Lesson 1

Introduction, Linear Programming, Combinatorial Optimization

Carlo Mannino
SINTEF Digital and University of Oslo

Outline

1. Introduction
2. Linear Programming and Polyhedra
3. Combinatorial Optimization
4. Exercises
5. Algorithms for Combinatorial optimization

Optimization and Learning

- (Mathematical) Optimization (MO) and (Machine) Learning (ML) are very intertwined
- Various MO techniques are exploited in ML algorithms (and increasingly vice-versa)
- We will give some basic knowledge on MO techniques with some application to relevant ML methods.

Structure of the course

- Instructors: Torkel Haufmann, Carlo Mannino
- Carlo Mannino: Combinatorial optimization
  - Topics: linear programming, integer and combinatorial optimization, network optimization
  - Applications: Classification trees, graphical models
- Torkel Haufmann: Convex and non-linear optimization
  - Topics: Continuous and convex optimization, duality, (sub)gradient methods, automatic differentiation
  - Applications: Support vector machines, training methods, ML in optimization

Machine Learning

- Many different definitions.
- The study of computer algorithms that improve automatically through experience (Tom Mitchell, 1997)
- The field of inquiry devoted to understanding and building methods that 'learn' (Wikipedia)
- A set of techniques used to try to elicit pattern recognition (and similar abilities) in computers (Torkel Haufmann, 2022)

Mathematical Optimization

- $C \subseteq \mathbb{R}^n$ is the set of *feasible solutions*.
- $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*

$$\begin{aligned} \max_x \quad & f(x) \\ & x \in C. \end{aligned}$$

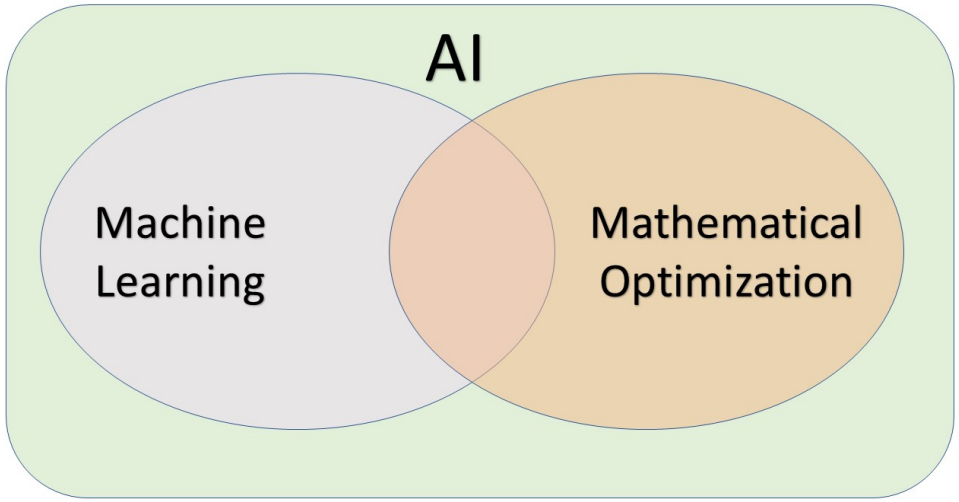
- We can always transform a maximization problem into an equivalent minimization problem
- Just let $g(x) = -f(x)$ and then solve $\{\min g(x) : x \in C\}$.
- (Similarly: a constraint $g(x) \geq b$ can be written as $-g(x) \leq -b$).

Mathematical Optimization and Learning

- Learning is exploited to enhance optimization solution algorithms
- Mathematical Optimization is a main component in several learning algorithms
- Mathematical optimization and Learning are the two legs on which Artificial Intelligence is walking

AI, ML, MO

- Machine Learning: classify and make predictions (*descriptive, predictive analytics*)
- Mathematical Optimization: make decisions (*prescriptive analytics*)



Mathematical Programming: basic definitions

$$\begin{aligned} \max_x \quad & f(x) \\ & x \in C. \end{aligned} \tag{1}$$

- If $C = \emptyset$ then the problem is **infeasible.**
- If, for any $y \in C$ there exists $x \in C$ such that $f(x) > f(y)$ then the problem is **unbounded.**
- **Optimal solution**: a point $x^* \in C$ such that $f(x^*) \geq f(y)$ for all $y \in C$.

Linear Programming

$$\begin{aligned} \max_x \quad & f(x) \\ & x \in C. \end{aligned} \tag{2}$$

- The set C is often represented by a set of constraints
  $C = \{x \in \mathbb{R}^n : g_1(x) \leq 0, \ldots, g_m(x) \leq 0, h_1(x) = 0, \ldots, h_p(x) = 0\}$,
  where $g_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, m$, $h_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, p$.
- When the objective function and the constraints are linear functions, then the problem is **Linear Programming**

$$\begin{aligned} \max \quad & x_1 + 3x_2 + 0.5x_3 \\ & x_1 + 3x_3 \leq 10 \\ & x_2 + x_3 = 4 \\ & x_1 \geq 0 \end{aligned} \tag{3}$$

An example of an LP-formulation

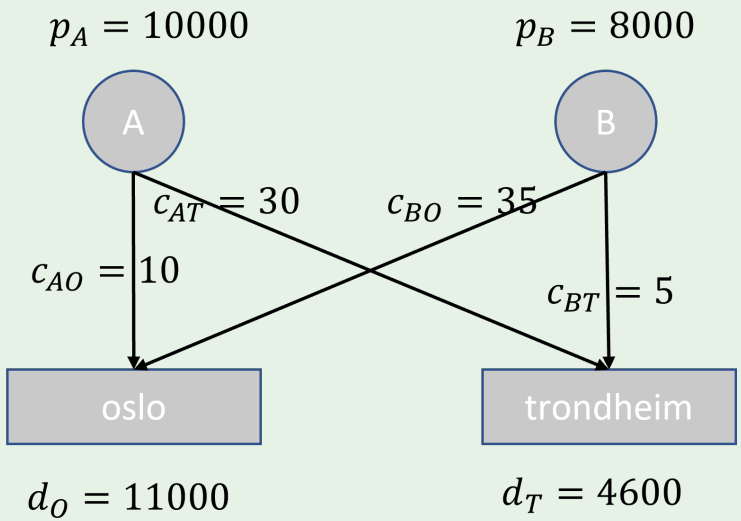
We can model several classes of problems as linear programs.

Example (A transportation problem)

A company produces one good in two production centers (A and B). The good must be transported in two distribution centers, one in Oslo and one in Trondheim.

Demand: Oslo $d_O = 11000$, Trondheim $d_T = 4600$

Max production capacity: A, $c_A = 10000$; B, $c_B = 8000$



Find the quantities to produce in each production center and to transport to each destination, which minimize the overall transportation cost.

Example of formulation - decision variables

- Step 1: decision variables.
  - x_1 quantity to be transported from A to Oslo
  - x_2 quantity to be transported from A to Trondheim
  - x_3 quantity to be transported from B to Oslo
  - x_4 quantity to be transported from B to Trondheim
- Step 2: objective function.

$$\min 10x_1 + 30x_2 + 35x_3 + 5x_4$$

Example of formulation - constraints

- Step 3: constraints.

  Demand constraints:

  - $x_1 + x_3 \geq 11000$ total quantity transported to Oslo
  - $x_2 + x_4 \geq 4600$ total quantity transported to Trondheim

  Production limits:

  - $x_1 + x_2 \leq 10000$ total quantity produced in A
  - $x_3 + x_4 \leq 8000$ 89 total quantity produced in B

  Non-negativity:

  - $x_1, x_2, x_3, x_4 \geq 0$

Linear Programming: Standard Form

- We can represent linear programs in compact form.
- Let $Ax \leq b$, with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $x \in \mathbb{R}^n$ be the constraints of a linear program.
- If the linear program contains the constraint $x_i \geq 0$, then variable x_i is *restricted in sign*. Otherwise x_i is *unrestricted*.
- An LP instance in *standard form* is written as

$$\begin{aligned} \max_x \quad & c^T x \\ & Ax \leq b \\ & x \geq \mathbf{0} \end{aligned} \tag{4}$$

with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$

Standard VS non-standard form

- A linear program in non-standard form:

$$\begin{aligned} \min \quad & x_1 + 4x_2 + 0.5x_3 \\ & x_1 + 2x_3 \geq 10 \\ & 3x_2 + x_3 = 4 \\ & x_1, x_3 \geq 0 \end{aligned}$$

  - min instead of max
  - $\geq$ and $=$ constraints instead of $\leq$
  - x_2 unrestricted in sing

- A linear program in **standard form**:

$$\begin{aligned} \max \quad & x_1 + x_2 + 4x_3 \\ & x_1 + 2x_2 + 3x_3 \leq 10 \\ & 3x_2 + 4x_3 \leq 5 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Universality of the standard form

Definition

Two optimization problems $P1$: $\{\max f(x) : x \in C \subseteq \mathbb{R}^n\}$ and $P2$: $\{\max g(y) : y \in D \subseteq \mathbb{R}^m\}$ are *equivalent* if for any solution $\bar{x}$ to $P1$ there exists a solution $\bar{y}$ to $P2$, with $f(\bar{x}) = g(\bar{y})$, and for any solution $\tilde{y}$ to $P2$ there exists a solution $\tilde{x}$ to $P1$, with $f(\tilde{x}) = g(\tilde{y})$.

Lemma

Every linear program can be transformed into an equivalent linear program in standard form

Universality of the standard form

Exercise

Given a linear program with some variables unrestricted in sign, equality constraints, and in minimization form, show that it can be transformed into an equivalent one in standard form.
Hint: you are allowed to introduce new variables

Exercise

Transform the following LP into an equivalent one in standard form

$$\begin{aligned} \min \quad & x_1 + 4x_2 + 0.5x_3 \\ & x_1 + 2x_3 \geq 10 \\ & 3x_2 + x_3 = 4 \\ & x_1, x_3 \geq 0 \end{aligned}$$

Hyperplanes and halfspaces

Definition

Let a be a nonzero vector in $\mathbb{R}^n$, and b be a scalar.

The set $\{x \in \mathbb{R}^n : a^T x = b\}$ is called a *hyperplane*.

The set $\{x \in \mathbb{R}^n : a^T x \leq b\}$ is called a *halfspace*.

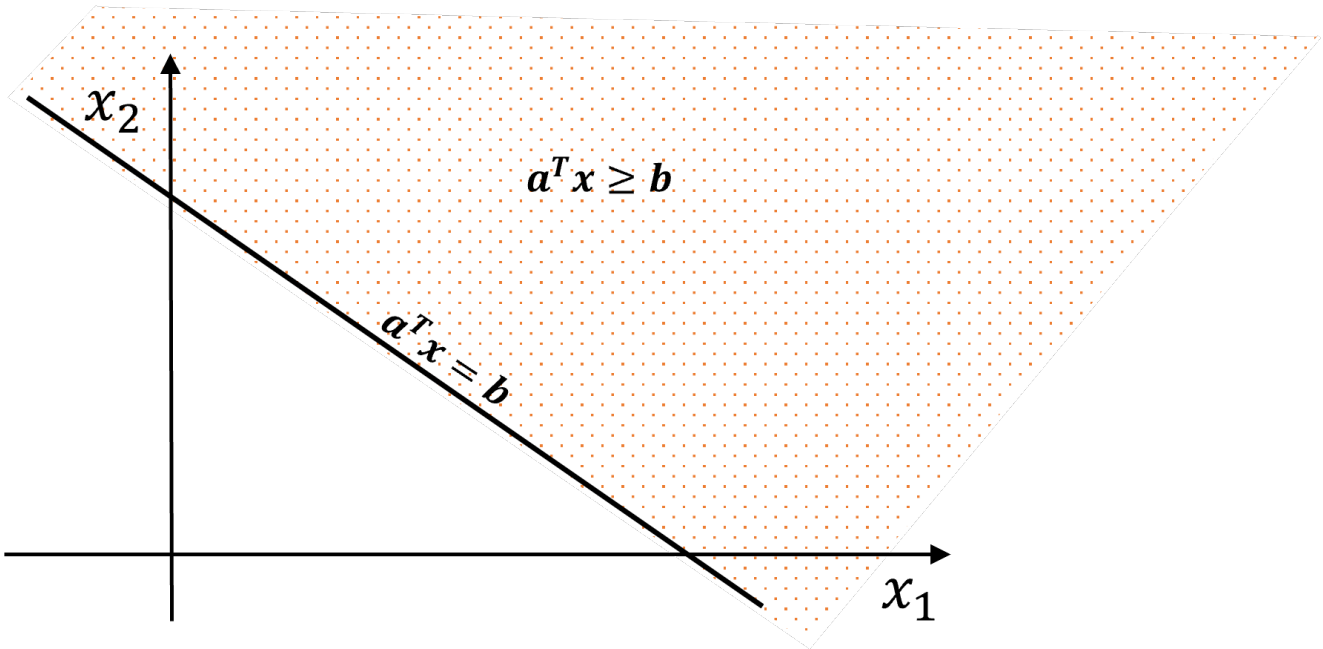


Figure: The hyperplane $a^t x = b$ and the halfspace $a^t x \geq b$

Polyhedra

Definition

A *polyhedron* is the set of points $x \in \mathbb{R}^n$ satisfying a set of linear inequalities $Ax \leq b$, where A is a $m \times n$ real matrix and $b \in \mathbb{R}^m$.

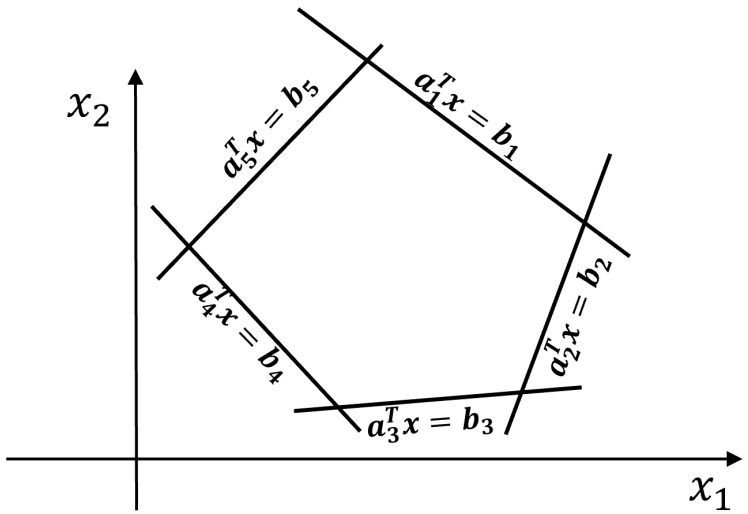


Figure: The polyhedron $Ax \leq b$

- A polyhedron is the intersection of a family of halfspaces
- Solving a linear program amounts to finding the optimal point in a polyhedron, if one exists.

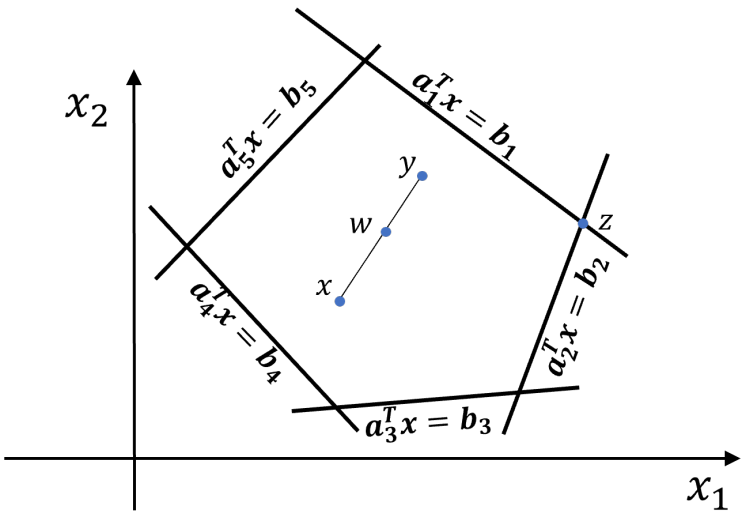
Extreme points of a polyhedron P

Lemma (Convexity of polyhedra.)

Let P be a polyhedron. Let $x, y \in P$, let $\lambda \in [0, 1]$ and let $z = \lambda x + (1 - \lambda)y$. Then $z \in P$.

Definition (Extreme point)

$z \in P$ is an **extreme point** of polyhedron P if we cannot find $x, y \in P$, with $x, y \neq z$, and scalar $\lambda \in [0, 1]$ such that $z = \lambda x + (1 - \lambda)y$.



- z is an extreme point.
- w is not an extreme point, since it belongs to the segment joining $x, y \in P$ with $x \neq w$ and $y \neq w$

Linear Programming: Fundamental theorem

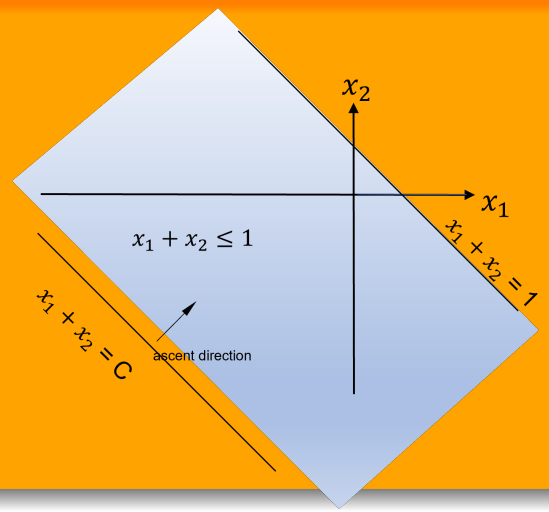
Theorem (Fundamental theorem of LP)

For any linear program in standard form one out of three cases can happen:

- *The problem is infeasible*
- *The problem is unbounded*
- *There is an extreme point which is an optimal solution*

Exercise

- What if the problem is not in standard form?
- For instance $\{\max x_1 + x_2 : x_1 + x_2 \leq 1\}$
- The instance is neither infeasible nor unbounded (optimal value $V^* = 1$), but there are no extreme points.



Linear Programming: Fundamental theorem

Theorem (Fundamental theorem of LP)

For any linear program in standard form one out of three cases can happen:

- *The problem is infeasible*
- *The problem is unbounded*
- *There is an extreme point which is an optimal solution*

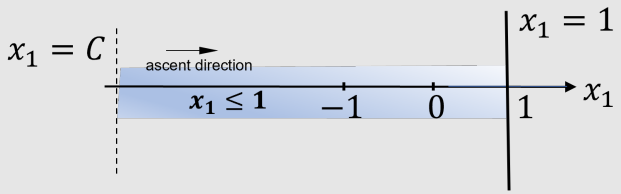
Exercise

- What if the problem is not in standard form?
- For instance $\{\max x_1 + x_2 : x_1 + x_2 \leq 1\}$
- The instance is neither infeasible nor unbounded (optimal value $V^* = 1$), but there are no extreme points.

In the next example we see what happens when the instance is transformed into an equivalent one in standard form.

Instance in non standard form

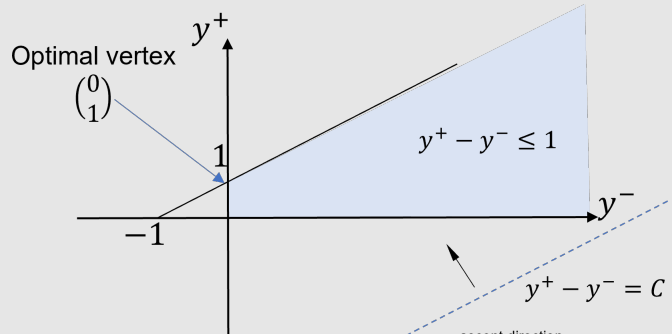
$$\begin{aligned} \max \quad & x_1 \\ & x_1 \leq 1 \\ & x_1 \leqslant 0 \end{aligned}$$



No optimal extreme point. All points on the line $x_1 = 1$ are optimal.

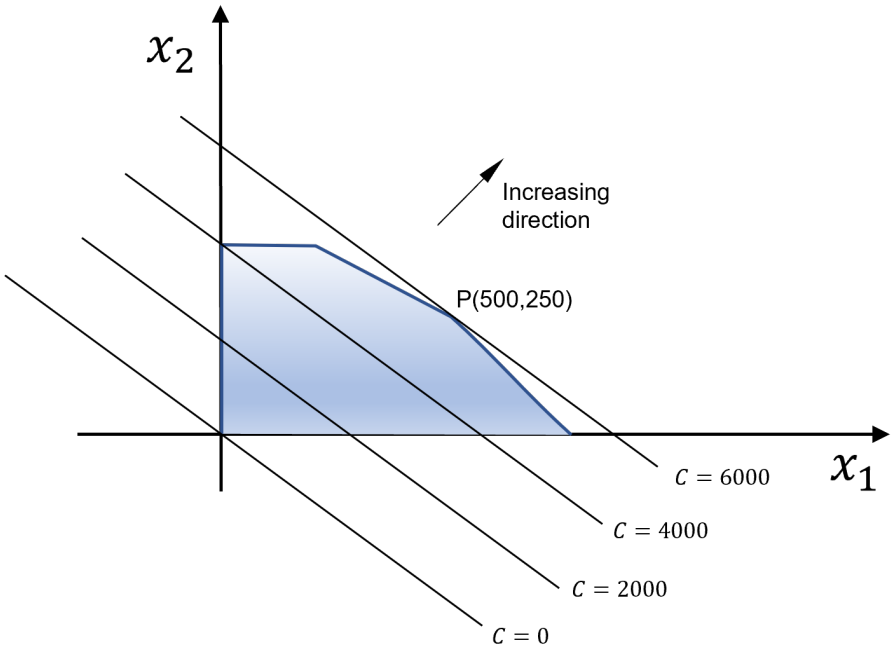
Equivalent instance in standard form ($x_1 = y^+ - y^-$)

$$\begin{aligned} \max \quad & y^+ - y^- \\ & y^+ - y^- \leq 1 \\ & y^+, y^- \geq 0 \end{aligned}$$



Linear Programming: Graphical Solution

$$\begin{cases} \max C = 7x_1 + 10x_2 \\ x_1 + x_2 \leq 750 \\ x_1 + 2x_2 \leq 1000 \\ x_2 \leq 400 \\ x_1 \geq 0, x_2 \geq 0. \end{cases}$$



- In two dimensions we can solve the problem graphically
- Draw the feasible region
- Draw the objective function through a feasible point.
- ”Drag” it in the direction of increase until it reaches the ”last” feasible point
- This is the optimum. Why?

Linear Programming: famous solution algorithms

There are several methods and commercial / open source tools, to solve linear programming instances.

- The simplex method
- The ellipsoid method
- Methods for ad-hoc problems
- ...

Solving an LP can be done very efficiently.

Mixed Integer Linear Programming

- If some of the variables of a linear programs are constrained to be integer, then the problem is called a *Mixed Integer Linear Program* (MILP):

$$\begin{aligned} &\max c^T x + d^T y \\ &Ax + Dy \leq b \\ &x \in \mathbb{R}^p, y \in Z^q, A \in \mathbb{R}^{m \times p}, D \in \mathbb{R}^{m \times q}, b \in \mathbb{R}^m. \end{aligned} \tag{5}$$

- Integer Linear Program (ILP): all variables are integer.
- Binary Linear Program (0,1-LP): all variables are 0 or 1.

A toy example of 01-LP formulation

Example (A project selection problem)

We are given two projects, say A, B. The expected returned profits are 10 MNOK and 12 MNOK, respectively. Running project A costs of 5 MNOK, while B costs 7 MNOK. The total budget is 10 MNOK. Find the projects to activate so to not exceed the budget and maximise the total profit.

- Step 1: decision variables.
  - $x_A \in \{0, 1\}$: 1 if project A is run, 0 otherwise
  - $x_B \in \{0, 1\}$: 1 if project B is run, 0 otherwise
- Step 2: objective function.
  - $\max 10x_A + 12x_B$
- Step 3: budget constraint.
  - $5x_A + 7x_B \leq 10$

How difficult is it to solve an LP?

- Any instance of LP can be solved efficiently, i.e. Linear Programming is in the class P of polynomially solvable problems.

How difficult is it to solve an LP?

- Any instance of LP can be solved efficiently, i.e. Linear Programming is in the class P of polynomially solvable problems.
- Roughly speaking this means that, if the LP-instance contains m constraints and n variables, the instance can be solved (by some algorithm) in a number of elementary steps not greater than $km^q n^p$, with k, q, p given constants.

How difficult is it to solve an LP?

- Any instance of LP can be solved efficiently, i.e. Linear Programming is in the class P of polynomially solvable problems.
- Roughly speaking this means that, if the LP-instance contains m constraints and n variables, the instance can be solved (by some algorithm) in a number of elementary steps not greater than $km^q n^p$, with k, q, p given constants.
- If each elementary step takes at most execution time T, then the time to solve the instance is bounded by $Tkm^q n^p$.

How difficult is it to solve an LP?

- Any instance of LP can be solved efficiently, i.e. Linear Programming is in the class P of polynomially solvable problems.
- Roughly speaking this means that, if the LP-instance contains m constraints and n variables, the instance can be solved (by some algorithm) in a number of elementary steps not greater than $km^q n^p$, with k, q, p given constants.
- If each elementary step takes at most execution time T, then the time to solve the instance is bounded by $Tkm^q n^p$.
- $Tkm^q n^p$ is polynomial in m and n: the exponents q, p do not depend on the instance (neither the other constants T, k).

How difficult is it to solve an LP?

- Any instance of LP can be solved efficiently, i.e. Linear Programming is in the class P of polynomially solvable problems.
- Roughly speaking this means that, if the LP-instance contains m constraints and n variables, the instance can be solved (by some algorithm) in a number of elementary steps not greater than $km^q n^p$, with k, q, p given constants.
- If each elementary step takes at most execution time T, then the time to solve the instance is bounded by $Tkm^q n^p$.
- $Tkm^q n^p$ is polynomial in m and n: the exponents q, p do not depend on the instance (neither the other constants T, k).
- m, n are the dimensions of the instance. We use the "big-$\mathcal{O}$" notation $\mathcal{O}(m^q n^p)$ to denote the complexity of this algorithm.

How difficult is it to solve an LP?

- Any instance of LP can be solved efficiently, i.e. Linear Programming is in the class P of polynomially solvable problems.
- Roughly speaking this means that, if the LP-instance contains m constraints and n variables, the instance can be solved (by some algorithm) in a number of elementary steps not greater than $km^q n^p$, with k, q, p given constants.
- If each elementary step takes at most execution time T, then the time to solve the instance is bounded by $Tkm^q n^p$.
- $Tkm^q n^p$ is polynomial in m and n: the exponents q, p do not depend on the instance (neither the other constants T, k).
- m, n are the dimensions of the instance. We use the "big-$\mathcal{O}$" notation $\mathcal{O}(m^q n^p)$ to denote the complexity of this algorithm.
- Unfortunately this is not true for MILP and 0,1-LP: the time to solve a MILP instance may grow exponentially with its dimensions.

How difficult is it to solve an LP?

- Any instance of LP can be solved efficiently, i.e. Linear Programming is in the class P of polynomially solvable problems.
- Roughly speaking this means that, if the LP-instance contains m constraints and n variables, the instance can be solved (by some algorithm) in a number of elementary steps not greater than $km^q n^p$, with k, q, p given constants.
- If each elementary step takes at most execution time T, then the time to solve the instance is bounded by $Tkm^q n^p$.
- $Tkm^q n^p$ is polynomial in m and n: the exponents q, p do not depend on the instance (neither the other constants T, k).
- m, n are the dimensions of the instance. We use the "big-$\mathcal{O}$" notation $\mathcal{O}(m^q n^p)$ to denote the complexity of this algorithm.
- Unfortunately this is not true for MILP and 0,1-LP: the time to solve a MILP instance may grow exponentially with its dimensions.
- Several commercial and open software available to solve MILPs.

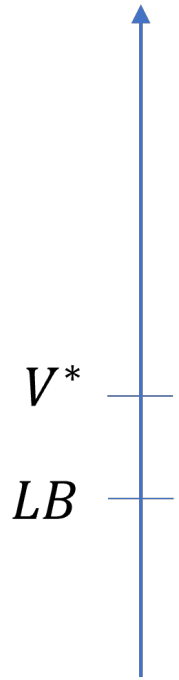
Feasible solution and lower bound

$$V^* = \max c^T x + d^T y$$

$$Ax + Dy \leq b$$

$$x \in \mathbb{R}^p, y \in Z^q, A \in \mathbb{R}^{m \times p}, D \in \mathbb{R}^{m \times q}, b \in \mathbb{R}^m.$$

- Suppose (x^*, y^*) is an optimal solution to MILP (5).
- The optimal solution value will be $V^* = c^T x^* + d^T y^*$
- Any feasible solution $(\bar{x}, \bar{y})$ to the MILP (5) provides lower bound $LB = c^T \bar{x} + d^T \bar{y} \leq V^*$

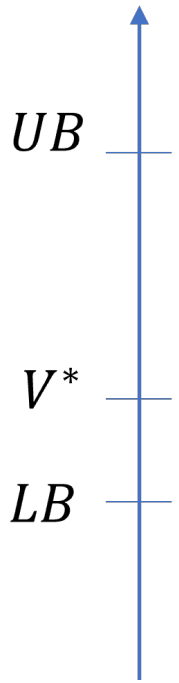


Linear Relaxation

$$V^* = \max c^T x + d^T y$$

$$Ax + Dy \leq b$$

$$x \in \mathbb{R}^p, y \in Z^q, A \in \mathbb{R}^{m \times p}, D \in \mathbb{R}^{m \times q}, b \in \mathbb{R}^m.$$



- We drop the integral stipulation on the integer variables a MILP is turned into an LP, that is we let $y \in \mathbb{R}^q$
- The new problem is called *linear relaxation*. Let $\tilde{x}, \tilde{y}$ be the optimal solution to the linear relaxation.
- $\tilde{y}$ may have fractional components and $\tilde{x}, \tilde{y}$ is not feasible for original MILP.
- $UB = c^t \tilde{x} + d^T \tilde{y}$ is an upper bound on the optimal value $V*$ to the original MILP.

OBS: For minimization problems, a feasible solution gives an upper bound UB and a relaxation gives a lower bound LB.

Example of linear relaxation: project selection

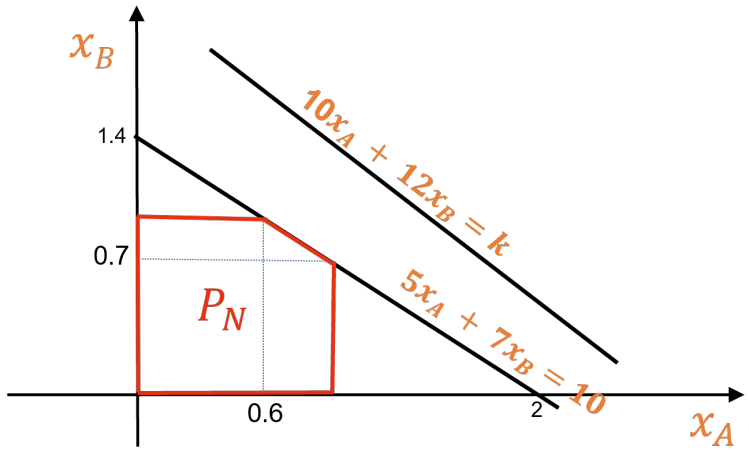
- Consider the project selection formulation and its relaxation

$$\begin{aligned} \max \quad & 10x_A + 12x_B \\ s.t. \quad & 5x_A + 7x_B \leq 10 \\ & x_A, x_B \in \{0,1\} \end{aligned}$$

$$\begin{aligned} \max \quad & 10x_A + 12x_B \\ s.t. \quad & 5x_A + 7x_B \leq 10 \\ & 0 \leq x_A \leq 1, 0 \leq x_B \leq 1 \end{aligned}$$

(the binary stipulation is replaced by box constraints).

- The feasible region P_N of the relaxation is a polyhedron
- The optimal solution to the relaxation is $x_A = 1, x_B = 0.7$ and $UB = 18.4 > V^* = 12$.
- The original feasible solution set is $P_N \cap \{0,1\}^n$



Combinatorial Optimization

- We are given a finite ground set E and a weight function $w : E \to \mathbb{R}$ (i.e. $w \in \mathbb{R}^E$)
- Feasible solution set $\mathcal{F} = \{F_1, \ldots, F_m\}$, with $F_i \subseteq E$, $i \in [m]$.
- Let $w(F) = \sum_{e \in F} w_e$.

Definition (Combinatorial Optimization (CO))

The combinatorial optimization problem with linear objective function is the problem:

$$\max\{w(F) : F \in \mathcal{F}\}, \tag{6}$$

- Example (project selection): Ground set (projects) $E = \{A, B\}$. $w_A = 10$, $w_B = 12$.
- Feasible solutions are the subsets $\mathcal{F} = \{\emptyset, \{A\}, \{B\}\}$.

Combinatorial Optimization and 0,1-LP

- For $F \in \mathcal{F}$, let $\chi^F \in \{0,1\}^E$ be its incidence vector, i.e. $\chi^F_e = 1$ if and only if $e \in F$.
- Let $S_{\mathcal{F}} = \{\chi^F : F \in \mathcal{F}\} \subseteq \{0,1\}^E$ be the set of feasible solutions.
- The combinatorial optimization problem (7) can be rewritten as

$$\min\{w^T x : x \in S_{\mathcal{F}}\}$$

- Example: project section. Feasible solutions are the subsets $\mathcal{F} = \{\emptyset, \{A\}, \{B\}\}$.
- Feasible 0,1-points: $S_{\mathcal{F}} = \{(0,0)^T, (0,1)^T, (1,0)^T\}$

Combinatorial Optimization: formulation

- Consider a combinatorial optimization problem

$$\min\{w^T x : x \in S_{\mathcal{F}} \subseteq \{0,1\}^E\}$$

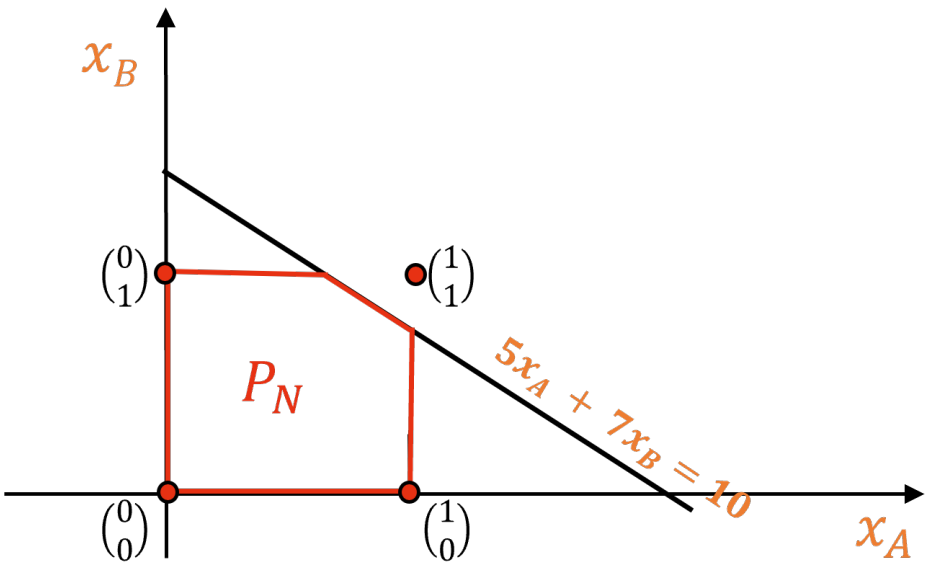
- Let $P_{\mathcal{F}}$ be a polyhedron such that $P_{\mathcal{F}} \cap \{0,1\}^E = S_{\mathcal{F}}$.
- The CO can be rewritten as a 0,1-LP:

$$\min\{w^T x : x \in P_{\mathcal{F}}, x \in \{0,1\}^E\} \tag{7}$$

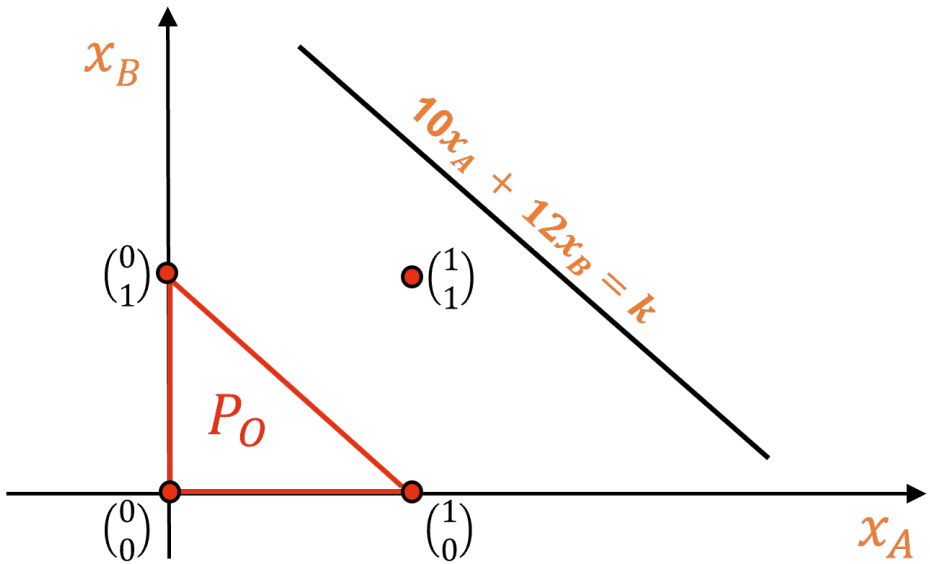
- $P_{\mathcal{F}}$ is called a *formulation* for $S_{\mathcal{F}}$.
- One can show that there exists an infinite number of such polyhedrons $P_{\mathcal{F}}$.

Example of formulation: project selection

- Set of projects $E = \{A, B\}$. Project A cost 5 MNOK, project B cost 7 MNOK, total budget 10 MNOK.
- Feasible solutions are the subsets $\mathcal{F} = \{\emptyset, \{A\}, \{B\}\}$.
- Feasible 0,1-points: $S_{\mathcal{F}} = \{(0,0)^T, (0,1)^T, (1,0)^T\}$
- Natural formulation: directly derived from the original constraint
- Optimal formulation: smallest possible formulation, the extreme points are exactly the 0,1-points in $S_{\mathcal{F}}$.



Natural formulation:



Optimal formulation

Knapsack Problems

The most classical and basic combinatorial optimization problem

Exercise

You need to fill your knapsack with some items chosen from a set $I = \{1, \ldots, n\}$. Each item has a value $a_i \geq 0$ and a weight $b_i \geq 0$. The maximum weight you are allowed to fill in your knapsack is K. You want to fill the knapsack with a selection of items with maximum value.

For each item, you introduce a binary variable $x_i \in \{0, 1\}$ which is 1 if and only if item $i \in I$ is chosen. We want to maximize the value of the chosen items.

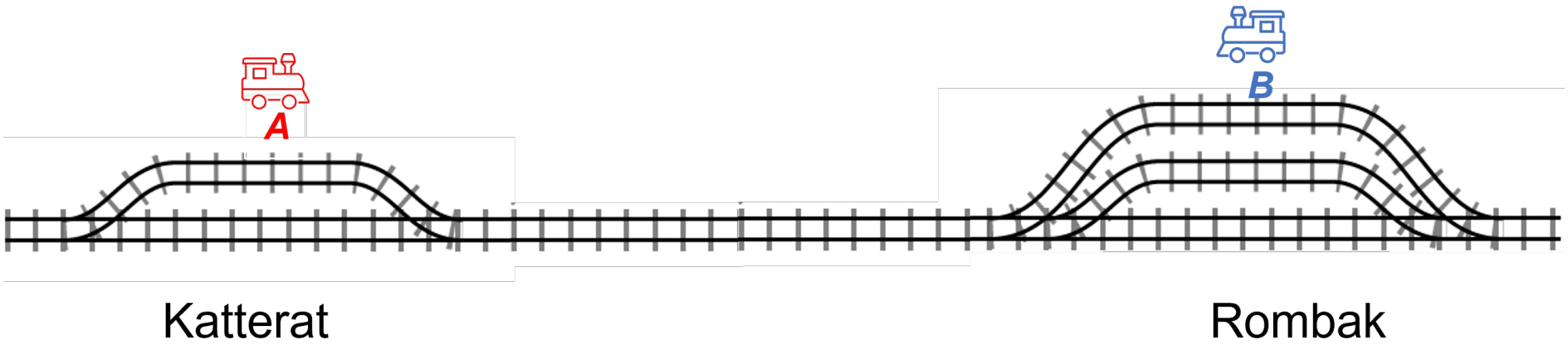
Objective:

$$\max a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

Constraints. You want to satisfy the capacity of the knapsack:

$$b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \leq K$$

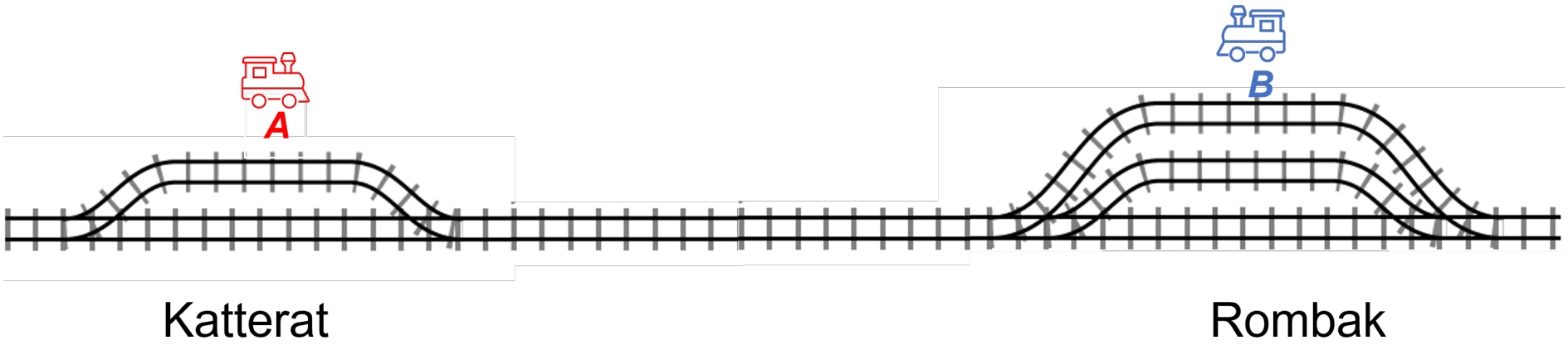
Scheduling Problems



Exercise

Freight train *A* is in Katterat station, where it will stay for (at least) another 10 minutes for refuelling. Rombak station is its final destination. Passenger train *B* is in Rombak station where it will stay for (at least) another 15 minutes to wait for passengers. Katterat is its final destination. The single track between Katterat and Rombak can only accommodate one train at a time. Train *A* takes 30 minutes to travel from Katterat to Rombak, whereas train *B* takes only 15 minutes. As soon as refuelling is completed, freight train *A* leaves Katterat. Write an LP which computes a schedule for both trains which minimizes the time when both trains are at their final destinations (*makespan*).

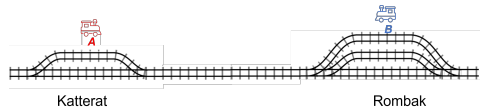
Scheduling Trains continues



Exercise

Actually, it is 9:00 pm and there is a timetable. Train B is expected in Katterat at 9:30, while train *A* in Rombak at 11:00. A dispatcher can now decide whether train *A* waits for train *B* in Katterat, or train *B* waits for train *A* in Rombak. Write a 0,1-LP which computes a feasible schedule which minimizes the sum of the delays.

Scheduling Trains continues



Exercise

Actually, it is 9:00 pm and there is a timetable. Train B is expected in Katterat at 9:30, while train *A* in Rombak at 11:00. A dispatcher can now decide whether train *A* waits for train *B* in Katterat, or train *B* waits for train *A* in Rombak. Write a 0,1-LP which computes a feasible schedule which minimizes the sum of the delays.

Variables $s_A^K, s_A^R, s_B^R, s_A^K \in \mathbb{R}$ represent, respectively, the time when train *A* leaves Katterat, when *A* arrives in Rombak, when *B* leaves Rombak and when *B* arrives in Katterat. δ_A, δ_B: delay of train *A* and *B*, respectively. Binary variable $x_{AB} \in \{0, 1\}$ which is 1 if train *A* traverses the single track before *B* and 0 otherwise. *M* is a very large constant.

min	$\delta_A + \delta_B$	Minimize sum of delays
	$s_A^K \geq 10$	*A* leaves Katterat in at least 10 minutes
	$s_B^R \geq 15$	*B* leaves Rombak in at least 15 minutes
	$s_A^R \geq s_A^K + 30$	running time of train *A* through track
	$s_B^K \geq s_B^R + 15$	running time of train *B* through track
	$\delta_A \geq s_A^R - 120$	minimum delay of train *A*
	$\delta_B \geq s_B^K - 30$	minimum delay of train *B*
	$\delta_A, \delta_B \geq 0$	non-negativity of delays
	$s_B^R \geq s_A^R - M(1 - x_{AB})$	*B* leaves Rombak after *A* arrives in Rombak
	$s_A^K \geq s_B^K - Mx_{AB}$	*A* leaves Katterat after *B* arrives in Katterat

The last two constraints implement the big-*M* trick.

(*i*) When $x_{AB} = 1$ then the first constraint reduces $s_B^R \geq s_A^R$, imposing that *B* leaves Rombak after *A* arrives, which corresponds to *A* traversing the single track first; the second constraint reduces to $s_A^K \geq s_B^K - M$ which is always satisfied and thus redundant.

(*ii*) When *B* traverses the single track first, then $x_{AB} = 0$: the first constraint reduces $s_B^R \geq s_A^R - M$, which is always satisfied and thus redundant, whereas the second constraint reduces to $s_A^K \geq s_B^K$ imposing that *A* leaves Katterat after *B* arrives.

RNA folding: a crude case

Gusfield, D., 2019. Integer linear programming in computational and systems biology: an entry-level text and course. Cambridge University Press.

Exercise

An RNA molecule may be viewed as a string from the alphabet $\{A, C, U, G\}$ (the nucleotides). *Complementary* nucleotides can bond, namely A with U and C with G. If we draw the string on a circle, and the bonds as lines joining the involved pair of nucleotides, the lines should never cross. Given an RNA string with n nucleotides, we want to maximize the number of bonds, which correspond to the minimum free energy.

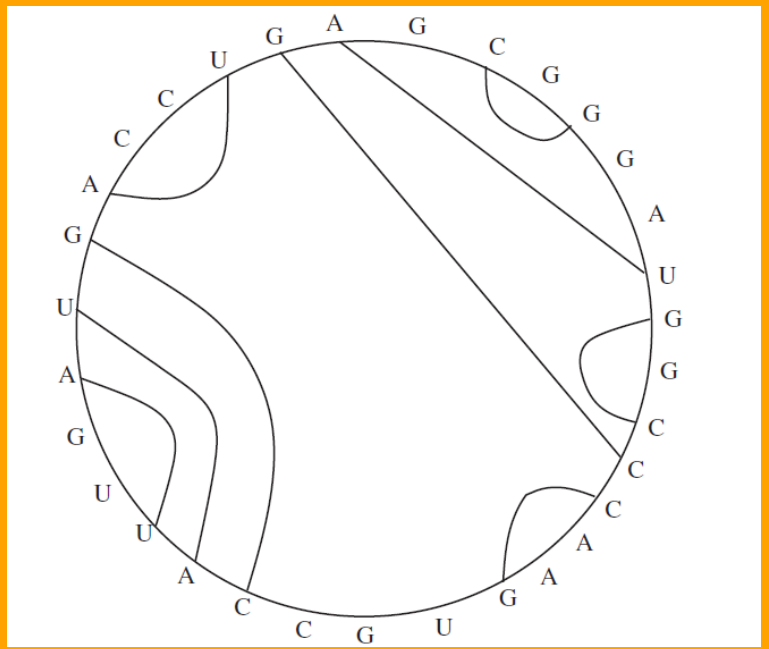


Figure from Gusfield (2019).

RNA folding: a crude case

Gusfield, D., 2019. Integer linear programming in computational and systems biology: an entry-level text and course. Cambridge University Press.

Exercise

An RNA molecule may be viewed as a string from the alphabet $\{A, C, U, G\}$ (the nucleotides). *Complementary* nucleotides can bond, namely A with U and C with G. If we draw the string on a circle, and the bonds as lines joining the involved pair of nucleotides, the lines should never cross. Given an RNA string with n nucleotides, we want to maximize the number of bonds, which correspond to the minimum free energy.

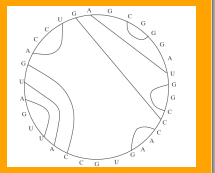


Figure from Gusfield (2019).

Let $N = \{1, \ldots, n\}$ be the set of the nucleotids, with $i < j$ if nucleotid j comes after i in the string. For $i, j \in N$, $i < j$,we introduce binary variable x_{ij} which is 1 if and only if i and j are bonded. We want to maximize the number of bonds and thus the objective function becomes

$$\max \sum_{\{i,j\} \subset N} x_{ij}$$

As for the constraints, first notice that a nucleotide can only be paired once.

$$\sum \sum_{j \in N \setminus \{i\}} x_{ij} \leq 1, \qquad i \in N$$

Next, we need to impede that pairings create crossings. This is equivalent to that if $i < l$ are paired and $j < m$ are paired, we do not have that $i < j < l < m$.

$$x_{il} + x_{jm} \leq 1, \qquad i, j, l, m \in N, i < j < l < m$$

Algorithms for combinatorial optimization and 0,1 LP

Given a base set $E = \{1, \ldots, m\}$, and solutions $\mathcal{F} = \{F_1, \ldots, F_m\}$, with $F_i \subseteq E$, $i \in [m]$ we look at algorithms to solve (CO) problem

$$\max\{w(F) : F \in \mathcal{F}\}$$

or, equivalently

$$\max\{w^T x : x \in P_{\mathcal{F}}, x \in \{0,1\}^E\}$$

- *Heuristic* algorithms: search for ("good") feasible solutions $\bar{F} \in \mathcal{F}$ (or $\bar{x} \in S = P_{\mathcal{F}} \cap \left\{x \in \{0,1\}^E\right\}$).
- *Exact* algorithms: find the optimal solution to (CO) or prove that no solution exists.

The simplest heuristic: greedy algorithm

- The greedy algorithm builds a solution $\bar{F} \in \mathcal{F}$ by adding iteratively one element of E at a time.
- There exist several versions. Here we assume $w \in \mathbb{R}^+$ and that $\mathcal{F}$ is closed under taking subsets, i.e. if $\bar{F} \in \mathcal{F}$, then $F \in \mathcal{F}$ for any $F \subseteq \bar{F}$

The Greedy algorithm

1. Start with $F^0 = \emptyset$. Set $i = 1$.
2. Find $e^i \in E \setminus F^{i-1}$ of maximum weight w^i such that $F^i = F^{i-1} \cup \{e^i\} \in \mathcal{F}$.
3. If no such element exists STOP. Return F^{i-1}.
4. Otherwise let $i = i + 1$ and go to Step 2.

Greedy exercises

Exercise

Find a feasible solution to the project selection example

Exercise

Find a feasible solution to the knapsack problem:

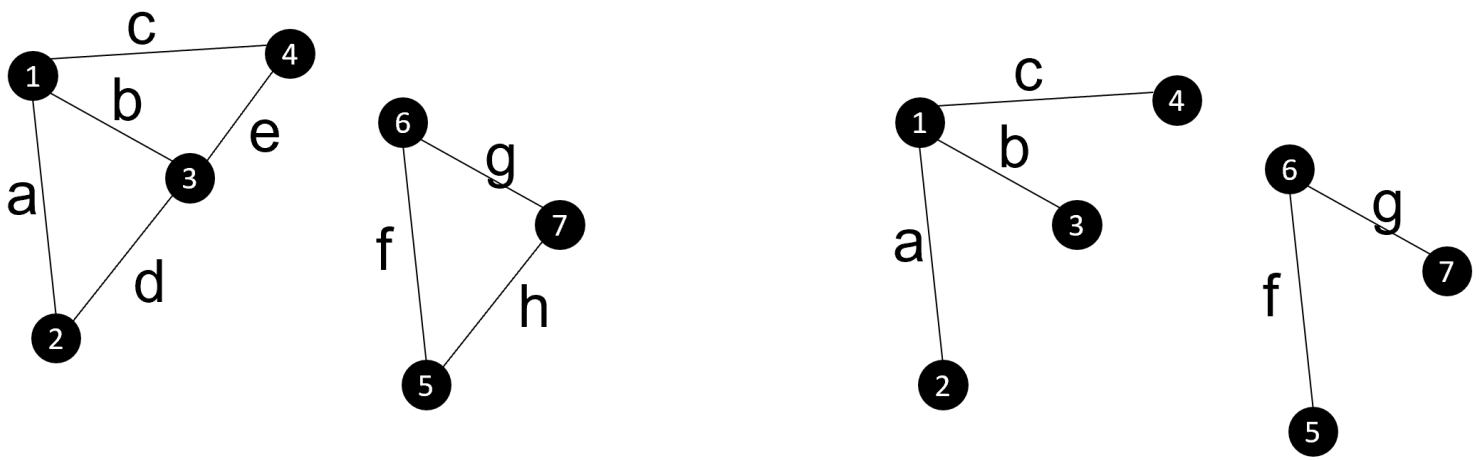
$$\begin{aligned} \max \quad & 10x_1 + 6x_2 + 5x_3 \\ & 6x_1 + 5x_2 + 5x_3 \leq 10 \\ & x_1, x_2, x_3 \in \{0, 1\} \end{aligned}$$

Greedy exercises

Exercise

Given an undirected graph $G = (V, E)$, a forest $F \subseteq E$ is a set of edges which do not contain cycles. Let $w : E \to \mathbb{R}_+$ a weight function. Apply the greedy algorithm to find a forest of large weight.

This is actually known as Prim's algorithm, and it can be shown to return the optimal forest (i.e. the forest of largest weight).



In the right part of the figure you see a (maximal) forest of the graph $G = (V, E)$ on the left, with $V = \{1, 2, 3, 4, 5, 6, 7\}$, $E = \{a, b, c, d, e, f, g, h\}$. Now, let $w(a) = 3, w(b) = 5, w(c) = 2, w(d) = 4, w(e) = 6, w(f) = 4, w(g) = 3, w(h) = 5$. Apply the greedy algorithm.

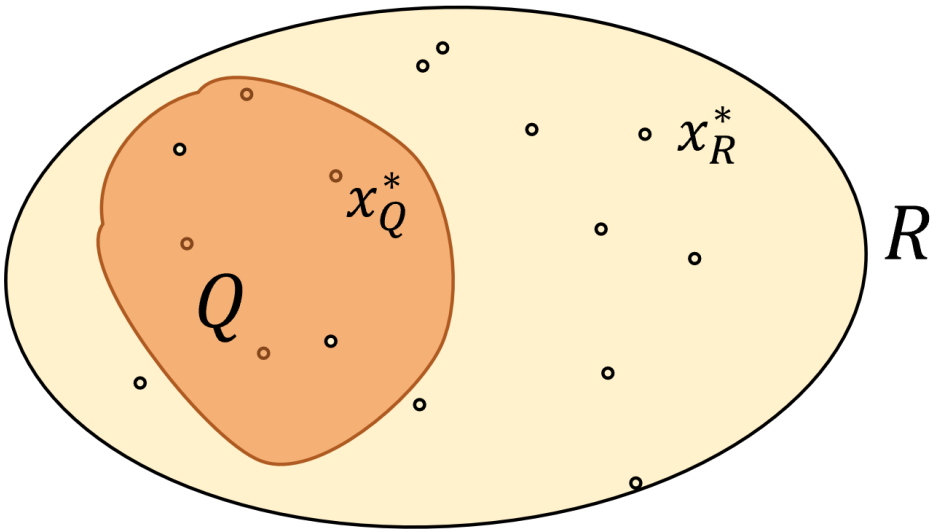
Relaxations

Definition

Given an optimization problem (Q) $\{\max w(x) : x \in S\}$ we call relaxation of Q any problem (R) $\{\max w(x) : x \in T\}$ such that $S \subseteq T$.

Lemma

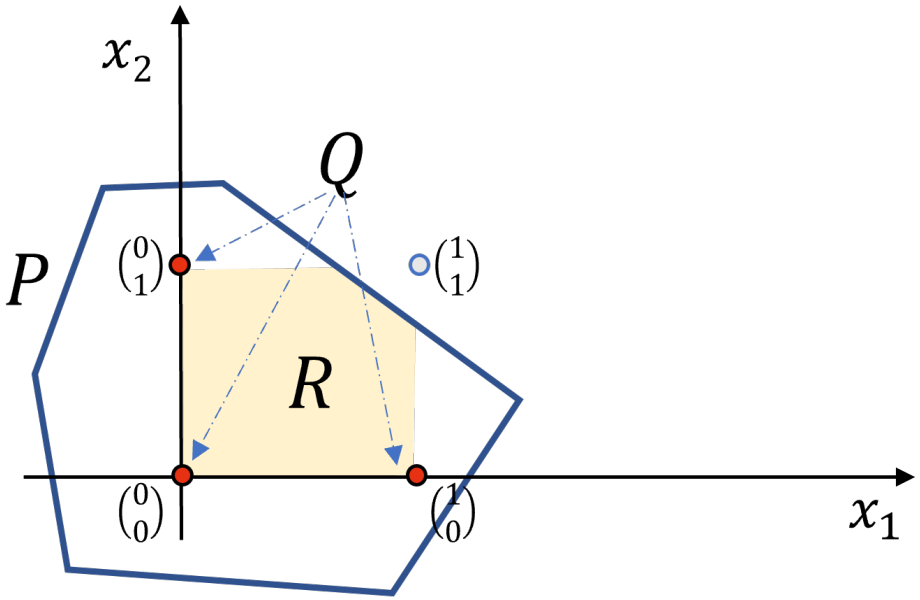
If (R) is infeasible (i.e. $T = \emptyset$), then (Q) is infeasible. Suppose (R) admits an optimal solution x_R^*, and (Q) admits an optimal solution x_Q^*. Then $UB = w(x_R^*) \geq w(x_Q^*)$. If $x_R^* \in S$, then x_R^* is optimal for (Q).



Relaxations

Suppose (Q) is a 0,1-LP, i.e. $S = P \cap \{0,1\}^n$, where P is a polyhedron.

- Normally (R) is the standard linear relaxation, that is $\{\max w^T x : x \in P \cap [0,1]^n\}$
- If $\overline{x}_Q \in \{0,1\}^n$ is feasible for (Q) (i.e. $\overline{x}_Q \in P \cap \{0,1\}^n$), then $w^T \overline{x}_Q \leq w^T x_Q^*$ is a lower bound for (Q).



Branch & bound: termination conditions

- Remark: Solving the linear relaxation is ”easy”.
- What if we solve (R) instead of (Q)?

(i) (R) is infeasible $\rightarrow$ (Q) infeasible

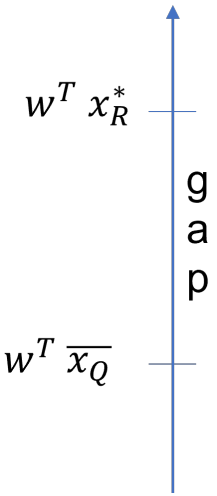
(ii) x_R^* optimal for $(R) \rightarrow w^T x_R^*$ is an upper bound for (Q) .

(iii) $\overline{x}_Q$ feasible for $(Q) \rightarrow w^T \overline{x}_Q$ is a lower bound for (Q).

(iv) x_R^* feasible for $(Q) \rightarrow x_R^*$ optimal for (Q)

(v) $(w^T x_R^* - w^T \overline{x}_Q)/w^T x_R^*$ is a quality measure for $\overline{x}_Q$ (the *gap*)

(vi) $gap = 0 \rightarrow w^T \overline{x}_Q$ optimal for (Q)

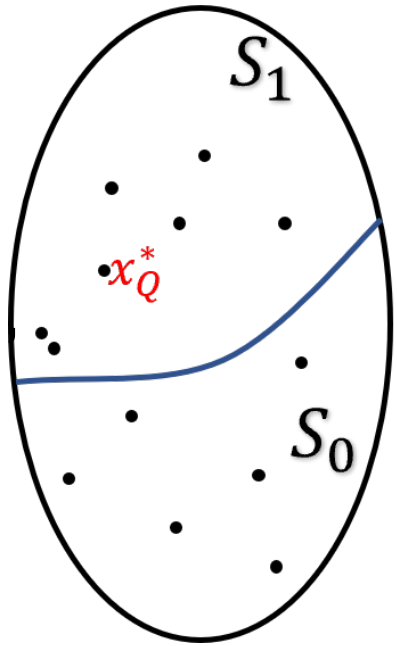


- If we solve (R) and either condition (i) or (iv) or (vi) is satisfied then we have solved (Q)!
- What to do in the other cases? We branch.

Branch & bound: branching

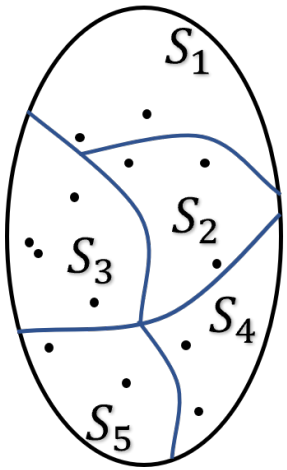
- We partition the feasible set S into S_0, S_1.
- $\{\max w^T x : x \in S = S_0 \cup S_1\} =$
  $\max\left\{\{\max w^T x : x \in S_0\}, \{\max w^T x : x \in S_1\}\right\}$.
- Typically $S_0 = \{x \in S : x_i = 0\}$, $S_1 = \{x \in S : x_i = 1\}$, where i is chosen by some criteria.
- We thus replace (Q) with two new problems:
  - (Q_0) $\{\max w^T x : x \in S_0\}$ and
  - (Q_1) $\{\max w^T x : x \in S_1\}$.
- We can solve the relaxations
  - $(R_0) : \{\max w^T x : x \in P \cap [0,1]^n \cap x_i = 0\}$
  - $(R_1) : \{\max w^T x : x \in P \cap [0,1]^n \cap x_i = 1\}$

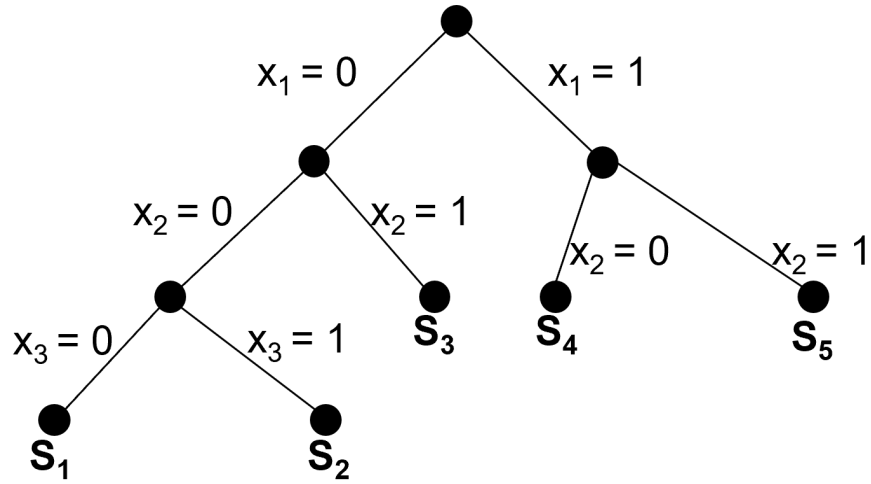
  and again check termination conditions (i) (infeasibility) or (iv) (feasible relaxation) or (vi) (Null Gap).
- Note that both problems (Q_0) and (Q_1) are smaller than (Q), and both can be partitioned recursively.



Branch & Bound: enumeration tree

- Partition can be created recursively, by fixing a variable at a time.





- Recursive partitions are represented by *enumeration trees*.
- Each node v is associated with a subset S_v of S and a variable x_v.
- Each non-leaf node v has two children.
- The left and right child of v are associated with the solutions $S_v^0 = \{x \in S_v : x_v = 0\}$ and $S_v^1 = \{x \in S_v : x_v = 1\}$, respectively.

Branch & bound: the algorithm

We solve the 0,1-LP $(Q) : \{\max w^T x : x \in S = P \cap \{0,1\}^n\}$.

1. Let $\mathcal{L} = \{(Q)\}$. If a feasible solution (the *incumbent*) $x^I \in S$ is at hand, let $LB = w^T x^I$, otherwise $LB = -\infty$.
2. If $\mathcal{L} = \emptyset$, STOP.
3. Choose a problem $(\overline{Q}) \in \mathcal{L}$ and let $\mathcal{L} = \mathcal{L} \setminus \{(\overline{Q})\}$. Let $\overline{S}$ be the feasible region of $(\overline{Q})$ and $(\overline{R})$ be the linear relaxation of $(\overline{Q})$.
4. Solve $\overline{R}$. If $\overline{R}$ is infeasible GOTO Step 2.
5. Otherwise let $\overline{x}^R$ be the optimal solution to $\overline{R}$. Let $UB = w^T \overline{x}^R$
6. If $UB \leq LB$ GOTO Step 2.
7. Otherwise if $\overline{x}^R \in \{0,1\}^n$, then $\overline{x}^R \in S$ and let $x^I = \overline{x}^R$, $LB = w^T x^I$ and GOTO Step 2.
8. Otherwise choose i such that $x_i^R \in (0,1)$ (x_i^R *fractional component* of x^R) and let $(\overline{Q}_0)$ $\{\max w^T x : x \in \overline{S} \cap x_i = 0\}$ and $(\overline{Q}_1)$ $\{\max w^T x : x \in \overline{S} \cap x_i = 1\}$.
9. Let $\mathcal{L} = \mathcal{L} \cup \{(\overline{Q}_0), (\overline{Q}_1)\}$ and GOTO Step 2.

Branch & bound: comments on algorithm

- At each iteration we pick one problem from $\mathcal{L}$ (Step 3) and possibly replace it by 2 new problems (Step 9). The size of $\mathcal{L}$ can thus grow as $O(2^n)$.

Branch & bound: comments on algorithm

- At each iteration we pick one problem from $\mathcal{L}$ (Step 3) and possibly replace it by 2 new problems (Step 9). The size of $\mathcal{L}$ can thus grow as $O(2^n)$.
- A problem in $\mathcal{L}$ is not replaced if one of termination conditions (i) or (iv) or (vi) (see slide "Termination conditions") is satisfied

Branch & bound: comments on algorithm

- At each iteration we pick one problem from $\mathcal{L}$ (Step 3) and possibly replace it by 2 new problems (Step 9). The size of $\mathcal{L}$ can thus grow as $O(2^n)$.
- A problem in $\mathcal{L}$ is not replaced if one of termination conditions (i) or (iv) or (vi) (see slide "Termination conditions") is satisfied
  - Condition (i) is tested at Step 4 (*Infeasibility Test*).

Branch & bound: comments on algorithm

- At each iteration we pick one problem from $\mathcal{L}$ (Step 3) and possibly replace it by 2 new problems (Step 9). The size of $\mathcal{L}$ can thus grow as $O(2^n)$.
- A problem in $\mathcal{L}$ is not replaced if one of termination conditions (i) or (iv) or (vi) (see slide "Termination conditions") is satisfied
  - Condition (i) is tested at Step 4 (*Infeasibility Test*).
  - Condition (iv) is tested at Step 6 (*Gap Test*).

Branch & bound: comments on algorithm

- At each iteration we pick one problem from $\mathcal{L}$ (Step 3) and possibly replace it by 2 new problems (Step 9). The size of $\mathcal{L}$ can thus grow as $O(2^n)$.
- A problem in $\mathcal{L}$ is not replaced if one of termination conditions (i) or (iv) or (vi) (see slide "Termination conditions") is satisfied
  - Condition (i) is tested at Step 4 (*Infeasibility Test*).
  - Condition (iv) is tested at Step 6 (*Gap Test*).
  - Condition (vi) is tested at Step 7 (*Integrality Test*).

Branch & bound: comments on algorithm

- At each iteration we pick one problem from $\mathcal{L}$ (Step 3) and possibly replace it by 2 new problems (Step 9). The size of $\mathcal{L}$ can thus grow as $O(2^n)$.
- A problem in $\mathcal{L}$ is not replaced if one of termination conditions (i) or (iv) or (vi) (see slide "Termination conditions") is satisfied
  - Condition (i) is tested at Step 4 (*Infeasibility Test*).
  - Condition (iv) is tested at Step 6 (*Gap Test*).
  - Condition (vi) is tested at Step 7 (*Integrality Test*).
- The list of generated problems can be represented as a rooted (partial) binary tree (enumeration tree). Each node correspond to a problem $(\overline{Q}) \in \mathcal{L}$ and each non-leaf node $(\overline{Q}^N)$ has two children, namely the two problems generated from $(\overline{Q}^N)$.

Branch & bound: comments on algorithm

- At each iteration we pick one problem from $\mathcal{L}$ (Step 3) and possibly replace it by 2 new problems (Step 9). The size of $\mathcal{L}$ can thus grow as $O(2^n)$.
- A problem in $\mathcal{L}$ is not replaced if one of termination conditions (i) or (iv) or (vi) (see slide "Termination conditions") is satisfied
  - Condition (i) is tested at Step 4 (*Infeasibility Test*).
  - Condition (iv) is tested at Step 6 (*Gap Test*).
  - Condition (vi) is tested at Step 7 (*Integrality Test*).
- The list of generated problems can be represented as a rooted (partial) binary tree (enumeration tree). Each node correspond to a problem $(\overline{Q}) \in \mathcal{L}$ and each non-leaf node $(\overline{Q}^N)$ has two children, namely the two problems generated from $(\overline{Q}^N)$.
- By choosing a "fractional" variable x_i at Step 8 we ensure that the method converge, since x_i has not been fixed in an ancestor of $\overline{Q}$.

Branch & bound: comments on algorithm

- At each iteration we pick one problem from $\mathcal{L}$ (Step 3) and possibly replace it by 2 new problems (Step 9). The size of $\mathcal{L}$ can thus grow as $O(2^n)$.
- A problem in $\mathcal{L}$ is not replaced if one of termination conditions (i) or (iv) or (vi) (see slide "Termination conditions") is satisfied
  - Condition (i) is tested at Step 4 (*Infeasibility Test*).
  - Condition (iv) is tested at Step 6 (*Gap Test*).
  - Condition (vi) is tested at Step 7 (*Integrality Test*).
- The list of generated problems can be represented as a rooted (partial) binary tree (enumeration tree). Each node correspond to a problem $(\overline{Q}) \in \mathcal{L}$ and each non-leaf node $(\overline{Q}^N)$ has two children, namely the two problems generated from $(\overline{Q}^N)$.
- By choosing a "fractional" variable x_i at Step 8 we ensure that the method converge, since x_i has not been fixed in an ancestor of $\overline{Q}$.
- If at termination $LB = -\infty$, then the initial problem (Q) is infeasible.

Branch & bound: comments on algorithm

- At each iteration we pick one problem from $\mathcal{L}$ (Step 3) and possibly replace it by 2 new problems (Step 9). The size of $\mathcal{L}$ can thus grow as $O(2^n)$.
- A problem in $\mathcal{L}$ is not replaced if one of termination conditions (i) or (iv) or (vi) (see slide "Termination conditions") is satisfied
  - Condition (i) is tested at Step 4 (*Infeasibility Test*).
  - Condition (iv) is tested at Step 6 (*Gap Test*).
  - Condition (vi) is tested at Step 7 (*Integrality Test*).
- The list of generated problems can be represented as a rooted (partial) binary tree (enumeration tree). Each node correspond to a problem $(\overline{Q}) \in \mathcal{L}$ and each non-leaf node $(\overline{Q}^N)$ has two children, namely the two problems generated from $(\overline{Q}^N)$.
- By choosing a "fractional" variable x_i at Step 8 we ensure that the method converge, since x_i has not been fixed in an ancestor of $\overline{Q}$.
- If at termination $LB = -\infty$, then the initial problem (Q) is infeasible.
- Question: how to choose the next problem to solve from $\mathcal{L}$? How to chose the branching variable?

Exercises

Exercise

Solve the project selection example by B&B using the graphical method to solve the relaxation

Exercise

Solve by B&B the knapsack problem:

$$\begin{array}{ll} \max & 10x_1 + 6x_2 + 5x_3 \\ & 6x_1 + 5x_2 + 5x_3 \leq 10 \qquad (Q^0) \\ & x_1, x_2, x_3 \in \{0, 1\} \end{array}$$

Hint: assume the elements of the knapsack are ordered for non-increasing ratio benefit/weight, i.e. $a_1/b_1 = r_1 \geq r_2 \geq \cdots \geq r_n = a_n/b_n$. Then, an optimal solution to the relaxation is the following

1. Let k be the maximum index such that $\sum_{i=1}^{k} b_i < K$ (K is the knapsack capacity).
2. Optimal fractional solution: $x_1 = \cdots = x_k = 1$, $x_{k+1} = (K - \sum_{i=1}^{k} b_i)/b_{k+1}$, $x_{k+2} = \cdots = x_n = 0$

Solution to the knapsack

The greedy algorithm produces an initial feasible solution $x^I = (1, 0, 0)^T$, and its value is $LB = 10$.

Iteration 1.

We extract the original problem (Q^0) from $\mathcal{L}$. The variables of Q^0 are already ordered according to non-increasing values of the ratio benefit/weight:

$$\frac{a_1}{b_1} = \frac{10}{6} \geq \frac{a_2}{b_2} = \frac{6}{5} \geq \frac{a_3}{b_3} = \frac{5}{5}.$$

So, the solution to the linear relaxation R^0 of Q^0 is $x_1^0 = 1$, $x_2^0 = 4/5$, $x_3^0 = 0$ and the value $UB^0 = 14.8 > LB$. None of the termination condition is encountered. We branch: the relaxation is feasible, the gap is strictly positive and the solution has a fractional component.

Choice of the branching variable. A typical choice is a "most fractional" variable, i.e. x_i closest to $1/2$. We create two problems Q^1, Q^2 from current one (Q^0) by fixing $x_2 = 0$ and $x_2 = 1$, respectively.

$Q^1 = \{\max 10x_1 + 5x_3 : 6x_1 + 5x_3 \leq 10, x_2 = 0, x_1, x_3 \in \{0, 1\}\}$

$Q^2 = \{6 + \max 10x_1 + 5x_3 : 6x_1 + 5x_3 \leq 5, x_2 = 1, x_1, x_3 \in \{0, 1\}\}$

$\mathcal{L} = \{Q^1, Q^2\}$

Note that both Q^1 and Q^2 are knapsack problems, and we can compute the relaxation as before.

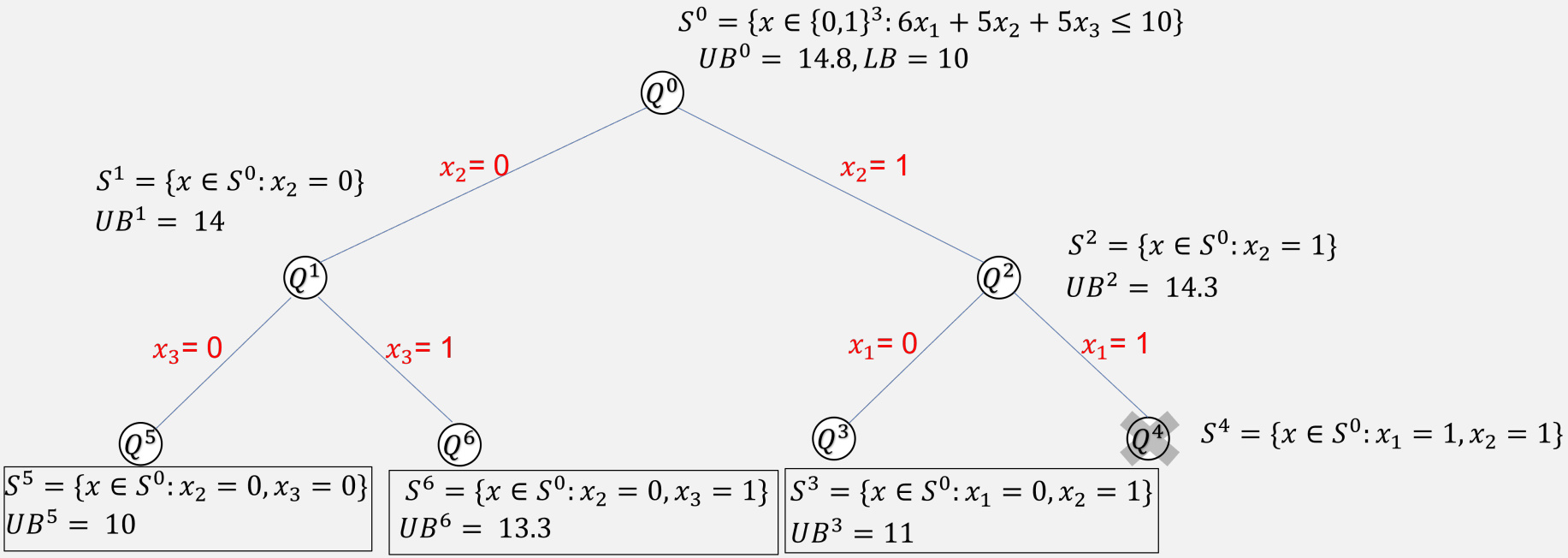
Solution to the knapsack (II)

Iteration 2.
- We need to pick up a problem from $\mathcal{L} = \{Q^1, Q^2\}$. The *best bound* criterium amounts to choosing the problem with largest value of the relaxation.
- Solving the relaxation R^1 of Q^1 yields $x_1^1 = 1, x_2^1 = 0, x_3^1 = 4/5$ with value $V^1 = 14$.
- Solving the relaxation R^2 of Q^2 yields $x_1^2 = 5/6, x_2^2 = 1, x_3^2 = 0$ with value $V^2 = 14.3$.
- We extract Q^2 from $\mathcal{L}$ (which becomes .
- Since R^2 is feasible, x^2 is fractional, and $UB^1 = 14.3 > 10 = LB$ we need to branch.
- We select a most fractional variable x^1 and fix it to 0 and 1, generating Q^3, Q^4.
$Q^3 = \{6 + \max 5x_3 : 5x_3 \leq 10, x_1 = 0, x_2 = 1, x_3 \in \{0, 1\}\}$
$Q^4 = \{16 + \max 5x_3 : 5x_3 \leq -1, x_1 = 1, x_2 = 1, x_3 \in \{0, 1\}\}$
- Note that Q^4 is infeasible as the constraints cannot be satisfied. So $\mathcal{L} = \{Q^1, Q^3\}$. - Solving R^3 of Q^3 yields $x_1 = 0, x_2 = 1, x_3 = 1$, with value $V^3 = 11$.

Iteration 3.
- Since $V^1 > V^3$ we extract Q^1 from the list. We branch on x_3 (most fractional variable). We generate - $Q^5 = \{\max 10x_1 : 6x_1 \leq 10, x_2 = 0, x_3 = 0, x_1 \in \{0, 1\}\}$
- $Q^6 = \{5 + \max 10x_1 : 6x_1 \leq 5, x_2 = 0, x_3 = 1, x_1 \in \{0, 1\}\}$
- The relaxation values are $V^5 = 10$, $V^6 = 13.3$.

Next iterations. For the reader ...

Knapsack Enumeration Tree (partial)



- The enumeration tree after iteration 3.
- The current list of open problems $\mathcal{L}^3 = \{Q^3, Q^5, Q^6\}$.

References

- Dahl, G. and Mannino, C., 2009. Notes on combinatorial optimization.
- Wolsey, L.A., 2020. Integer programming. John Wiley & Sons.
- Cook, W.J., Cunningham, W.H., Pulleyblank, W.R. and Schrijver, A., 2009. Combinatorial optimization. Wiley Interscience