a_2 = 6$, $a_3 = 7$, costs $c_1 = 4$, $c_2 = 5$, $c_3 = 11$, and the budget $b = 10$:*

$$\begin{aligned} \max \ & \sum_{i=1}^{3} c_i x_i \\ \text{s.t.} \ & \sum_{i=1}^{3} a_i x_i \leq 10, \\ & x_i \in \{0, 1\}. \end{aligned}$$

The algorithm can be represented by a table where each element contains the locally best solution.

	$r/\ \lambda$	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 of the dynamic programming principles include the uncapacitated lot-sizing problem, cf. Subsection 3.1.3, and the shortest path problem.

3.5 Introduction to computational complexity

This is a short introduction to computational complexity theory adopted from Wolsey (1998). First, consider *decision problems* having YES–NO answers. Any optimization problem

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

can be replaced by a decision (for some k integral)

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

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

Example Consider the knapsack decision problem with an instance

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

where the length of the input is

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

Now, we can define the running time for an algorithm.

Definition 3.5. *Let $f_A(X)$ be the number of elementary calculations required to run the algorithm A on the instance $X \in P$. Then the 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 $p \in \mathbb{N}$.

Note that the running time of an algorithm is based on the worst behaviour ($\sup_X$) over all problem instances.

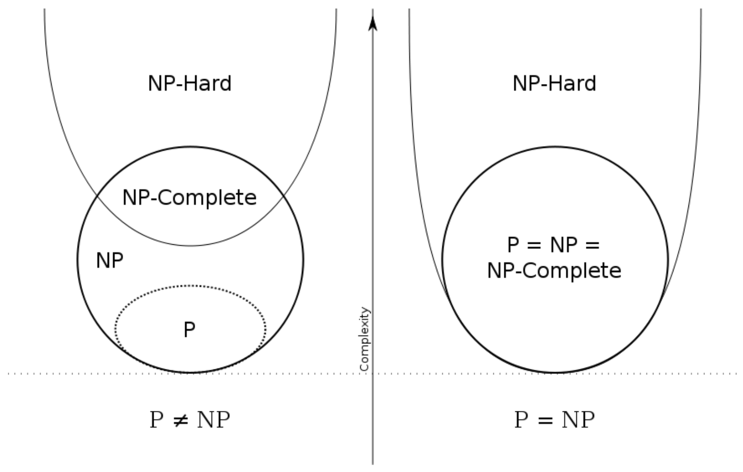
Based on the running time, we can define classes $\mathcal{NP}$ and $\mathcal{P}$.

Definition 3.6. *$\mathcal{NP}$ (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.

The class $\mathcal{NP}$ may be equivalently defined as the set of decision problems that can be solved in polynomial time on a non-deterministic Turing machine[2].

[2]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".

Figure 7: Euler diagram: Is $\mathcal{P} = \mathcal{NP}$?



Definition 3.7. *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.*

Definition 3.8. *$\mathcal{NPC}$, 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.*

The following proposition provides a way how to estimate the complexity of the problems using the polynomial reduction within the classes.

Proposition 3.9. *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 $Q \in \mathcal{NPC}$.*

Even nowadays, an open question is whether there is a problem both in $\mathcal{P}$ and $\mathcal{NPC}$.

Proposition 3.10. *If $\mathcal{P} \cap \mathcal{NPC} \neq \emptyset$, then $\mathcal{P} = \mathcal{NPC}$.*

The relations between the classes are captures by the Euler diagram.

We can return back to the optimiyation problems and define the class of $\mathcal{NP}$-hard problems.

Definition 3.11. *An optimization problem for which the decision problem lies in $\mathcal{NPC}$ is called $\mathcal{NP}$-hard.*

We can shortly discuss complexity of the simplex algorithm. Klee–Minty (1972) proposed an example where the simplex has an exponential complexity:

$$\begin{aligned} \max \ & \sum_{j=1}^{n} 10^{n-j} x_j \\ \text{s.t. } & 2\sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \le 100^{i-1}, \ i = 1, \dots, n, \\ & x_j \ge 0, \ j = 1, \dots, n. \end{aligned} \tag{52}$$

The linear program 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.

3.6 Totally unimodular matrices and network flows

Totally unimodular matrices

Definition 3.12. *A matrix A is totally unimodular (TU) iff every square submatrix of A has determinant +1, -1, or 0.*

The linear program has an integral optimal solution for all integer r.h.s. b if and only if A is TU.

... based on Laplace expansion for the determinant of a basic matrix and the Cramer rule.

A set of **sufficient conditions**:

- $a_{ij} \in \{-1, 0, 1\}$ for all i, j
- Each column contains at most two nonzero coefficients, i.e. $\sum_{i=1}^{m} |a_{ij}| \leq 2$,
- There exists a partitioning $M_1 \cap M_2 = \emptyset$ of the rows $1, \ldots, m$ such that each column j containing two nonzero coefficients satisfies

$$\sum_{i \in M_1} a_{ij} = \sum_{i \in M_2} a_{ij}.$$

If A is TU, then A^T and $(A|I)$ are TU.

3.6.1 Minimum cost network flow problem

- $G = (V, A)$ – graph with vertices V and (oriented) arcs A
- h_{ij} – arc capacity
- c_{ij} – flow cost
- b_i – demand, ASS. $\sum_i b_i = 0$
- $V^+(i) = \{k : \ (i,k) \in A\}$ – successors of i
- $V^-(i) = \{k : \ (k,i) \in A\}$ – predecessors of i

$$\begin{aligned} \min_{x_{ij}} \ & \sum_{(i,j) \in A} c_{ij} x_{ij} \\ \text{s.t.} \ & \sum_{k \in V^+(i)} x_{ik} - \sum_{k \in V^-(i)} x_{ki} = b_i, \ i \in V, \\ & 0 \leq x_{ij} \leq h_{ij}, \ (i,j) \in A. \end{aligned}$$

Wolsey (1998), Ex. 3.1 ($M_1 = \{1, \ldots, m\}, M_2 = \emptyset$)

Figure 8: xxx

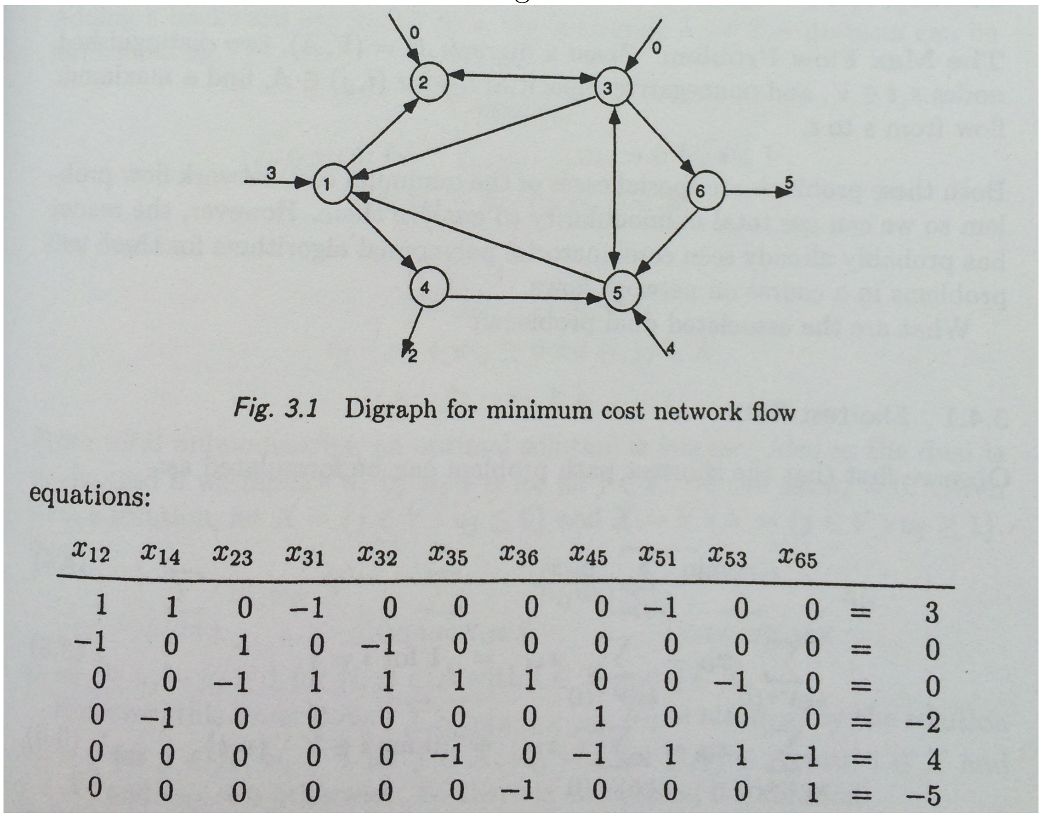


Fig. 3.1 Digraph for minimum cost network flow

equations:

x_{12}	x_{14}	x_{23}	x_{31}	x_{32}	x_{35}	x_{36}	x_{45}	x_{51}	x_{53}	x_{65}		
1	1	0	−1	0	0	0	0	−1	0	0	=	3
−1	0	1	0	−1	0	0	0	0	0	0	=	0
0	0	−1	1	1	1	1	0	0	−1	0	=	0
0	−1	0	0	0	0	0	1	0	0	0	=	−2
0	0	0	0	0	−1	0	−1	1	1	−1	=	4
0	0	0	0	0	0	−1	0	0	0	1	=	−5

x_{12}	x_{14}	x_{23}	x_{31}	x_{32}	x_{35}	x_{36}	x_{45}	x_{51}	x_{53}	x_{65}		
1	1	0	-1	0	0	0	0	-1	0	0	=	3
-1	0	1	0	-1	0	0	0	0	0	0	=	0
0	0	-1	1	1	1	1	0	0	-1	0	=	0
0	-1	0	0	0	0	0	1	0	0	0	=	-2
0	0	0	0	0	-1	0	-1	1	1	-1	=	4
0	0	0	0	0	0	-1	0	0	0	1	=	-5

Special cases

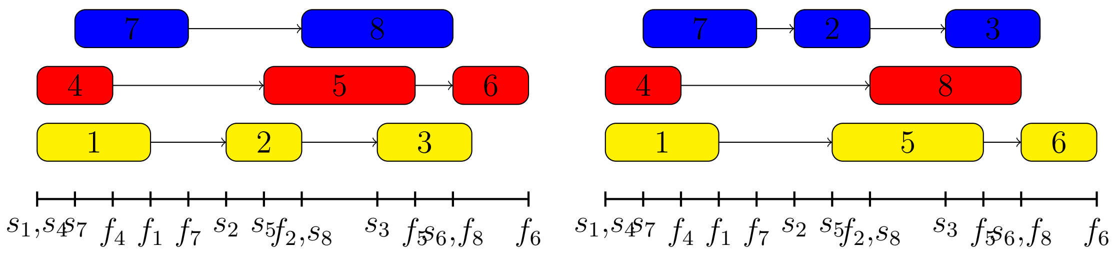
- Shortest path problem
- Critical (longest time) path problem in project scheduling (PERT = Program Evaluation and Review Technique)
- Fixed interval scheduling
- Transportation problem

3.6.2 Shortest path problem

Find a minimum cost $s - t$ **path** given nonnegative arc costs c_{ij}, set

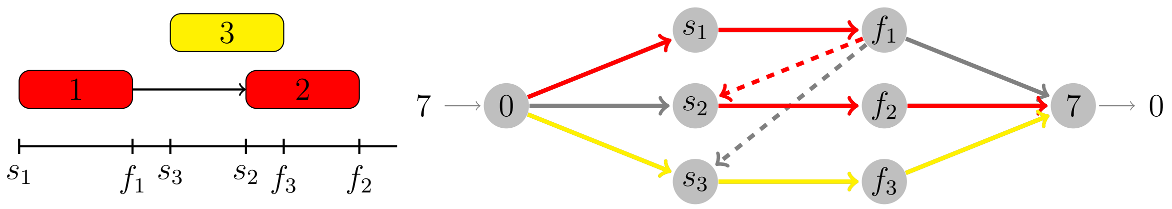
- $b_i = 1$ if $i = s$,

Figure 9: Two different assignments of 8 jobs to 3 machines



7
8
4
5
6
1
2
3
s_1, s_4 s_7 f_4 f_1 f_7 s_2 s_5 f_2, s_8 s_3 f_5 s_6, f_8 f_6
7
2
3
4
8
1
5
6
s_1, s_4 s_7 f_4 f_1 f_7 s_2 s_5 f_2, s_8 s_3 f_5 s_6, f_8 f_6

Figure 10: Fixed interval schedule and corresponding network flow



- $b_i = -1$ if $i = t$,
- $b_i = 0$ otherwise.

Then the problem can be formulated as

$$\begin{aligned}
\min_{x_{ij}} \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{k\in V^+(i)} x_{ik} - \sum_{k\in V^-(i)} x_{ki} = 1, \ i = s, \\
& \sum_{k\in V^+(i)} x_{ik} - \sum_{k\in V^-(i)} x_{ki} = 0, \ i \in V \setminus \{s,t\}, \\
& \sum_{k\in V^+(i)} x_{ik} - \sum_{k\in V^-(i)} x_{ki} = -1, \ i = t, \\
& 0 \le x_{ij} \le 1, \ (i,j) \in A.
\end{aligned}$$

$\hat{x}_{ij} = 1$ identifies the shortest path.

3.6.3 Fixed interval scheduling

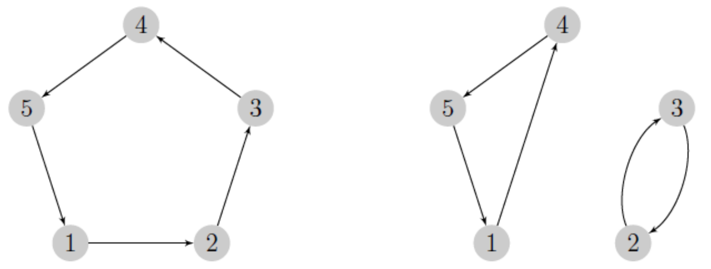
Basic **Fixed interval scheduling** (FIS) problem: given J jobs with prescribed starting s_j and finishing f_j times, find a minimal number of identical machines that can process all jobs such that no processing intervals intersect.

FIS – network flow reformulation

Network structure:

1. $2J + 2$ **vertices** $\mathcal{V}$: $\{0, s_1, f_1, \dots s_J, f_J, 2J + 1\}$; vertices 0, $2J + 1$ correspond to the source and sink,

Figure 11: 5 towns – cycle and subcycles (subroutes), Kafka (2013)



2. **oriented edges** E: $\{0, s_j\}$, $\{s_j, f_j\}$, $j \in \mathcal{J}$, $\{f_i, s_j\}$ if $f_i \leq s_j$, $\{f_j, 2J+1\}$, $j \in \mathcal{J}$, $(2J+1, 0)$
3. demands: $d_0 = d_{2J+1} = 0$, $d_{s_j} = -1$, $d_{f_j} = 1$, $j \in \mathcal{J}$,
4. return edge $(2J+1, 0)$: capacity $u_{2J+1,0} = M$, $c_{2J+1,0} = 1$,
5. edge capacities $u_{uv} = 1$, and costs $c_{uv} = 0$, $(u, v) \in E \setminus (2J+1, 0)$.

Solve the min-cost network flow problem.

3.7 Traveling salesman problem

Traveling salesman problem is characterized as follows

- Consider n towns and in one of them there is a traveling salesman.
- Traveling salesman must visit all towns and return back.
- For each pair of towns the traveling costs are known and the traveling salesman is looking for the cheapest route.

= Finding a Hamilton cycle in a graph with edge prices.

Assignment problem

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{53}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \; j = 1, \ldots, n, \tag{54}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \; i = 1, \ldots, n, \tag{55}$$

$$x_{ij} \in \{0, 1\}. \tag{56}$$

We minimize the traveling costs, we arrive to j from exactly one i, we leave i to exactly one j.

Subroute elimination conditions I

- $x_{ii} = 0$, $c_{ii} = \infty$

- $x_{ij} + x_{ji} \leq 1$
- $x_{ij} + x_{jk} + x_{ki} \leq 2$
- ...
- $\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1$, $S \subseteq \{1,\dots,n\}$, $2 \leq |S| \leq n-1$

Approximately 2^n inequalities, it is possible to reduce to $|S| \leq \lceil n/2 \rceil$.

Subroute elimination conditions II

Other valid inequalities:

$$u_i - u_j + nx_{ij} \leq n-1, i,j = 2,\dots,n.$$

Eliminate subroutes: There is at least one route which does not go through vertex 1, denote this route by C and the number of edges by $|E(C)|$. If we sum all these inequalities over all edges $\{i,j\}$, which are in C, i.e. the corresponding variables $x_{ij} = 1$, we obtain

$$n|E(C)| \leq (n-1)|E(C)|, \tag{57}$$

which is a contradiction.

Hamilton cycle is feasible: let the vertices be ordered as $v_1 = 1$, v_2, ..., v_n. We set $u_i = l$, if $v_l = i$, i.e. u_i represent the order. For each edge of the cycle $\{i,j\}$ it holds $u_i - u_j = -1$, i.e.

$$u_i - u_j + nx_{ij} = -1 + n \leq n-1. \tag{58}$$

For edges, which are not in the cycle, the inequality holds too: $u_i - u_j \leq n-1$ a $x_{ij} = 0$.

Subroute elimination conditions – example

Consider subroutes: 1–4–5, 2–3

Add inequalities

$$u_2 - u_3 + 5x_{23} \leq 4,$$
$$u_3 - u_2 + 5x_{32} \leq 4,$$

or

$$x_{23} + x_{32} \leq 1.$$

Computational complexity: $\mathcal{NP}$ (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.

3.7.1 Traveling Salesman Problem with Time Windows

- t_i – time when customer i is visited
- T_{ij} – time necessary to reach j from i
- l_i, u_i – lower and upper bound (time window) for visiting customer i
- M – a large constant

$$
\begin{aligned}
\min_{x_{ij},t_i} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} & & \qquad (59)\\
\sum_{i=1}^{n} x_{ij} &= 1,\ j = 1,\dots,n, & (60)\\
\sum_{j=1}^{n} x_{ij} &= 1,\ i = 1,\dots,n, & (61)\\
t_i + T_{ij} - t_j &\le M(1 - x_{ij})\ i,j = 1,\dots,n, & (62)\\
l_i \le t_i &\le u_i,\ i = 1,\dots,n, & (63)\\
x_{ij} &\in \{0,1\}.
\end{aligned}
$$

3.7.2 Capacitated Vehicle Routing Problem

Parameters

- n – number of customers
- 0 – depo (starting and finishing point of each vehicle)
- K – number of vehicles (homogeneous)
- $d_j \ge 0$ – customer demand, for depo $d_0 = 0$
- $Q > 0$ – vehicle capacity ($KQ \ge \sum_{j=1}^{n} d_j$)
- c_{ij} – transportation costs from i to j (usually $c_{ii} = 0$)

Decision variables

- x_{ij} – equal to 1, if j follows after i on the route, 0 otherwise
- u_j – upper bound on transported amount after visiting customer j

$$\min_{x_{ij},u_i} \sum_{i=0}^{n}\sum_{j=0}^{n} c_{ij}x_{ij} \tag{64}$$

$$\begin{aligned}
\sum_{i=0}^{n} x_{ij} &= 1,\ j=1,\dots,n, && (65)\\
\sum_{j=0}^{n} x_{ij} &= 1,\ i=1,\dots,n, && (66)\\
\sum_{i=1}^{n} x_{i0} &= K, && (67)\\
\sum_{j=1}^{n} x_{0j} &= K, && (68)\\
u_i - u_j + d_j &\le Q(1-x_{ij})\ i,j=1,\dots,n, && (69)\\
d_i \le u_i &\le Q,\ i=1,\dots,n, && (70)\\
x_{ij} &\in \{0,1\}.
\end{aligned}$$

(64) minimization of transportation costs

(65) exactly one vehicle arrives to customer j

(66) exactly one vehicle leaves customer i

(67) exactly K vehicles return to depot 0

(68) exactly K vehicles leave depot 0

(69) balance conditions of transported amount (serve also as subroute elimination conditions)

(70) bounds on the vehicle capacity

(All vehicles are employed.)

3.8 Heuristic algorithms

3.8.1 Greedy heuristic

Start with an empty set (solution) and choose the item with **the best immediate reward** at each step.

Example: Traveling Salesman Problem with the (symmetric) distance matrix

$$\begin{pmatrix}
- & 9 & 2 & 8 & 12 & 11\\
 & - & 7 & 19 & 10 & 32\\
 & & - & 29 & 18 & 6\\
 & & & - & 24 & 3\\
 & & & & - & 19\\
 & & & & & -
\end{pmatrix}$$

Greedy steps: 1–3 (2), 3–6 (6), 6–4 (3), 4–5 (24), 5–2 (10), 2–1 (9), i.e. the route length is 54.

3.8.2 Local search heuristic

Choose an initial solution x and search its neighborhood $U(x)$. Repeat until you are able to find a better solution, i.e. if $y \in U(x)$, $f(y) < f(x)$, set $x = y$.

Example: Traveling Salesman Problem, define the neighborhood $U(x)$ as **2-exchange**, i.e. if $S = \{(i,j) \in A : \ x_{ij} = 1\}$ is a feasible solution, then

$$U(x) = \{S' : \ |S \cap S'| = n - 2\},$$

in other words: **replace edges** (i,j), (i',j') by (i,i'), (j,j').

Greedy steps: 1–3 (2), 3–6 (6), 6–4 (3), 4–5 (24), 5–2 (10), 2–1 (9), i.e. the route length is 54.

2-exchange: 1–3 (2), 3–4 (29), 4–6 (3), 6–5 (19), 5–2 (10), 2–1 (9), i.e. the route length is 72.

3.8.3 Basic heuristics for VRP

Insertion heuristic:

- Start with empty routes.
- FOR all customers DO: Insert the customer to the place in a route where it causes the lowest increase of the traveled distance.

Clustering:

- Cluster the customers according to their geographic positions ("angles").
- Solve[3] the traveling salesman problem in each cluster.

Possible difficulties: time windows, vehicle capacities, ...

3.8.4 Tabu search for VRP

For a given number of iteration, run the following steps:

- Find the best solution in a *neighborhood* of the current solution. Such solution can be worse than the current one or even infeasible (use a penalty function).
- Forbid moving back for a random number of steps by actualizing the **tabu list**.
- Remember the best solution.

The tabu search algorithm enables moving from local solutions (compared with a simple "hill climbing alg.").

[3]..exactly, if the clusters are not large.

3.8.5 Genetic algorithms

Iterative procedure:

- Population – finite set of individuals with genes
- Generation
- Evaluation – fitness
- Parent selection
- Crossover produces one or two new solutions (offspring).
- Mutation
- Population selection

3.8.6 Rich Vehicle Routing Problems

- **Goal** – maximization of the ship *filling rate* (operational planning), optimization of fleet composition, i.e. number and capacity of the ships (strategic planning)
- **Rich Vehicle Routing Problem**
  - time windows
  - heterogeneous fleet (vehicles with different capacities and speed)
  - several depots with inter-depot trips
  - several routes during the planning horizon
  - *non-Euclidean distances* (fjords)
- Mixed-integer programming :-(, constructive heuristics for getting an initial feasible solution and tabu search
- M. Branda, K. Haugen, J. Novotný, A. Olstad, **Downstream logistics optimization at EWOS Norway**. Research report.

Our approach used

- Mathematical formulation
- GAMS implementation
- Heuristic (insertion heuristic, tabu search) implementation
- Decision Support System (DSS)

Literature

- L. Adam: Nelinearity v úlohách stochastického programování: aplikace na řízení portfolia. Diplomová práce MFF UK, 2011. (IN CZECH)
- Bazaraa, M.S., Sherali, H.D., and Shetty, C.M. (2006). Nonlinear programming: theory and algorithms, Wiley, Singapore, 3rd edition.
- J.F. Benders (1962): Partitioning procedures for solving mixed-variables programming problems, Numerische Mathematik 4(3), 238–252.
- Boyd, S., Vandenberghe, L. (2004). Convex Optimization, Cambridge University Press, Cambridge.
- M. Branda, K. Haugen, J. Novotný, A. Olstad (2017). Downstream logistics optimization at EWOS Norway. Research report.
- M. Branda, J. Novotný, A. Olstad (2016) Fixed interval scheduling under uncertainty - a tabu search algorithm for an extended robust coloring formulation. Computers & Industrial Engineering 93, 45–54.
- O. Kafka: Optimální plánování rozvozu pomocí dopravních prostředků, Diploma thesis MFF UK, 2013. (IN CZECH)
- P. Kall, J. Mayer (2005). Stochastic Linear Programming: Models, Theory, and Computation. Springer.
- 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 1â“9, 1969). New York-London: Academic Press, 159–175.
- P. Lachout (2011). Matematické programování. Skripta k (zaniklé) přednášce Optimalizace I (IN CZECH).
- Matoušek and Gärtner (2007). Understanding and using linear programming, Springer.
- 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.
- P. Toth, D. Vigo (2002). The vehicle routing problem, SIAM, Philadelphia.
- 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]