Integer Linear Programming

Mikaël Capelle

mikael.capelle@thalesaleniaspace.com

2026

Thales Alenia Space

Course Organization

- Lectures — 8h45 (7 sessions)
  - Instructor —
    - Mikaël Capelle — `mikael.capelle@insa-toulouse.fr`
  - Slides available at `https://pdf.typename.fr`.
- Practical work — 11h (4 sessions)
  - Teaching assistants —
    - Marianne Desfrene - `marianne.defresne@insa-toulouse.fr`
  - Jupyter notebook — Be familiar with python!
- Grading — Written exam.

Table of Contents

Reminders

- Complexity & Optimization
- Linear Programming (LP)

Integer Linear Programming

- Definition
- Why are Integer Linear Programs hard to solve?
- Examples

Solving Integer Linear Programs

- Relaxations
- Branch & Bound
- Cuts
- Decomposition methods

Modelling problems as integer linear programs

- Modeling elementary operations
- Examples

Introduction

- Linear Programming (LP) is a paradigm for solving optimization problem where constraints are linear inequalities.
- Within linear programming, decision variables are continuous. The corresponding optimization problem can be solve in polynomial time.
- Within Integer Linear Programming (ILP), decision variables are discrete (integers). The corresponding optimization problem is hard ($\in$ NP-Hard).
- Integer-Linear Programming is often used in practice to model industrial problems, even the ones that do not appear as linear, and then use off-the-shelf solvers to find (optimal) solutions.

Reminders

Reminders

Complexity & Optimization

Reminders — What is an optimization problem?

Optimization problem $Q = (X, f, m, g) \ : \ X \to \bigcup_{x \in X} f(x)$

- X is a set of instances.
- For any instance $x \in X$, $f(x)$ is the set of feasible solutions.
- $m : X \times f(X) \to \mathbb{R}$ is the objective function.
- $g \in \{\min, \max\}$
- $Q(x) = \arg_g \{m(x, y) \ : \ y \in f(x)\}$

Decision problem Q_D **corresponding to** $Q = (X, f, m, g)$

- **Inputs**: $x \in X$, $k \in \mathbb{R}$
- **Question**: Is there $y \in f(x)$ such that $m(x, y) \leq k$?

Reminders — Classification of optimization problems (Complexity)

- P = PTIME = $\bigcup_{c \geq 0} \text{TIME}(n^c)$
  - Problems for which there exists a polynomial-time algorithm $\rightarrow$ "easy" problems.
  - Contains most of the problems you know of: shortest-path in a weighted-graph, sorting an array, finding maximum flow in a graph, ...
- EXPTIME = $\bigcup_{c \geq 0} \text{TIME}(2^{n^c})$
  - Contains P... P $\subseteq$ EXPTIME,
  - ...and much more P $\subset$ EXPTIME.

The NP complexity class — NP = Nondeterministic Polynomial

Class of problems for which we do not know if there exists a polynomial-time algorithm, but we cannot prove that none exist.

The NP-Hard complexity class

Class of problems that are "*at least as hard*" as the hardest problems in NP.

Reminders

Linear Programming (LP)

Linear Programming

$$\begin{aligned} \max. \sum_{i=1}^{n} c_i x_i & \\ \text{s.t.} \sum_{i=1}^{n} a_{ij} x_i \leq b_j, & \qquad 1 \leq j \leq m \\ x_i \geq 0, & \qquad 1 \leq i \leq n \end{aligned}$$

Linear Programming — Short version

$$\max \left\{ c^T x \mid Ax \leq b,\ x \in \mathbb{R}^n,\ x \geq 0 \right\}$$
$$c \in \mathbb{R}^n, \quad A \in \mathbb{R}^{m \times n}, \quad b \in \mathbb{R}^m$$

Comments:

- Minimizing $f \iff$ Maximizing $-f$.
- Greater-or-equal constraints:
$$x_i \geq b \iff -x_i \leq -b$$
- Equality constraints:
$$x_i = b \iff ((x_i \leq b) \wedge (-x_i \leq -b))$$
- If a variable x is negative, we can define $y = -x$ as a positive variable.
- If x is not constrained in sign ($x \in \mathbb{R}$), we can define two new variables $x^+ \geq 0$ and $x^- \geq 0$ and set $x = x^+ - x^-$.
- There can be no strict inequalities in a LP ($<$, $>$, $\neq$).

https://homepages.laas.fr/arzelier/drupal/content/current-teaching-material

Definition: Polyhedron of constraints

- Set of feasible solutions: subset $\mathcal{P}$ of $\mathbb{R}^n$ satisfying the $m + n$ constraints of the problem:
  - $\Rightarrow$ it is a polyhedron.
  - $\Rightarrow$ it is convex.
- If the polyhedron is bounded, it is a polytope.

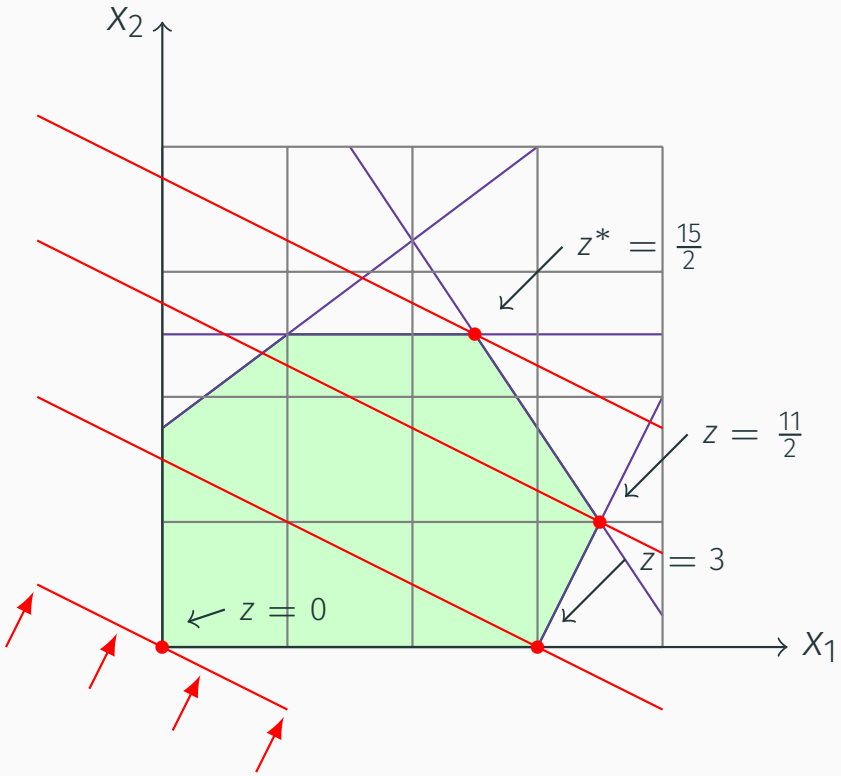
$$\mathcal{P} = \left\{ x \in \mathbb{R}^n \ : \ Ax \leq b, \ x \geq 0 \right\}$$

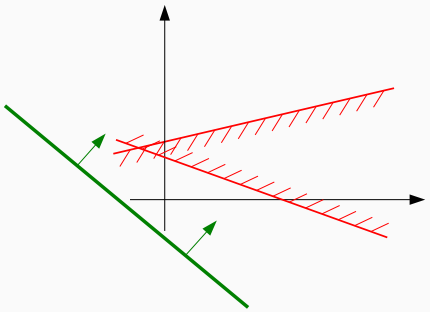
Theorem

The optimum (maximum, minimum) of a linear program, if there is one, is found on at least one of the vertices of its polyhedron.

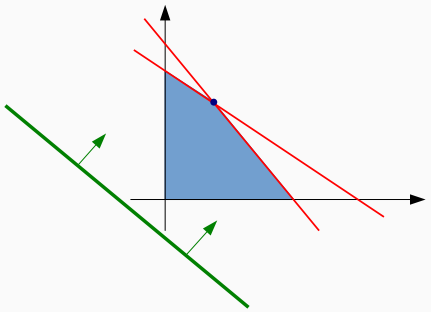
A linear program:

$$
\begin{aligned}
\text{max.} \quad & z = x_1 + 2x_2 \\
\text{s.t.} \quad & -3x_1 + 4x_2 \leq 7 \\
& 2x_2 \leq 5 \\
& 6x_1 + 4x_2 \leq 25 \\
& 2x_1 - x_2 \leq 6 \\
& x_1,\ x_2 \in \mathbb{R}^+
\end{aligned}
$$

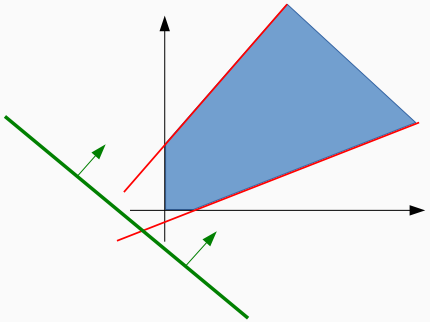




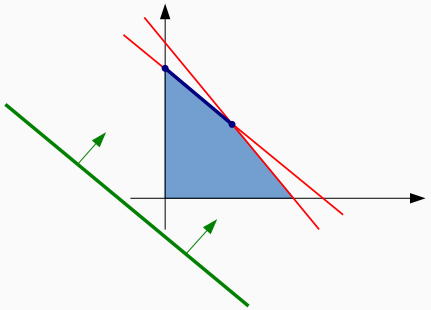
Empty polyhedron $\leftrightarrow$ No feasible solutions.



Polytope — Problem as a unique solution.



Non-bounded polyhedron — Problem is not bounded.



Polytope — Problem as multiple solutions.

Linear Programming is a polynomial-time optimization problem

- Simplex algorithm — Exponential-time algorithm in theory, but very efficient in practice.
- Interior point (Barrier) methods — Polynomial-time algorithms, more and more used in practice.

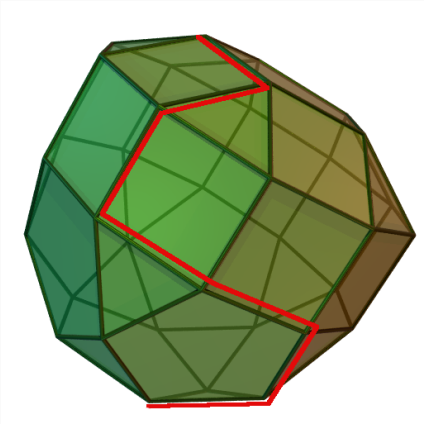


Illustration of the Simplex algorithm on a 3D polyhedron.

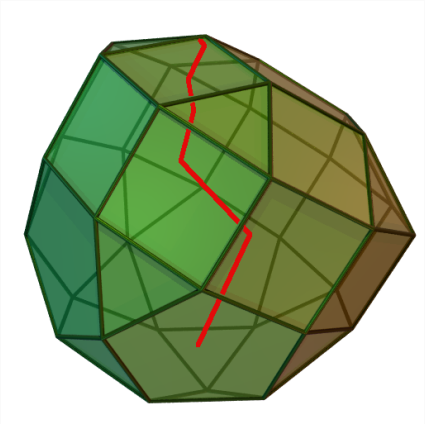


Illustration of an Interior point method on a 3D polyhedron.

Integer Linear Programming

Integer Linear Programming

Definition

Definitions

Integer Linear Programming (ILP)

Decision variables are discretes:

$$\max \left\{ c^T x \mid Ax \leq b,\ x \in \mathbb{N}^n \right\}$$

0–1 Integer Linear Programming (0–1 ILP)

Decision variables are binary variables ($\in \{0,\ 1\}$):

$$\max \left\{ c^T x \mid Ax \leq b,\ x \in \{0,\ 1\}^n \right\}$$

Mixed-Integer Linear Programming (MILP)

Decision variables can be discrete or continuous:

$$\max \left\{ c^T x \mid Ax \leq b,\ x \in \mathbb{N}^q \times \mathbb{R}_+^{n-q} \right\}$$

Integer Linear Programming

Why are Integer Linear Programs hard to solve?

Why are Integer Linear Programs hard to solve?

We cannot directly use LP algorithms (simplex, barrier).

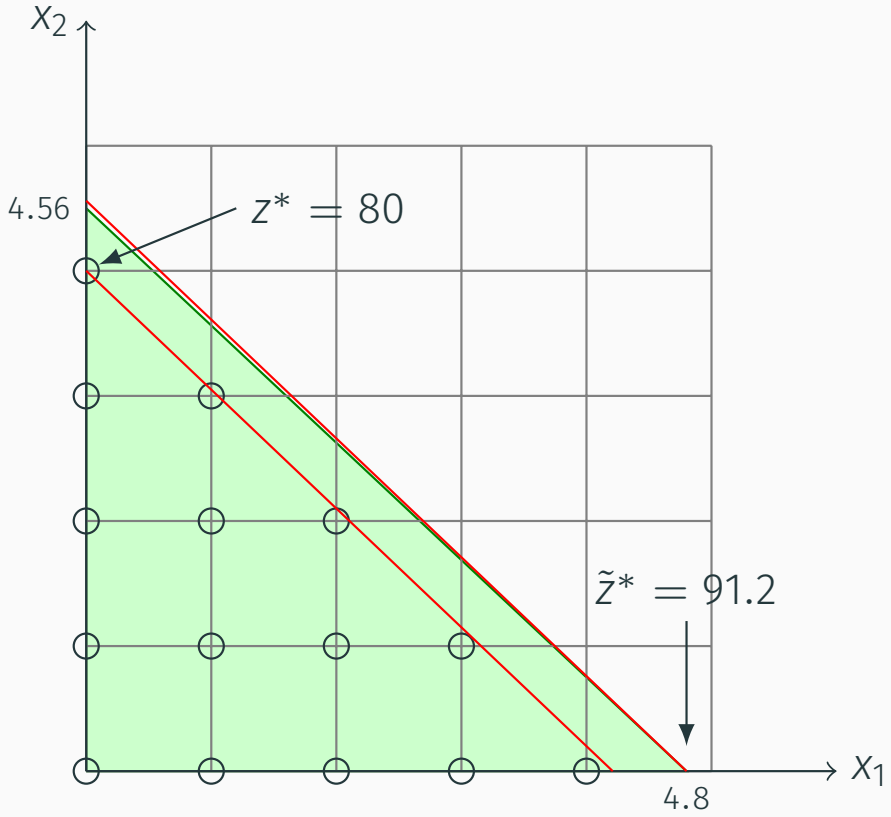
- The set of feasible solutions is not continuous.
- The optimal solution is not guaranteed to be on a vertex of the polyhedron.
- Rounding the solution of the linear relaxation is often a bad idea.

A brute-force algorithm will quickly fail as the sets of solutions grow larger in size:

- There are two many feasible solutions to enumerate all of them.

Why are Integer Linear Programs hard to solve? — Example

$$\begin{aligned} \max. \quad & z = 19x_1 + 20x_2 \\ \text{s.t.} \quad & 15x_1 + 16x_2 \leq 72 \\ & x_1,\ x_2 \in \mathbb{N} \end{aligned}$$



Why are Integer Linear Programs hard to solve?

ILP, 0–1 ILP and MILP are NP-Hard problems:

- $\rightarrow$ There are no known polynomial-time algorithms that can be used to solve any integer linear program.
  - $\rightarrow$This does not mean that all integer linear programs represent NP-Hard problems!

These are usually solved:

- using exact arborescence-based methods:
  - Generic methods: *Branch & Bound*, *Branch & Cut*, *Branch & Price*
  - Decomposition methods: *Benders*, *Dantzig-Wolfe*, *Column generation*
- using approximated algorithms — meta-heuristics, greedy heuristics, dedicated heuristics, ...

Solvers & Modeling languages off-the-shelf

It is often not necessary to implement these methods ourselves because there are multiple off-the-shelf solvers available to solve mixed-integer linear program. A non-exhaustive list of these:

- IBM CPLEX ($)
- GUROBI ($)
- COIN-OR / CBC / CLP
- GLPK

Aside of these, various languages have been developped to model mixed-integer linear program in an easy way and then use one of the above mentioned solvers: AIMMS, AMPL, LINDO, MPL, OPL, ...

Integer Linear Programming

Examples

Knapsack problem — We have a knapsack of capacity W and a set of n items with associated weights w_i and profits p_i. How can we maximize our profits while being limited by the capacity of the knapsack?

- x_i — Binary variable indicating if we take item i.

$$
\begin{aligned}
\text{max.} \quad & \sum_{i=1}^{n} x_i p_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_i w_i \leq W \\
& x_i \in \{0,\ 1\}, \qquad \forall i \in \{1,\ \ldots,\ n\}
\end{aligned}
$$

The knapsack problem is **weakly NP-Hard**.

Warehouse allocations — A company needs to supply a set of n clients and needs to open new warehouses (from a set of m possible warehouses). Opening a warehouse j costs f_j and supplying a client i from a warehouse j costs c_{ij} per supply unit. All clients have the same unit demands. Which warehouses should be opened in order to satisfy all clients while minimizing the total cost?

$$
\begin{aligned}
\text{min.} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} + \sum_{j=1}^{m} f_j y_j & \\
\text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} = 1, & \forall i \in \{1, \ldots, n\} \\
& \sum_{i=1}^{n} x_{ij} \leq n y_j, & \forall j \in \{1, \ldots, m\} \\
& y_j \in \{0, 1\}, & \forall j \in \{1, \ldots, m\} \\
& x_{ij} \geq 0, & \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}
\end{aligned}
$$

$$\text{min.} \quad \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} + \sum_{j=1}^{m} f_j y_j \tag{1a}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} x_{ij} = 1, \qquad \forall i \in \{1, \ldots, n\} \tag{1b}$$

$$\sum_{i=1}^{n} x_{ij} \leq n y_j, \qquad \forall j \in \{1, \ldots, m\} \tag{1c}$$

$$y_j \in \{0, 1\}, \qquad \forall j \in \{1, \ldots, m\} \tag{1d}$$

$$x_{ij} \geq 0, \qquad \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\} \tag{1e}$$

- x_{ij} is the fraction supplied from warehouse j to customer i.
- $y_j = 1$ if and only if warehouse j is opened.
- (1b) indicates that a client must be fully supplied.
- (1c) indicates that if we supply a client from a warehouse j ($x_{ij} > 0$ for some client i), this warehouse must be opened ($y_j = 1$).

$$\min. \quad \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{j=1}^{m} f_j y_j \tag{1a}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} x_{ij} = 1, \qquad \forall i \in \{1, \ldots, n\} \tag{1b}$$

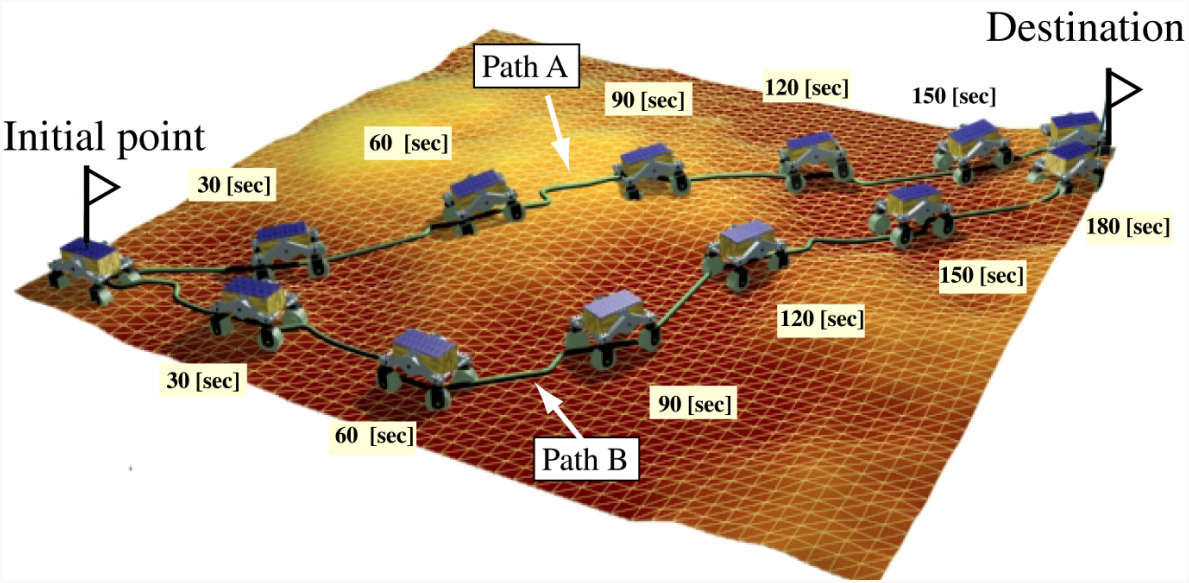
$$\sum_{i=1}^{n} x_{ij} \leq y_j, \qquad \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\} \tag{1c}$$

$$y_j \in \{0, 1\}, \qquad \forall j \in \{1, \ldots, m\} \tag{1d}$$

$$x_{ij} \geq 0, \qquad \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\} \tag{1e}$$

- x_{ij} is the fraction supplied from warehouse j to customer i.
- $y_j = 1$ if and only if warehouse j is opened.
- (1b) indicates that a client must be fully supplied.
- (1c) indicates that if we supply a client from a warehouse j ($x_{ij} > 0$ for some client i), this warehouse must be opened ($y_j = 1$).

Shortest-path — A road network is represented by a directed graph $\mathcal{G} = (V, A)$ with costs c_{ij} associated to each road (arc) $(i, j) \in A$. What is the shortest-path from s to t?



Spatial application: Autonomous Mars rover navigation (Candaian Space Agency: [DRB+08], LAAS [HST02])

Shortest-path — A road network is represented by a directed graph $\mathcal{G} = (V, A)$ with costs c_{ij} associated to each road (arc) $(i, j) \in A$. What is the shortest-path from s to t?

- x_{ij} — Binary variable indicating if we use arc (i, j).

$$
\begin{aligned}
\text{min.} \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{(s,k)\in A} x_{sk} - \sum_{(k,s)\in A} x_{ks} = 1 \\
& \sum_{(i,k)\in A} x_{ik} - \sum_{(k,i)\in A} x_{ki} = 0, && \forall i \in V \setminus \{s, t\} \\
& \sum_{(t,k)\in A} x_{tk} - \sum_{(k,t)\in A} x_{kt} = -1 \\
& x_{ij} \in \{0,\ 1\}, && \forall (i, j) \in A
\end{aligned}
$$

Shortest-path is a polynomial problem, we will see later why we can solve this integer linear program in polynomial time using a linear relaxation.

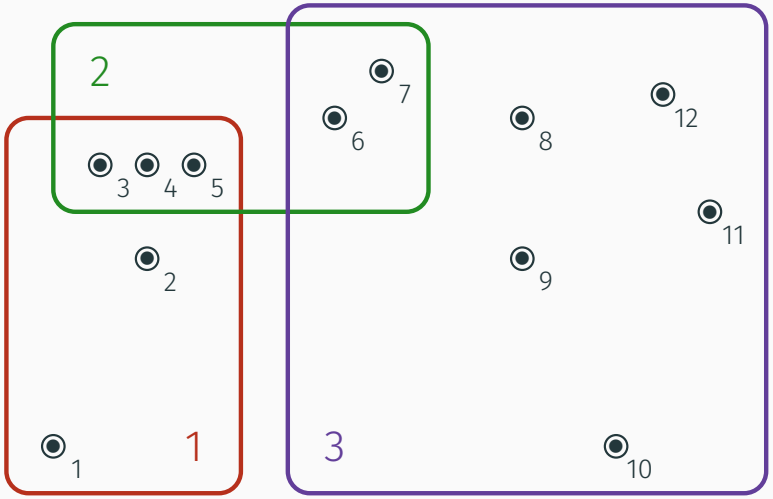
Set cover — We want to check a set S of critical points in a system using a set T possible tests. Each test cover a subset of critical points ($a_{ij} = 1$ if test i covers critical point j) and costs c_i. Which tests should be set up in order to minimize the costs while ensuring that each critical points is tested?

- x_i — Binary variable indicating if we set up test i.

$$
\begin{aligned}
\text{min.} \quad & \sum_{i \in T} c_i x_i & \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_{ij} x_i \geq 1 & \forall j \in S \\
& x_i \in \{0,\ 1\}, & \forall i \in T
\end{aligned}
$$

The set-cover problem is strongly NP-Hard.

Set cover — We want to check a set S of critical points in a system using a set T possible tests. Each test cover a subset of critical points ($a_{ij} = 1$ if test i covers critical point j) and costs c_i. Which tests should be set up in order to minimize the costs while ensuring that each critical points is tested?

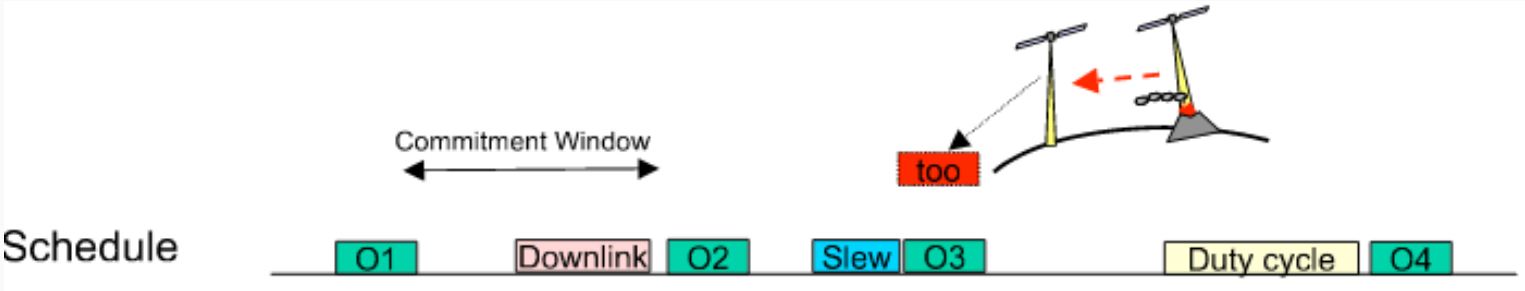


$c_1 = 6 \quad c_2 = 15 \quad c_3 = 7$

$$\begin{array}{ll} \text{min.} & 6x_1 + 15x_2 + 7x_3 \\ \text{s.t.} & \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} (x_1, x_2, x_3) \geq \mathbb{1}_{12} \\ & x_1, x_2, x_3 \in \{0,\ 1\} \end{array}$$

Optimal solution is $x_1 = 1, x_2 = 0, x_3 = 1$ which covers all tests and cost $6 + 7 = 13$.

Single machine scheduling — A machine needs to produce a set L of n items. Each item i has a manufacturing time p_i, a release time r_i (time at which its components are available) and a deadline d_i (time at which the time should be manufactured). The factory can only produce **one item at a time**, and manufacturing process **cannot be interrupted** (non-preemptive machine), what is the schedule that minimize the overdue deliveries?



Spatial application: Scheduling communications in satellite control network (U.S. Air Force, [BWWH04])

The single machine scheduling problem is **strongly NP-HARD**.

Integer linear programming — Single machine scheduling example

We will use disjunctive model: we will enforce that a task (item) i must be scheduled (manufactured) before or after a task j.

- x_{ij} — Binary variable indicating if i is manufactured before j.
- s_i — Continuous variable indicating the starting time of task i.
- y_i — Binary variable indicating if the item i is overdue.

$$\min. \quad \sum_{j=1}^{n} y_i \tag{2a}$$

$$\text{s.t.} \quad s_j \geq s_i + p_i - M(1 - x_{ij}), \qquad \forall i, j \in L,\ i \neq j \tag{2b}$$

$$s_i + p_i \leq d_i + My_i, \qquad \forall i \in L \tag{2c}$$

$$x_{ij} + x_{ji} = 1, \qquad \forall i, j \in L,\ i \neq j \tag{2d}$$

$$x_{ij} \in \{0,\ 1\}, \qquad \forall i, j \in L \tag{2e}$$

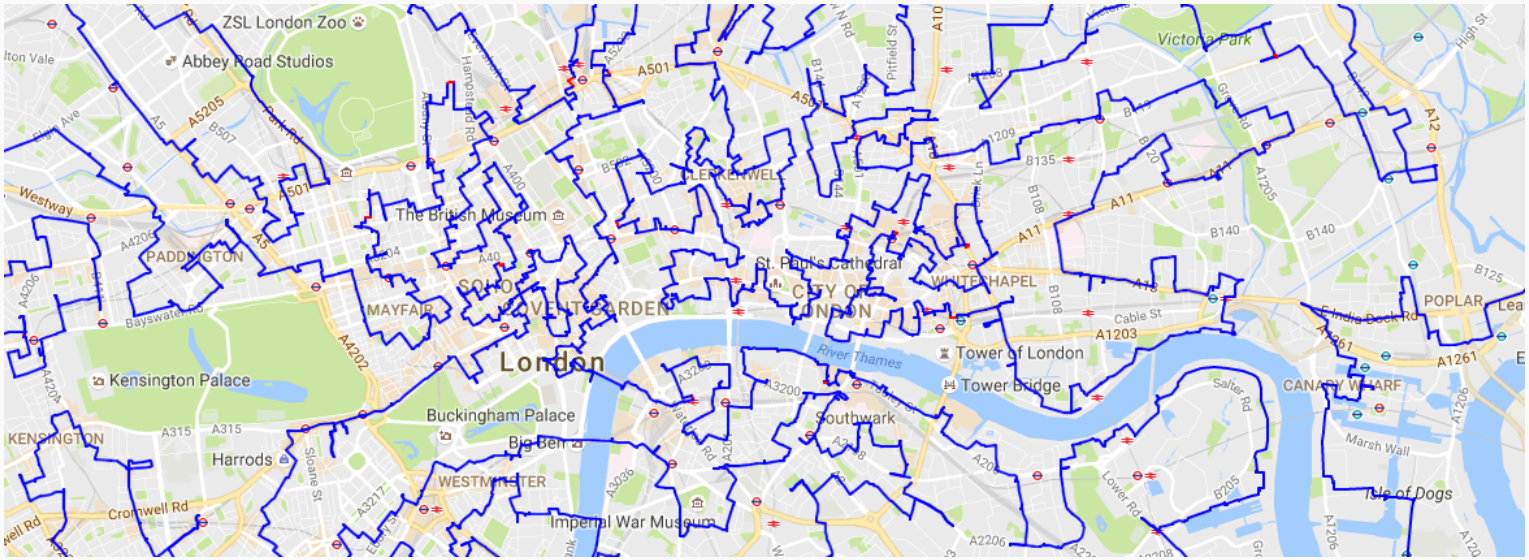
$$y_i \in \{0,\ 1\}, \qquad \forall i \in L \tag{2f}$$

$$s_i \geq r_i, \qquad \forall i \in L \tag{2g}$$

- (2d) Two items must be ordered — i is before j or j is before i.
- (2c) $y_i = 1$ if $s_i + p_i > d_i - M$ must be big enough to inhibit the constraint when $y_i = 0$.
- (2b) If j is after i ($x_{ij} = 1$), then $s_j \geq s_i + p_i - M$ must be big enough to inhibit the constraint if j is before i ($x_{ij} = 0$).

The larger M, the harder the resolution will be, but if M is not big enough, the formulation may not be valid — We want to find a "good" value for M. Here we can choose $M = \max_{i \in L} r_i + \sum_{i \in L} p_i$.

Travelling salesman problem — Given a list of n cities and the distances c_{ij} between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?



General application: A shortest-possible walking tour through the pubs of the United Kingdom. (W. Cook et al., [UK24727]) — `http://www.math.uwaterloo.ca/tsp/pubs/`

The travelling salesman problem is **strongly NP-Hard**.

Miller-Tucker-Zemlin (MTZ) formulation —

- $x_{ij} \in \{0, 1\}$ — Binary variable indicating if we go directly from city i to city j.
- $u_i \in \{1, \ldots, n\}$ — Subtour elimination variables — u_i is the position of city i in the tour.

$$\min. \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{3a}$$

$$\text{s.t.} \quad \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1, \qquad \forall i \in \{1, \ldots, n\} \tag{3b}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1, \qquad \forall j \in \{1, \ldots, n\} \tag{3c}$$

$$u_1 = 1 \tag{3d}$$

$$2 \leq u_i \leq n, \qquad \forall i \in \{1, \ldots, n\} \tag{3e}$$

$$u_i - u_j + 1 \leq (n-1)(1 - x_{ij}), \qquad \forall i, j \in \{2, \ldots, n\} \tag{3f}$$

$$x_{ij} \in \{0, 1\}, \; u_i \in \mathbb{N} \qquad \forall i, j \in \{1, \ldots, n\} \tag{3g}$$

- (3b) We leave each city exactly once.
- (3c) We enter each city exactly once.
- (3d–3f) Subtour elimination — arc-constraints: $x_{ij} = 1 \Rightarrow u_j \geq u_i + 1$.

Solving Integer Linear Programs

Solving Integer Linear Programs

Relaxations

Relaxations

Let (P) be an optimization problem $\max z = f(x)$ under $x \in \mathcal{X} \subseteq \mathbb{R}^n$.

General definition

The optimization problem $(R) \ : \ \max z_R = \phi(x)$ under $x \in \mathcal{T} \subseteq \mathbb{R}^n$ is a relaxation of (P) if $\mathcal{X} \subseteq \mathcal{T}$ and $\forall x \in \mathcal{X},\ \phi(x) \geq f(x)$.

It is possible to relax the set of feasible solutions (constraints), the objective function, or both.

Theorem

If (R) is a relaxation of (P), then $z_R^* \geq z^*$ (maximization) — (R) provides an ***upper bound*** *for (P).*

Let (R) and (R') be two relaxations of (P), (R) is **better** than (R') if:

$$z_R^* \leq z_{R'}^*,$$

Some common relaxations for integer linear programs:

"Combinatorial"

Remove some or all constraints:

$$\max\left\{c^T x \mid Ax \leq b,\ x \in \mathbb{N}^n\right\} \Rightarrow \max\left\{c^T x \mid A'x \leq b',\ x \in \mathbb{N}^n\right\}$$

Lagrangian

Remove and penalize (some) constraints in the objective:

$$\max\left\{c^T x \mid Ax = b,\ x \in \mathbb{R}^n_+\right\} \Rightarrow \max\left\{c^T x - \lambda^T(b_1 - A_1 x) \mid A_2 x = b_2,\ x \in \mathbb{R}^n_+\right\}$$

Linear

Remove the integrity constraints on the variables:

$$\max\left\{c^T x \mid Ax \leq b,\ x \in \mathbb{N}^n\right\} \Rightarrow \max\left\{c^T x \mid Ax \leq b,\ x \in \mathbb{R}^n_+\right\}$$

$$\max\left\{c^T x \mid Ax \leq b,\ x \in \{0,\ 1\}^n\right\} \Rightarrow \max\left\{c^T x \mid Ax \leq b,\ x \in \mathbb{R}^n,\ 0 \leq x \leq 1\right\}$$

Consider an (integer) linear programs (P) containing $p + q$ constraints. A lagrangian relaxation of (P) is an (integer) linear program (L) similar to (P) where q constraints have been removed and **penalized** in the objective function.

Linear program (P)

$$\begin{aligned} \text{max.} \quad & z = c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & Dx \leq e \\ & x \in \mathbb{N}^+ \end{aligned}$$

Lagrangian relaxation (L_λ) of (P)

$$\begin{aligned} \text{max.} \quad & z = c^T x + \lambda^T (b - Ax) \\ \text{s.t.} \quad & Dx \leq e \\ & x \in \mathbb{N}^+ \end{aligned}$$

If we consider only **non-negative weights** $\lambda \geq 0$, we ensure that this is a **valid** relaxation:

$$\begin{aligned} \forall x \in \mathcal{P} : \quad & Ax \leq b \\ & c^T x \leq c^T x + \lambda^T (b - Ax) \end{aligned}$$

Generalized assignment problem — We want to assign n items to m bins. Assigning item i to bin j costs c_{ij} and uses a_{ij} space in the bin. Each bin j has a capacity b_j. Each item must be assigned to a bin. What is the assignment that minimizes the total assignment cost?

- $x_{ij} \in \{0,\ 1\}$ — $x_{ij} = 1$ if we put item i in bin j.

$$\text{min.} \quad z = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \tag{4a}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} x_{ij} = 1, \qquad \forall j \in \{1,\ \ldots,\ m\} \tag{4b}$$

$$\sum_{i=1}^{n} a_{ij} x_{ij} \leq b_j, \qquad \forall j \in \{1,\ \ldots,\ m\} \tag{4c}$$

$$x_{ij} \in \{0,\ 1\}, \qquad \forall i, j \tag{4d}$$

- (4a) — The objective is to minimize the total cost.
- (4b) — Each item must be assigned to a bin.
- (4c) — Capacities of the bins must be respected.

Generalized assignment problem — We want to assign n items to m bins. Assigning item i to bin j costs c_{ij} and uses a_{ij} space in the bin. Each bin j has a capacity b_j. Each item must be assigned to a bin. What is the assignment that minimizes the total assignment cost?

A possible lagrangian relaxation for the generalized assignment problem:

$$\text{min.} \quad z = \sum_{i=1}^{n} \sum_{j=1}^{m} (c_{ij} + \lambda_i) x_{ij} - \sum_{i=1}^{n} \lambda_i \tag{5a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} a_{ij} x_{ij} \leq b_j, \qquad \forall j \in \{1, \ \ldots, \ m\} \tag{5b}$$

$$x_{ij} \in \{0, \ 1\}, \qquad \forall i, j \tag{5c}$$

For a given bin j, this problem reduces to a knapsack problem, and since the problems are independent regarding the bins, we can solve this problem by solving m knapsack problems.

Problem: How to find good λ_i?

Constrained Shortest-Path — Consider a directed graph $\mathcal{G}- = (V, A)$ where travel times t_{ij} and costs c_{ij} are associated to edges $(i, j) \in E$. What is the shortest path from node $s \in V$ to node $t \in V$ that costs less than K?

- x_{ij} — Binary variable indicating if we use arc (i, j).

$$
\begin{aligned}
\text{min.} \quad & \sum_{(i,j)\in A} t_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{(i,k)\in A} x_{ik} - \sum_{(k,i)\in A} x_{ki} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \forall i \in V \setminus \{s, t\} \end{cases} \\
& \sum_{(i,j)\in A} c_{ij} x_{ij} \leq K \\
& x_{ij} \in \{0,\ 1\}, \quad \forall (i, j) \in A
\end{aligned}
$$

We can relax the last constraint $\sum_{(i,j)\in A} c_{ij} x_{ij} \leq K$.

Constrained Shortest-Path — Consider a directed graph $\mathcal{G}- = (V, A)$ where travel times t_{ij} and costs c_{ij} are associated to edges $(i, j) \in E$. What is the shortest path from node $s \in V$ to node $t \in V$ that costs less than K?

- x_{ij} — Binary variable indicating if we use arc (i, j).

$$
\begin{aligned}
\text{min.} \quad & \sum_{(i,j)\in A} (t_{ij} + \mu c_{ij}) x_{ij} \\
\text{s.t.} \quad & \sum_{(i,k)\in A} x_{ik} - \sum_{(k,i)\in A} x_{ki} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \forall i \in V \setminus \{s, t\} \end{cases} \\
& x_{ij} \in \{0,\ 1\}, \quad \forall (i, j) \in A
\end{aligned}
$$

We obtain a simple shortest-path problem on $\mathcal{G}$ with modified travel times $t_{ij} + \mu c_{ij}$.

Lagrangian relaxation (L_λ) of (P)

$$L(\lambda) = \max \left\{ c^T x - \lambda^T (b - Ax) \mid Dx \leq e,\ x \in \mathbb{N}^n \right\}$$

$L(\lambda)$ is an upper bound for (P), L^* is the tightest of all bounds:

Lagrangian dual

$$L^* = \min_{\lambda \geq 0} L(\lambda)$$

Problem: How to solve the lagrangian dual?

Subgradient method

- Choose a starting point λ^0 ($t = 0$), e.g. $\lambda^0 = 0$.
- Find the optimum x^t of (L_{λ^t}) and compute the subgradient $s^t = b - Ax^t$.
- If $s^t = 0$, then stop.
- Compute $\lambda^{t+1} = \max\left\{0,\ \lambda^t + \gamma^t s^t\right\}$, where γ^t denotes the step size.
- Increment t and go to 2.

Choosing a good step size γ^t is crucial for the method to converge quickly — Any γ^t series with the following properties guarantees convergence to L^*:

$$\sum_{t=0}^{+\infty} \gamma^t = +\infty \quad \text{and} \quad \lim_{t \to +\infty} \gamma^t = 0$$

One example for such a sequence is $\gamma^t = \frac{1}{t+1}$ — In practice this does not always lead to quick convergence, hence other step sizes are chosen.

The following γ^t sequence was proposed by Held and Karp:

$$\gamma^t = \mu^t \frac{z^* - L(\lambda_t)}{||b - Ax||_2}$$

Where:

- z^* is the value of the best solution for (P) found so far.
- μ^t is a decreasing adaptation parameter with $0 < \mu^0 < 2$ and:

$$\mu^{t+1} = \begin{cases} \alpha\mu^t & \text{if } L^* \text{ did not increase in the last } T \text{ iterations.} \\ \mu^t & \text{otherwise.} \end{cases}$$

Consider an integer linear programs (P) containing constraints. A linear relaxation of (P) is a linear program (R) similar to (P) where the integrity constraints on the decision variables have been replaced by positivity constraints.

Linear program (P)

$$\begin{aligned} \max. \quad & z = c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{N}^n \end{aligned}$$

A linear relaxation of (P)

$$\begin{aligned} \max. \quad & z = c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{R}^n_+ \end{aligned}$$

Linear relaxations can be used to find upper/lower **bounds** on the objective functions (*Branch & Bound*) or to generate **cuts** for the integer linear program (*Branch & Cuts*).

If the optimal solution of the linear relaxation (P_R) is integral, it is also the optimal solution of the original program (P):

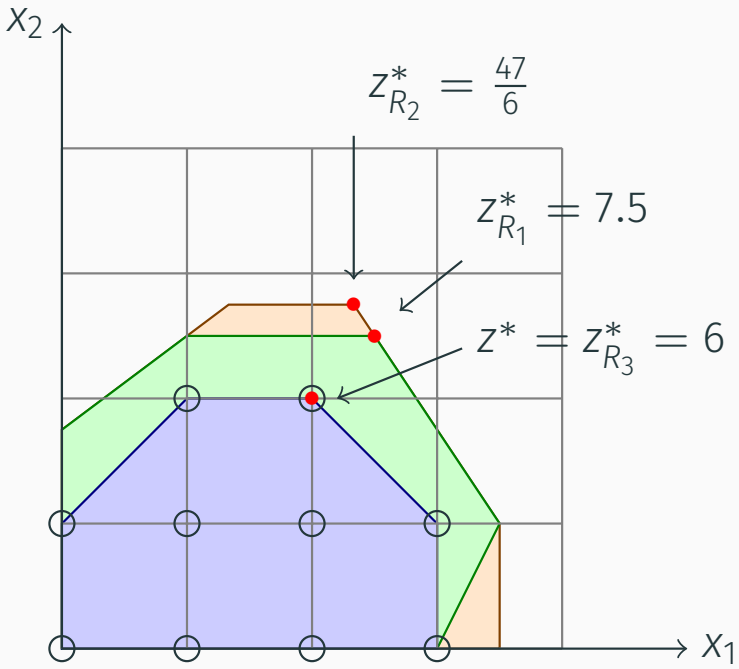
- most of the time, this will not be the case;
- there are some integer linear programs for which the optimal solution of the linear relaxation is **guaranteed** to be integral.

Proposition

Any integer linear program has an infinite number of linear relaxations.

Initial ILP (P):

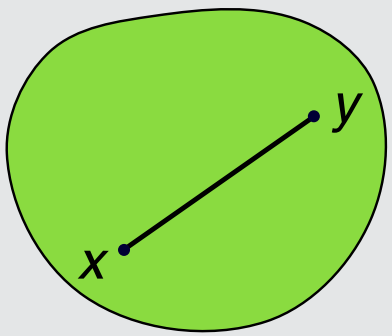
$$\begin{aligned} \max. \quad & z = x_1 + 2x_2 \\ \text{s.t.} \quad & -3x_1 + 4x_2 \leq 7 \\ & 2x_2 \leq 5 \\ & 6x_1 + 4x_2 \leq 25 \\ & 2x_1 - x_2 \leq 6 \\ & x_1,\ x_2 \in \mathbb{N} \end{aligned}$$



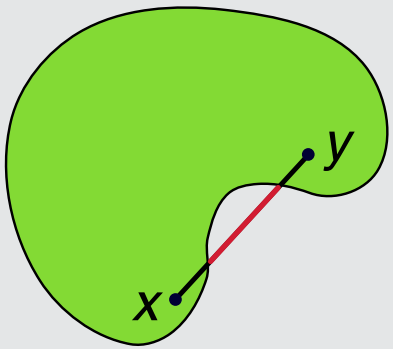
(R_3) is an ideal linear relaxation for (P) since $z^* = z^*_{R_3}$ — The polyhedron defined by (R_3) is the convex hull of (P).

Reminder: Convex set

A convex set is a set of points such that, given any two points x, y in that set, the line joining them lies entirely within that set.



Convex set.



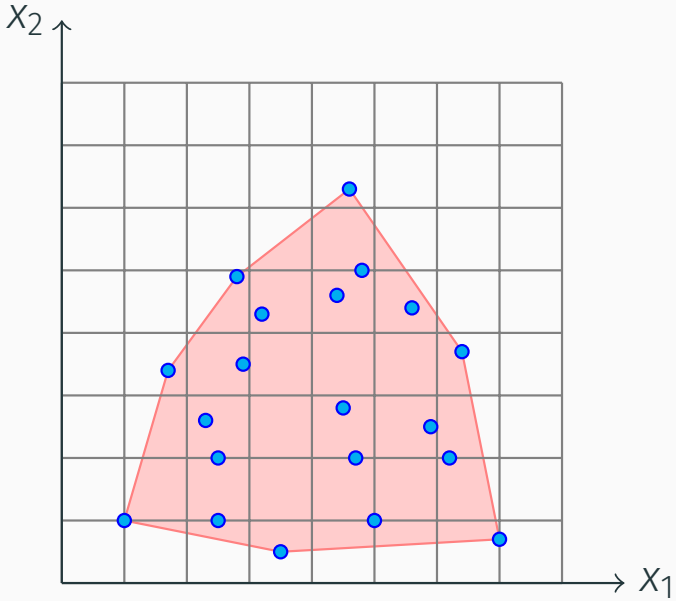
Non-convex set.

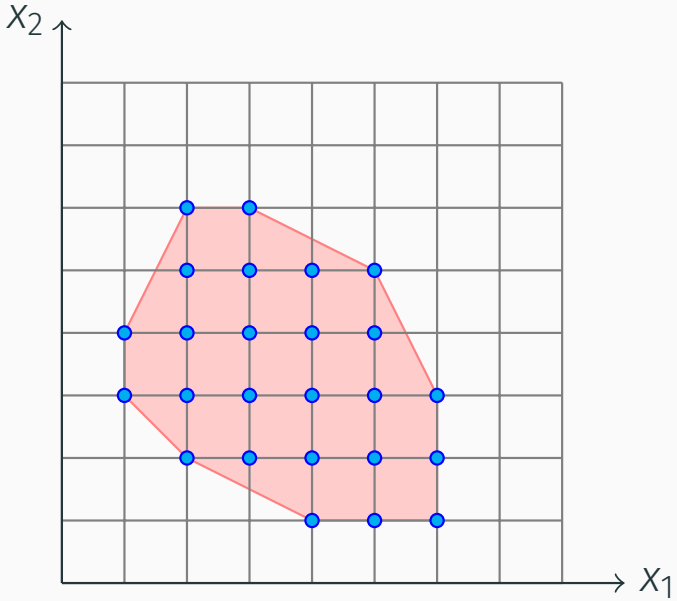
Definition: Convex Hull

The convex hull (or convex envelope) of a set $\mathcal{X}$ of points is the **unique** smallest convex set that contains $\mathcal{X}$.

Let $\mathcal{X} = \{x_i,\ 1 \leq i \leq n\}$ be a set of points, its **convex hull** $conv(\mathcal{X})$ can be defined as:

$$conv(\mathcal{X}) = \left\{ \sum_{i=1}^{k} \lambda_i x_i \ : \ \lambda_i \geq 0, \ \sum_{i=1}^{k} \lambda_i = 1 \right\}$$





Consider the following integer linear program:

$$(P) \ : \ \max \left\{ z = f(x),\ x \in \mathcal{X} \subseteq \mathbb{N}^n \right\}$$

And a general relaxation of its integrity constraints:

$$(R) \ : \ \max \left\{ z_R = f(x),\ x \in \mathcal{T} \subseteq \mathbb{R}^n \right\}$$

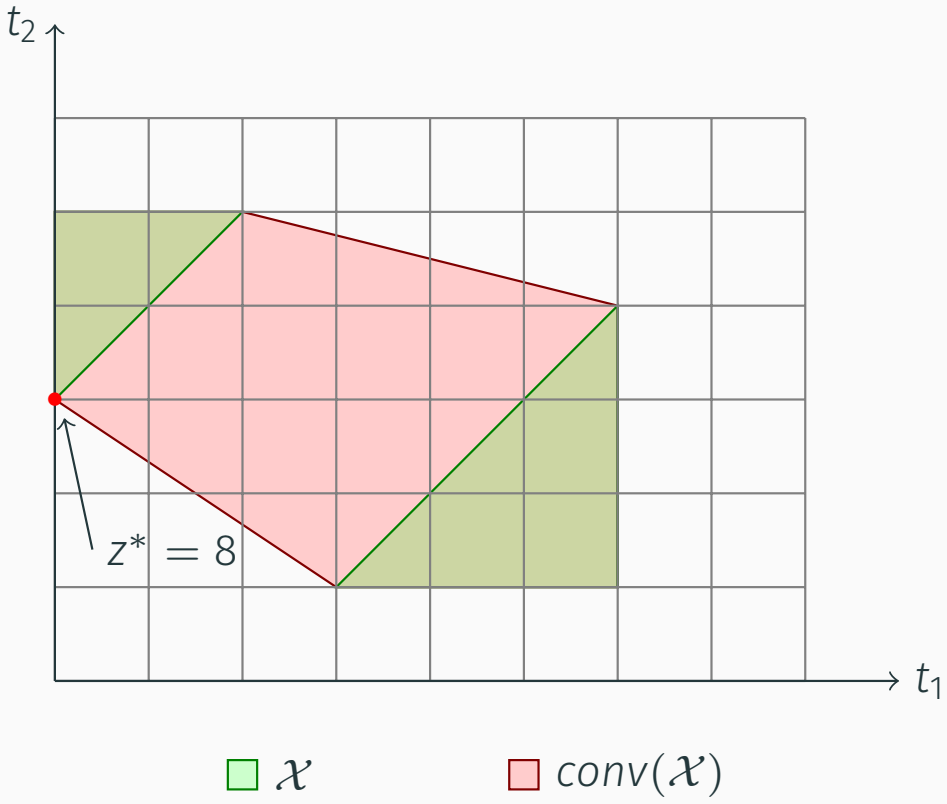
Theorem

$$\mathcal{T} = conv(\mathcal{X}) \implies z_R^* = z^*$$

Problem: Finding the convex hull of (P) is NP-Hard.

We want to schedule two jobs J_1 and J_2 with knows durations $p_1 = 3$ and $p_2 = 2$ on a single sequential machine. We need to find the starting time t_1 and t_2 of each task, such that $t_1 \in [0,\ 6]$ and $t_2 \in [1,\ 5]$. We want to minimize $f(t) = t_1 + t_2 + p_1 + p_2$.

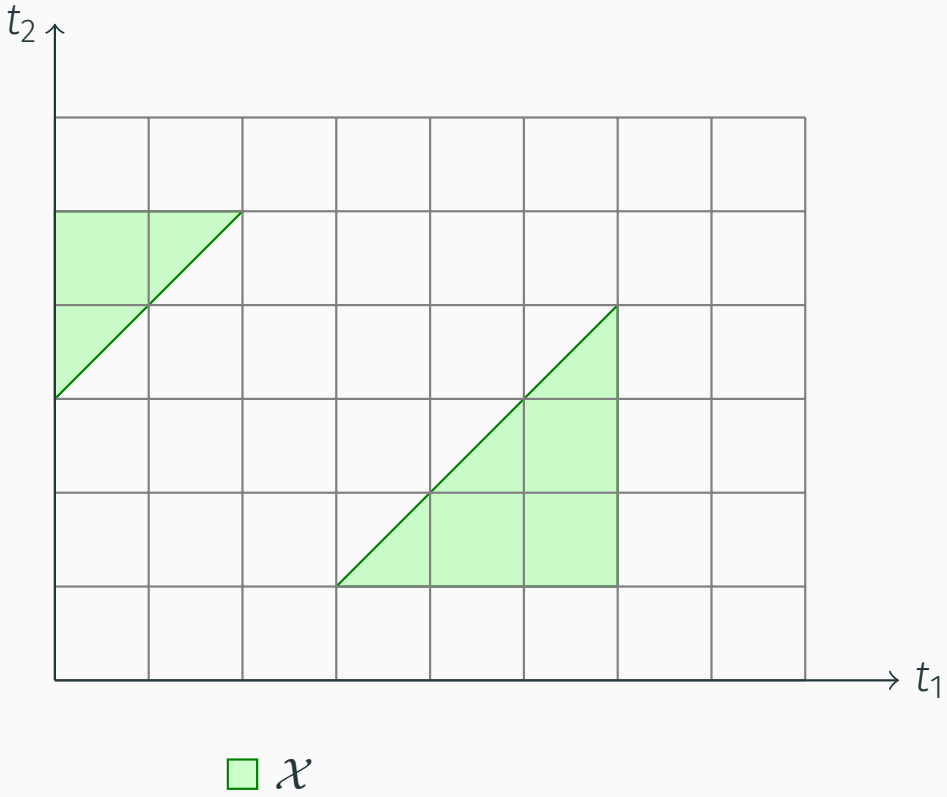
$$
\begin{aligned}
\text{min.} \quad & t_1 + t_2 + 5 \\
\text{s.t.} \quad & t_1 \geq 0 \\
& t_1 \leq 6 \\
& t_2 \geq 1 \\
& t_2 \leq 5 \\
& (t_2 \geq t_1 + 3) \vee (t_1 \geq t_2 + 2)
\end{aligned}
$$



Problem: How can we linearize the disjunctive constraint ($\vee$)?

We want to introduce a binary variable x to indicates which alternative is chosen: J_1 before J_2 or J_2 before J_1. We get an integer linear programs on $\mathcal{X}' \subseteq \mathbb{R} \times \mathbb{R} \times \{0,\ 1\}$:

$$
\begin{aligned}
\text{min.} \quad & t_1 + t_2 + 5 \\
\text{s.t.} \quad & t_1 \geq 0 \\
& t_1 \leq 6 \\
& t_2 \geq 1 \\
& t_2 \leq 5 \\
& t_2 - t_1 + 8x \geq 3 \\
& t_1 - t_2 + 7(1 - x) \geq 2 \\
& x \in \{0,\ 1\}
\end{aligned}
$$

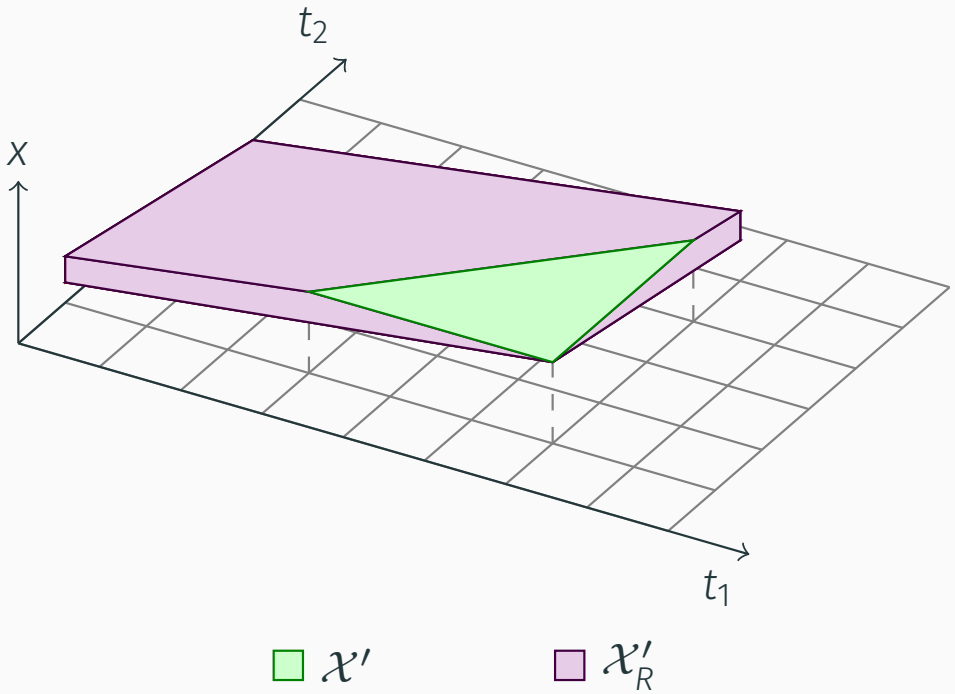


The formulation is **valid** because the projection of $\mathcal{X}'$ on $\mathbb{R}^2$ is $\mathcal{X}$.

We want to introduce a binary variable x to indicates which alternative is chosen: J_1 before J_2 or J_2 before J_1. We get an integer linear programs on $\mathcal{X}' \subseteq \mathbb{R} \times \mathbb{R} \times \{0,\ 1\}$:

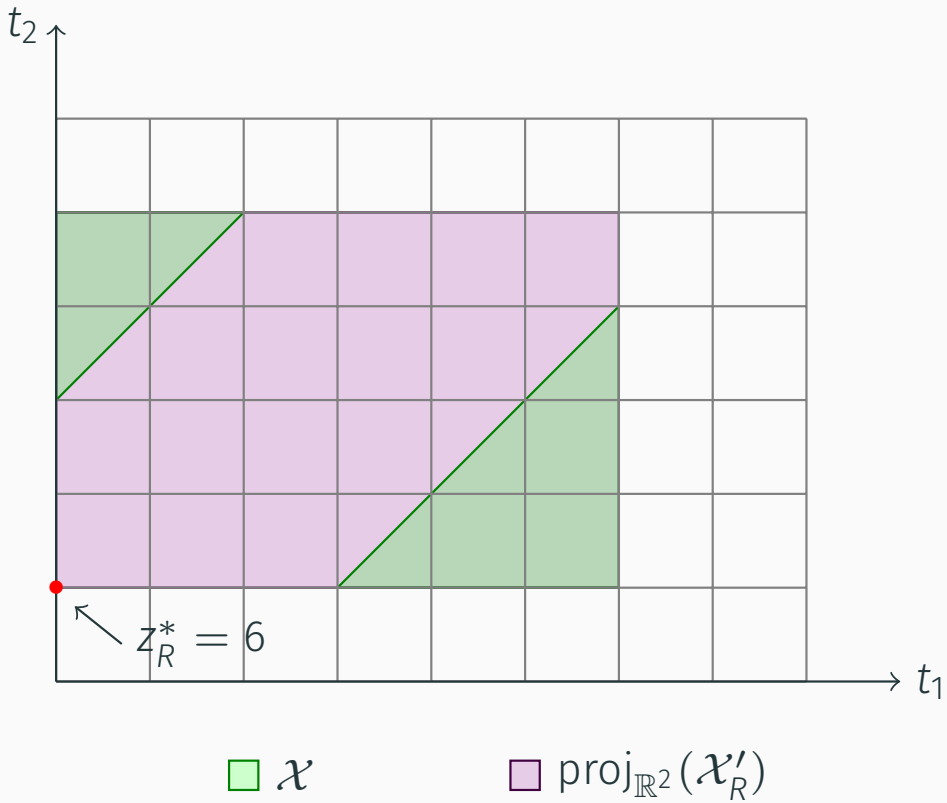
;

$$
\begin{aligned}
\text{min.} \quad & t_1 + t_2 + 5 \\
\text{s.t.} \quad & t_1 \geq 0 \\
& t_1 \leq 6 \\
& t_2 \geq 1 \\
& t_2 \leq 5 \\
& t_2 - t_1 + 8x \geq 3 \\
& t_1 - t_2 + 7(1 - x) \geq 2 \\
& x \in \{0,\ 1\}\ 0 \leq x \leq 1
\end{aligned}
$$



We want to introduce a binary variable x to indicates which alternative is chosen: J_1 before J_2 or J_2 before J_1. We get an integer linear programs on $\mathcal{X}' \subseteq \mathbb{R} \times \mathbb{R} \times \{0,\ 1\}$:

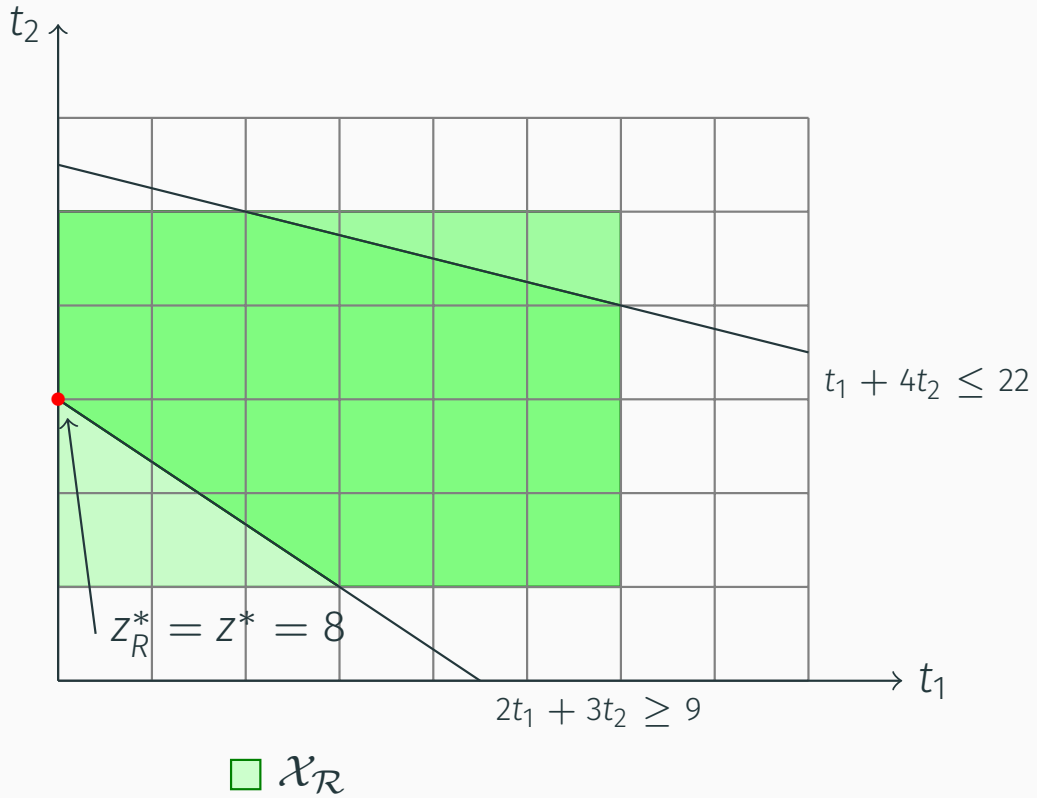
$$
\begin{aligned}
\text{min.} \quad & t_1 + t_2 + 5 \\
\text{s.t.} \quad & t_1 \geq 0 \\
& t_1 \leq 6 \\
& t_2 \geq 1 \\
& t_2 \leq 5 \\
& t_2 - t_1 + 8x \geq 3 \\
& t_1 - t_2 + 7(1 - x) \geq 2 \\
& 0 \leq x \leq 1
\end{aligned}
$$



This formulation is **valid** because the projection of $\mathcal{X}'$ on $\mathbb{R}^2$ is $\mathcal{X}$ **but** since the projection of its linear relaxation $\mathcal{X}'_R$ on $\mathbb{R}^2$ is far from $conv(\mathcal{X})$, it is a **weak** relaxation.

We want to find a formulation for (P) such that the linear relaxation of the formulation gives us the convex hull of (P) — This would be an **ideal** formulation.

$$
\begin{aligned}
\text{min.} \quad & t_1 + t_2 + 5 \\
\text{s.t.} \quad & t_1 \geq 0 \\
& t_1 \leq 6 \\
& t_2 \geq 1 \\
& t_2 \leq 5 \\
& 2t_1 + 3t_2 \geq 9 \\
& t_1 + 4t_2 \leq 22 \\
& t_1, t_2 \in \mathbb{R}
\end{aligned}
$$



Problem: Finding the **ideal** formulation for a problem is most of the time very difficult or impossible.

Theorem

Let $B \in \mathbb{Z}^{m \times m}$ be a non-singular matrix, then:

$$Det(B) = \pm 1 \quad \iff \quad \forall b \in \mathbb{Z}^m,\ B^{-1}b \in \mathbb{Z}^m$$

Let $B \in \mathbb{Z}^{m \times m}$ be a non-singular matrix, if B is **unimodular**, then $Bx = b$ has an integral solution.

Definition

A square matrix B is **unimodular** if $Det(B) = \pm 1$.

Definition

A matrix $A \in \mathbb{Z}^{m \times n}$ is **totally unimodular** if every square non-singular submatrix of A is unimodular.

Let $(P) \ : \ \max \left\{ c^T x, \ Ax \leq b, \ x \in \mathbb{N}^n \right\}$ be an integer linear programs.

Corollary

If A is totally unimodular, one of the solution of the linear relaxation of (P) is integral.

https://www.ise.ncsu.edu/fuzzy-neural/wp-content/uploads/sites/9/2016/02/or766_TUM.pdf

The following conditions together are **sufficient** for $A \in \mathbb{Z}^{m \times n}$ to be totally unimodular:

- Every column of A contains at most **two** non-zero entries:

$$\forall i : \sum_{i=1}^{m} \left| a_{ij} \right| \leq 2$$

- Every entry in A is 0, $+1$ or -1:

$$\forall i, j : a_{ij} \in \{-1,\ 0,\ 1\}$$

- We can split the rows of A into two disjoint sets M_1 and M_2 such that:
  - If two non-zero entries in a column of A have the same sign, then the rows corresponding to the non-zero entries are not in the same set.
  - If two non-zero entries in a column of A have opposite signs, both corresponding rows are in the same set.

$$\forall j,\ \sum_{i=1}^{m} \left| a_{ij} \right| = 2 \quad \Longrightarrow \quad \sum_{i \in M_1} a_{ij} = \sum_{i \in M_2} a_{ij}$$

Assignment problem — We want to assign n employees to n tasks. It takes c_{ij} hours for employee i to complete j — Some employees are more qualified than others at doing some tasks. What is the assignment that minimizes the total working time of employees?

- $x_{ij} \in \{0,\ 1\}$ — $x_{ij} = 1$ if employee i is assigned to task j.

$$
\begin{aligned}
\text{min.} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_{ij} = 1, && \forall j \in \{1,\ \ldots,\ n\} \\
& \sum_{j=1}^{n} x_{ij} = 1, && \forall i \in \{1,\ \ldots,\ n\} \\
& x_{ij} \in \{0,\ 1\}, && \forall i, j \in \{1,\ \ldots,\ n\}
\end{aligned}
$$

Example for $n = 3$:

$$
\begin{array}{ccccccccc}
x_{11} & x_{12} & x_{13} & x_{21} & x_{22} & x_{23} & x_{31} & x_{32} & x_{33} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}
\qquad
\begin{array}{l}
\\ M_1 \\ \\ \\ M_2 \\ \\
\end{array}
$$

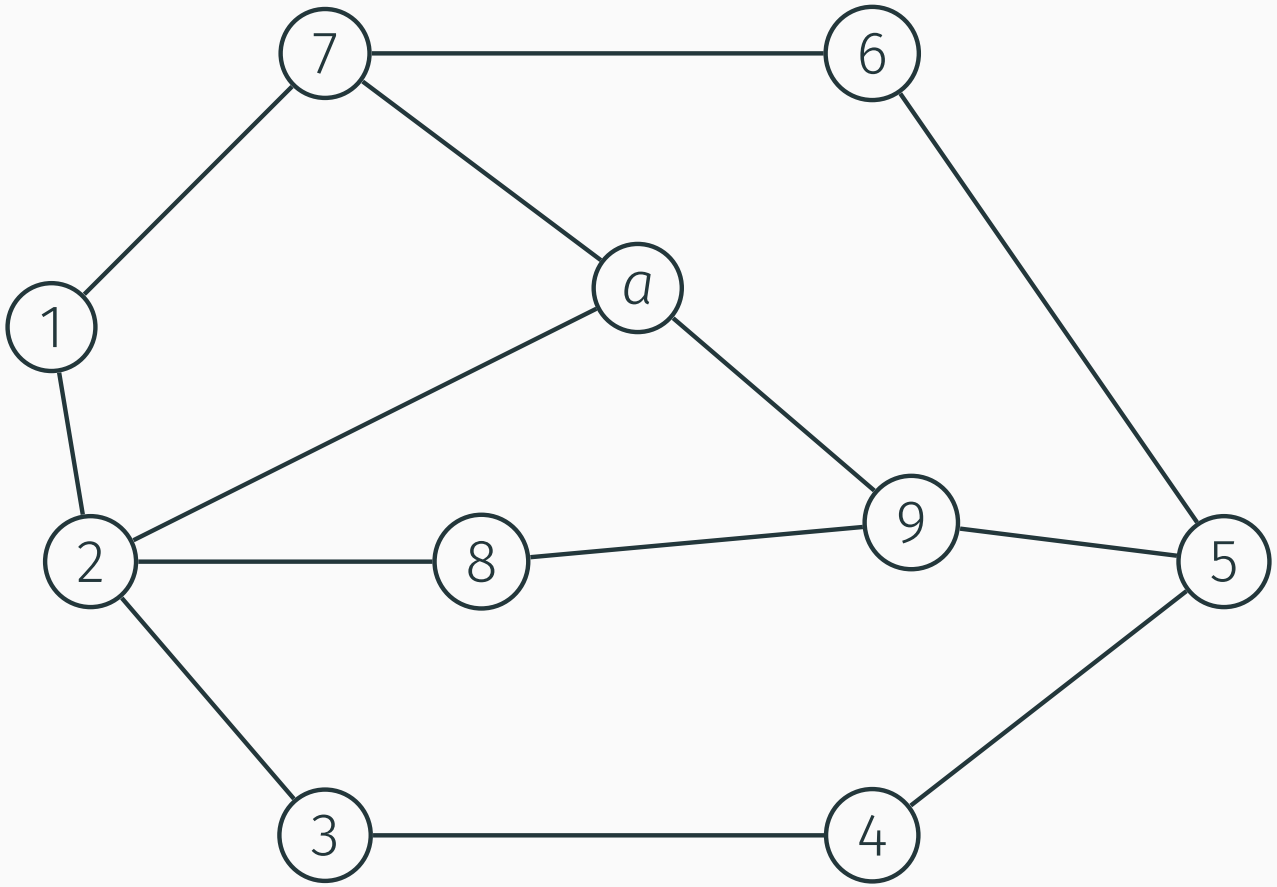
The matrix satisfies the sufficient conditions for total unimodularity. We can find the optimum of (P) by solving its linear relaxation $(0 \leq x_{ij} \leq 1)$ — The problem is **polynomial**.

Shortest-path — The integer linear program we found for the shortest-path problem satisfies the sufficient conditions.

$$
\begin{aligned}
\text{min.} \quad & \sum_{(i,j)\in A} x_{ij} c_{ij} \\
\text{s.t.} \quad & \sum_{(s,k)\in A} x_{sk} - \sum_{(k,s)\in A} x_{ks} = 1 \\
& \sum_{(i,k)\in A} x_{ik} - \sum_{(k,i)\in A} x_{ki} = 0, && \forall i \in V \setminus \{s,t\} \\
& \sum_{(t,k)\in A} x_{tk} - \sum_{(k,t)\in A} x_{kt} = -1 \\
& x_{ij} \in \{0,\ 1\}, && \forall (i,j) \in A
\end{aligned}
$$

We can replace constraints $x_{ij} \in \{0,\ 1\}$ by $0 \leq x_{ij} \leq 1$ and solve the resulting linear program in polynomial time.

Shortest-path — The integer linear program we found for the shortest-path problem satisfies the sufficient conditions.

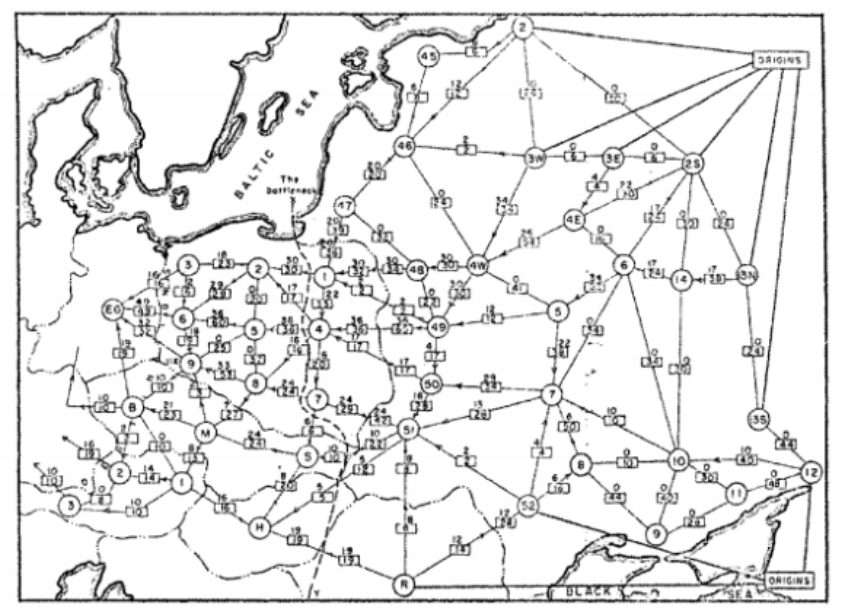


Shortest-path — The integer linear program we found for the shortest-path problem satisfies the sufficient conditions.

x_{12}	x_{17}	x_{21}	x_{23}	x_{28}	x_{2a}	x_{32}	x_{34}	x_{43}	x_{45}	x_{54}	x_{56}	x_{59}	x_{65}	x_{67}	x_{71}	x_{76}	x_{7a}	x_{82}	x_{89}	x_{95}	x_{98}	x_{9a}	x_{a2}	x_{a7}	x_{a9}
1	1	−1	0	0	0	0	0	0	0	0	0	0	0	0	−1	0	0	0	0	0	0	0	0	0	0
−1	0	1	1	1	1	−1	0	0	0	0	0	0	0	0	0	0	0	−1	0	0	0	0	−1	0	0
0	0	0	−1	0	0	1	1	−1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	−1	1	1	−1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	−1	1	1	1	−1	0	0	0	0	0	0	−1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	−1	0	1	1	0	−1	0	0	0	0	0	0	0	0	0
0	−1	0	0	0	0	0	0	0	0	0	0	0	0	−1	1	1	1	0	0	0	0	0	0	−1	0
0	0	0	0	−1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	−1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	−1	0	0	0	0	0	0	−1	1	1	1	0	0	−1
0	0	0	0	0	−1	0	0	0	0	0	0	0	0	0	0	0	−1	0	0	0	0	−1	1	1	1

Each colum contains exactly one 1 and one −1, so the set M_1 contains all the rows and the set M_2 is empty.

A transportation network is represented by a directed graph $\mathcal{G} = (V, A)$. To each node $i \in V$ is associated a value b_i representing supplies ($b_i > 0$), transit ($b_i = 0$) or demands ($b_i < 0$). Arcs $(i, j) \in A$ represent transport resources with limited capacities h_{ij} and unit cots c_{ij}. What is the least expensive plan that satisfies all demands?



Railway network — A. N. Tolstoi (1930s)

Minimum-cost flow — A transportation network is represented by a directed graph $\mathcal{G} = (V, A)$. To each node $i \in V$ is associated a value b_i representing supplies ($b_i > 0$), transit ($b_i = 0$) or demands ($b_i < 0$). Arcs $(i, j) \in A$ represent transport resources with limited capacities h_{ij} and unit cots c_{ij}. What is the least expensive plan that satisfies all demands?

- x_{ij} — Variable indicating the quantity that pass through arc (i, j).

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

The matrix is totally unimodular, if the demands and capacities are integral, the resulting flow will be integral and the linear relaxation will provide an integral solution.

Solving Integer Linear Programs

Branch & Bound

Branch & Bound — Basic idea

Consider the following optimization problem:

$$(P) \ : \ \max z = f(x), \ x \in \mathcal{X}$$

We can separate the set of feasible solutions $\mathcal{X}$ into k multiple (disjoint) sets $\mathcal{X}_i$:

$$\bigcup_{i=0}^{k} \mathcal{X}_i = \mathcal{X} \quad \text{and} \quad \mathcal{X}_i \cap \mathcal{X}_j = \emptyset, \forall i, j \in \{1, \ \ldots, \ k\}$$

We obtain a set (P_i) of problems — If we can find the optimal solution z_i of each problem (P_i), we can compute the optimal solution of (P):

$$z = \max_i z_i$$

Branch & Bound — Basic idea

Solving all the (P_i) is as hard as solving the original problem (P). How can we reduce the number of problems to solve?

- Maintain the best integral solution x^* we found (best objective value) — Update it each time a new integral solution is found.
- Solve a relaxation for each (P_i):
  - If the relaxation is worse than the current best solution, drop (P_i).
  - Otherwise, re-apply the branch & bound procedure starting with (P_i).

Drop problems that cannot lead to an optimal solution, re-apply the branch & bound procedure on problems that may lead to an optimal solution and update the best solution when an integral solution is found.

Algorithm Generic branch & bound algorithm — Version 1

```
procedure Branch & Bound(P)
    x* ← −∞                                  ▷ Current best solution (assume maximization).
    L ← {P}                                  ▷ List of problems to solve.
    while L ≠ ∅ do
        P ← Pop(L)                           ▷ Remove one of the problem from L.
        x_R ← Solve(Relax(P))                ▷ Solve a relaxation.
        if x_R > x* then                     ▷ If the relaxation is better than the current solution.
            if x_R ∈ ℕ^n then                ▷ If the solution is integral...
                x* ← x_R                     ▷ ...update the current solution.
            else
                L ← L ∪ Separate(P)          ▷ Otherwise split the problem.
            end if
        end if
    end while
    return x*
end procedure
```

Algorithm Generic branch & bound algorithm — Version 2

```
procedure Branch & Bound(P)
    x* ← −∞                                   ▷ Current best solution (assume maximization).
    L ← {P}                                   ▷ List of problems to solve.
    while L ≠ ∅ do
        P ← Pop(L)                            ▷ Remove one of the problem from L.
        for each P_i ∈ Separate(P) do
            x_R ← Solve(Relax(P_i))           ▷ Solve a relaxation for (P_i).
            if x_R > x* then                  ▷ If the relaxation is better than the current solution.
                if x_R ∈ ℕ^n then             ▷ If the solution is integral...
                    x* ← x_R                  ▷ ...update the current solution.
                else
                    L ← L ∪ {P_i}             ▷ Otherwise add the problem to L.
                end if
            end if
        end for
    end while
    return x*
end procedure
```

Branch & Bound — What can we customize?

There are **three** parts of the generic algorithm that are not specified and that we can customize in order to obtain an efficient algorithm:

1. Which problem should be solved next? — $Pop(L)$
   - Different data structures: queue for breadth-first search, stack for depth-first-search, ...
   - "Best" node first: priority-queue with a custom evaluation function.
   - Custom branching strategy: problem-specific, heuristics, ...
2. Which relaxation to use?
3. How to separate a problem?

We also want to quickly find a solution in order to be able to prune the set of problems:

- Starting with $\pm\infty$ is generally not efficient.
- We can use a depth-first search approach at the beginning to quickly converge to a solution.
- We can use a custom heuristic to find a solution before starting the algorithm.

Branch & Bound — Branching on binary variables

Consider a problem with a set x of binary variables — $x \in \{0,\ 1\}$.

We can branch:

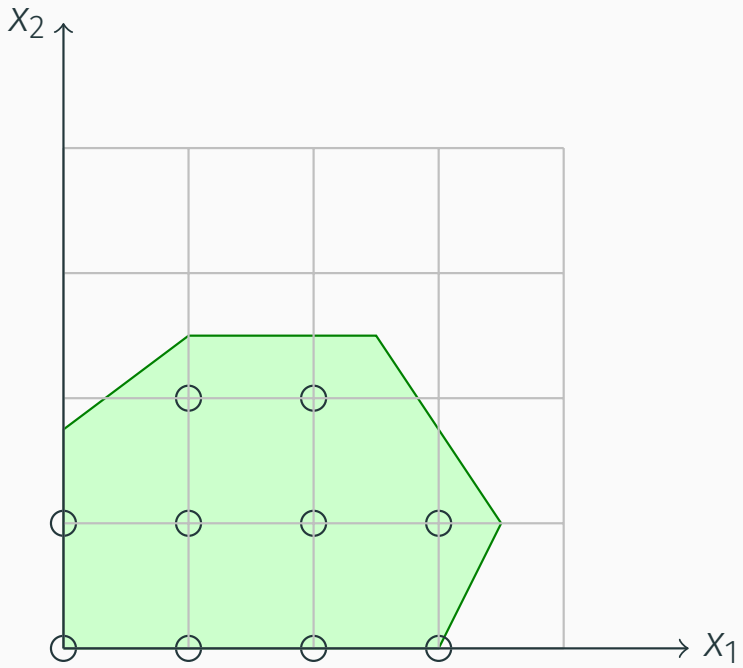
- On a single variable — $x_i = 0$ and $x_i = 1$.
- On a pair of variables — $(x_i,\ x_j) = (0, 0)$, $(x_i,\ x_j) = (0, 1)$, $(x_i,\ x_j) = (1, 0)$ and $(x_i,\ x_j) = (1, 1)$.
- On a n-tuples of variables.

Branching on multiple binary variables does not reduce the number of branches but it may sometimes be useful:

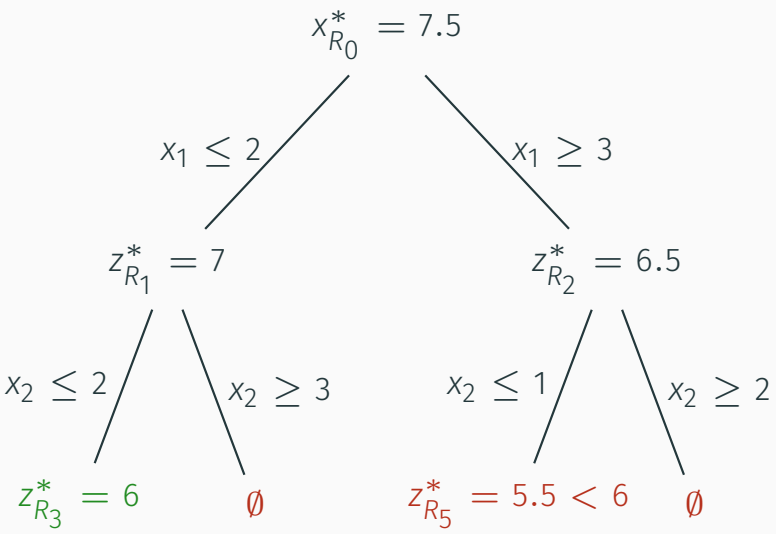
- We often find better bounds when branching on multiple variables.
- Branching on a single variable may not update the bound of the current branch.

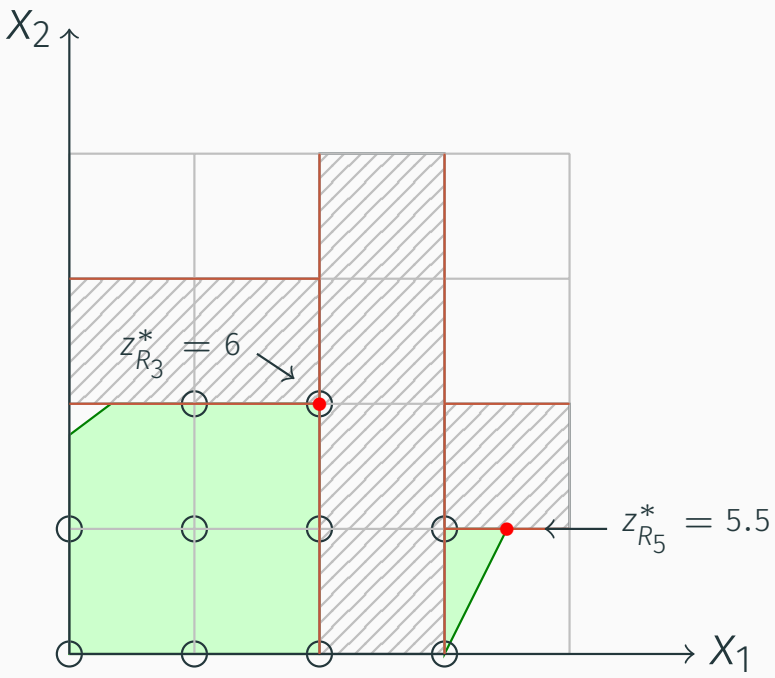
Problem

$$
\begin{aligned}
\text{max.} \quad & z = x_1 + 2x_2 \\
\text{s.t.} \quad & -3x_1 + 4x_2 \leq 7 \\
& 2x_2 \leq 5 \\
& 6x_1 + 4x_2 \leq 25 \\
& 2x_1 - x_2 \leq 6 \\
& x_1,\ x_2 \in \mathbb{N}
\end{aligned}
$$



Branch & Bound — Example





$z^*_{R_5}$ is worse than the current best solution ($z^*_{R_3} = 6$) so it is not necessary to explore its branch further.

Branch & Bound — For integer linear programs

What is usually done for integer linear programs:

- Evaluate using linear relaxation ($x \in \mathbb{N}^n \to x \in \mathbb{R}^n$).
- Separate using intersecting planes — Find a fractional x_i in the current solution and set $x_i \leq k$ on one branch, $x_i \geq k + 1$ on the other.
- Add cuts to the problems: *Branch & Cut*

What solvers usually add:

- Pre-processing: delete redundant variables or constraints, reduce domains of variables, detect sub-problems (graph, cliques, covers, ...), propagate constraints, ...
- Branching: simulate branching to find the best branch, remember past branching information, combine both, ...
- Multi-threading, detect infeasibility, find multiple solutions, ...

Solving Integer Linear Programs

Cuts

Cuts — Introduction

Cuts — Add constraints to the linear relaxation (R) of (P) until the optimum of (P) is on a vertex of (R).

Basic idea:

- The set of feasible solutions of the original problem (P) is unchanged.
- The current optimal solution z_R^* of the linear relaxation (R) is excluded using new constraints (cuts).

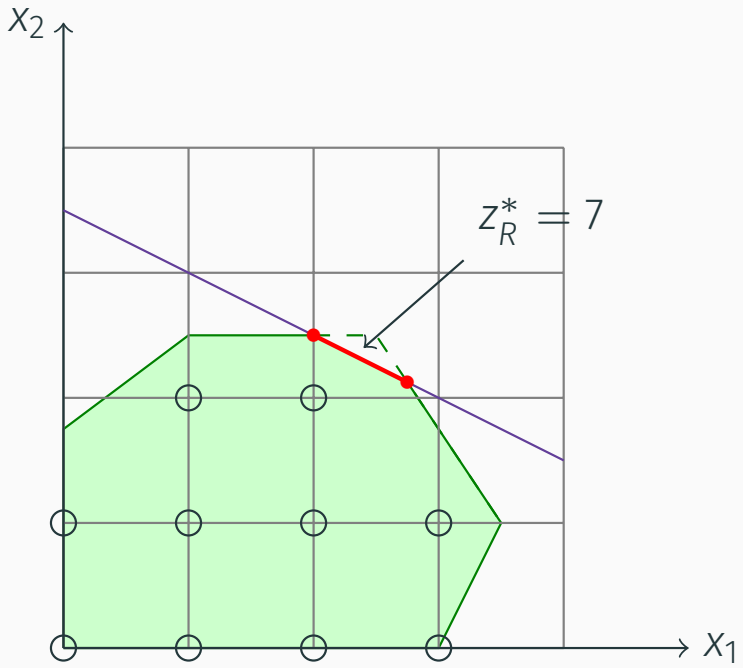
How to determine valid cuts?

1. Cuts from "rounding" objective function and constraints.
2. Cuts specific to a problem (clique, knapsack, ...).
3. Generic cuts: Gomory cuts.

Rounding cuts — Find cuts from rounding the right-hand side of existing constraints or the current value of the objective function.

Initial ILP (P):Adding a cut from the objective:

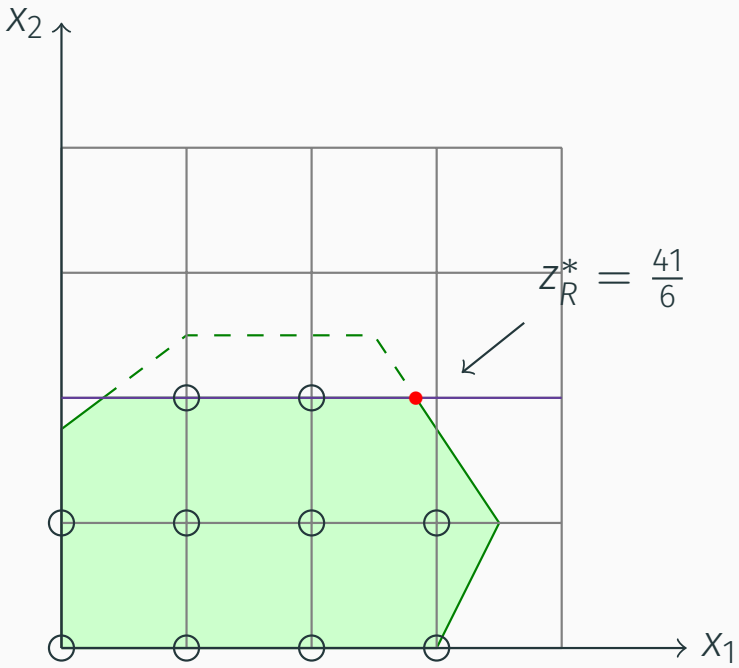
$$
\begin{aligned}
\text{max.} \quad & z = x_1 + 2x_2 \\
\text{s.t.} \quad & -3x_1 + 4x_2 \leq 7 \\
& 2x_2 \leq 5 \\
& 6x_1 + 4x_2 \leq 25 \\
& 2x_1 - x_2 \leq 6 \\
& x_1 + 2x_2 \leq \lfloor 7.5 \rfloor \\
& x_1,\ x_2 \in \mathbb{N}
\end{aligned}
$$



Rounding cuts — Find cuts from rounding the right-hand side of existing constraints or the current value of the objective function.

Initial ILP (P):Adding a cut from a constraint:

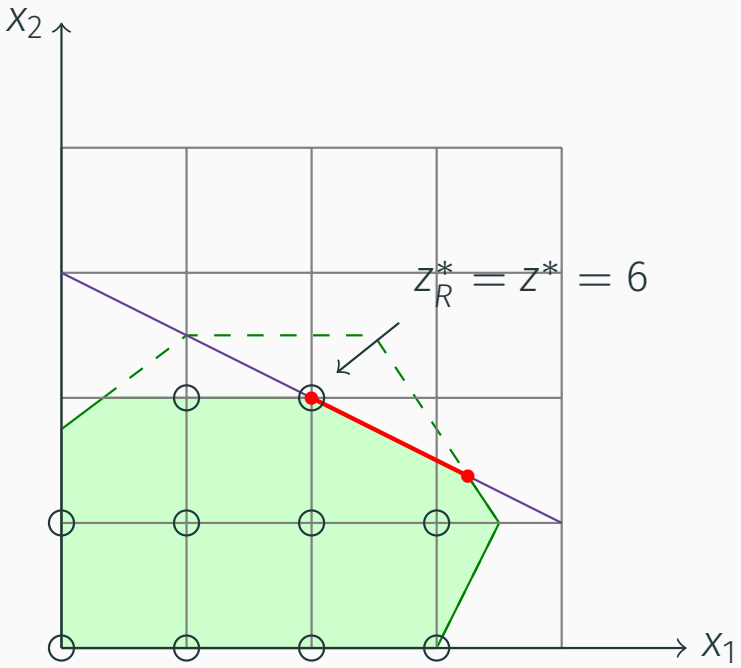
$$
\begin{aligned}
\text{max.} \quad & z = x_1 + 2x_2 \\
\text{s.t.} \quad & -3x_1 + 4x_2 \leq 7 \\
& x_2 \leq \lfloor \tfrac{5}{2} \rfloor \\
& 6x_1 + 4x_2 \leq 25 \\
& 2x_1 - x_2 \leq 6 \\
& x_1,\ x_2 \in \mathbb{N}
\end{aligned}
$$



Rounding cuts — Find cuts from rounding the right-hand side of existing constraints or the current value of the objective function.

Initial ILP (P):Adding a cut from the objective:

$$
\begin{aligned}
\text{max.} \quad & z = x_1 + 2x_2 \\
\text{s.t.} \quad & -3x_1 + 4x_2 \leq 7 \\
& x_2 \leq \lfloor \frac{5}{2} \rfloor \\
& 6x_1 + 4x_2 \leq 25 \\
& 2x_1 - x_2 \leq 6 \\
& x_1 + 2x_2 \leq \lfloor \frac{41}{6} \rfloor \\
& x_1,\ x_2 \in \mathbb{N}
\end{aligned}
$$



Problem-specific cuts — Clique cuts

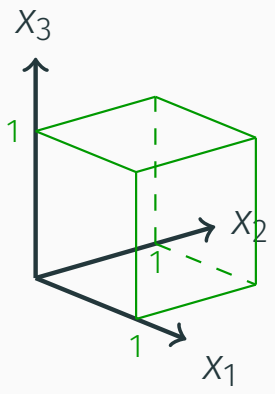
Clique: Set of **binary** variables $\mathcal{C} = \{x_1, \ \ldots, \ x_k\}$ which are mutually exclusive:

$$x_i + x_j \leq 1, \quad \forall x_i, x_j \in \mathcal{C}$$

For any clique $\mathcal{C}$ of size k the following inequality is **valid**:

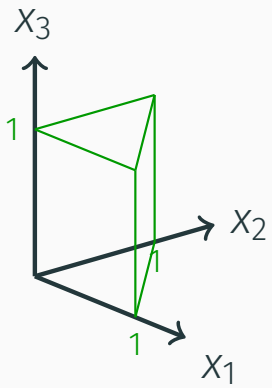
$$\sum_{i=1}^{k} x_i \leq 1$$

Example with 3 variables $x_1,\ x_2,\ x_3 \in \{0,\ 1\}$:

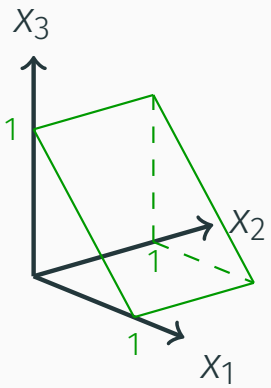


Domain of variables

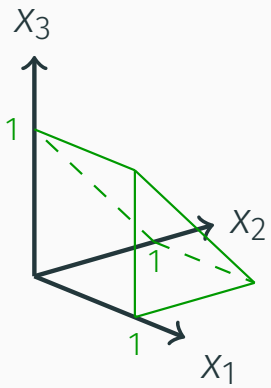
Example with 3 variables $x_1,\ x_2,\ x_3 \in \{0,\ 1\}$:



$x_1 + x_2 \leq 1$

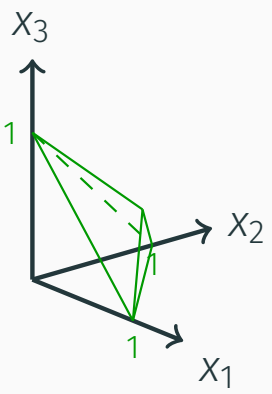


$x_1 + x_3 \leq 1$

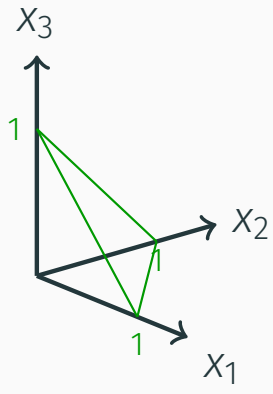


$x_2 + x_3 \leq 1$

Example with 3 variables $x_1,\ x_2,\ x_3 \in \{0,\ 1\}$:



3 exclusion constraints



Clique cut

Reminder: Knapsack problem

Given a set of n items associated to profits and weights, and a knapsack of capacity K, find the set of items that maximize the total profit while ensuring the capacity constraint.

$$\max \left\{ p^T X \mid w^T X \leq K,\ X \in \{0,\ 1\}^n \right\}$$

With:

- $p \in \mathbb{N}^n$ the profits of item.
- $w \in \mathbb{N}^n$ the weights of items.
- $K \in \mathbb{N}^+$ the capacity of the knapsack.

A minimal cover $\mathcal{C} \subseteq X$ is a subset of the variables such that if all variables in the subset were set to one, the knapsack constraint would be violated, but if any one was excluded, the constraint would be satisfied:

$$\sum_{x_i \in \mathcal{C}} w_i > K$$

$$\sum_{x_i \in \mathcal{C}} w_i - w_j \leq K, \quad \forall x_j \in \mathcal{C}$$

For any minimal cover $\mathcal{C}$, the following inequality is valid:

$$\sum_{x_i \in \mathcal{C}} x_i \leq |\mathcal{C}| - 1$$

This inequality is called a cover cut.

Travelling salesman problem — Given a list of n cities and the distances c_{ij} between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

Subtour formulation —

- x_{ij} — Binary variable indicating if we go directly from city i to city j.

$$\text{min.} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{6a}$$

$$\text{s.t.} \quad \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1, \qquad \forall i \in \{1, \ \ldots, \ n\} \tag{6b}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1, \qquad \forall j \in \{1, \ \ldots, \ n\} \tag{6c}$$

$$\sum_{i \in S} \sum_{\substack{j=1 \\ j \notin S}}^{n} x_{ij} \geq 1, \qquad \forall S \subset \{1, \ \ldots, \ n\}, \ S \neq \emptyset \tag{6d}$$

$$x_i \in \{0, \ 1\}, \qquad \forall i \in T \tag{6e}$$

- (6b) We leave each city exactly once.
- (6c) We enter each city exactly once.
- (6d) Subtour elimination.

Problem: There is an exponential number of subtour elimination constraints.

$$\sum_{i \in S} \sum_{\substack{j=1 \\ j \notin S}}^{n} x_{ij} \geq 1, \quad \forall S \subset \{1, \ \ldots, \ n\}, \ S \neq \emptyset$$

Solution: Generate constraints only when necessary:

Algorithm Travelling Salesman Problem

```
procedure TSP(𝒢 = (V, E))
    P ← RelaxTSP(𝒢)                          ▷ Create the problem wihout subtour constraints.
    while true do
        x ← Solve(P)
        if NumberOfSubtours(x) > 1 then
            P ← P ∪ SubtourConstraints(P, x)
        else
            return x
        end if
    end while
end procedure
```

For a graph $\mathcal{G}$, there exists a strict subset of subtour constraints that would be sufficient to reach the optimum. In practice, more constraints than necessary will be added — A trade-off needs to be found regarding the constraints we add to the model:

- if we add too many constraints, the ILP becomes larger and larger;
- if we do not add enough constraints, the algorithm will not converge.

The subtour formulation is often more efficient than the MTZ formulation because the relaxation of the MTZ formulation is pretty weak — In practice, it may be interesting to start with a relaxed problem (without subtour constraints), add subtour constraints up to a certain points (e.g. up to 1000 subtour constraints) and then add the arc-constraints and solve the final ILP.

Ref.: *Teaching Integer Programming Formulations Using the Traveling Salesman Problem, 2003*, Gábor Pataki — `http://epubs.siam.org/doi/pdf/10.1137/S00361445023685`

Cuts — How to find generic cuts?

How to find generic cuts?

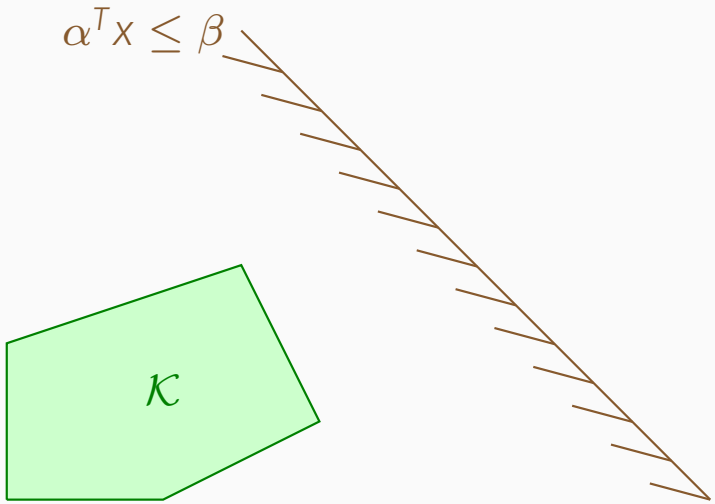
- Find hyperplanes that separate the current solution of the linear relaxation (if it is not integer) from the convex hull of $\mathcal{X}$.
- We need to find the most efficient cuts: cuts that allow the algorithm to converge to the solution as fast as possible.

Let $\mathcal{K} = \{x \in \mathbb{R}^n \ : \ Ax \leq b\}$ be a polyhedron in $\mathbb{R}^n$.

Definition

An inequality $\alpha^T x \leq \beta$ is **valid** for $\mathcal{K}$ if it is true for all $x \in \mathcal{K}$:

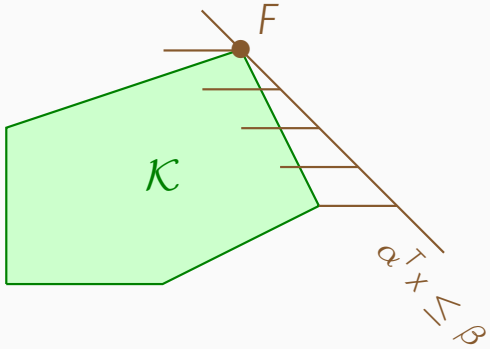
$$\mathcal{K} \subseteq \left\{x \in \mathbb{R}^n \ : \ \alpha^T x \leq \beta\right\}$$

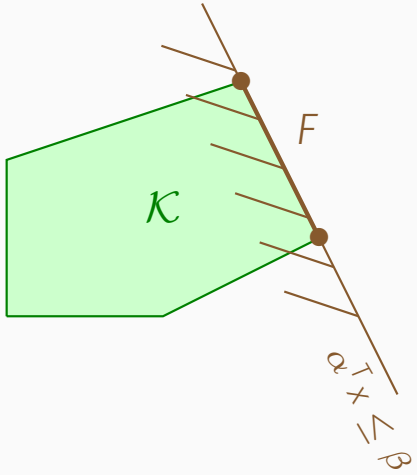


Definition

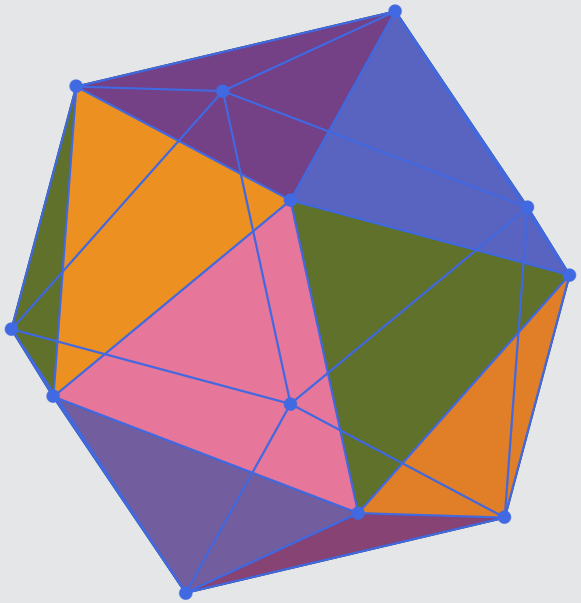
Let $\alpha^T x \leq \beta$ be a valid inequality for a polyhedron $\mathcal{P}$ in $\mathbb{R}^n$:

- $F = \left\{ x \in \mathcal{P} \ : \ \alpha^T x = \beta \right\}$ is a face of $\mathcal{P}$.
- A face F of dimension k ($F \subseteq \mathbb{R}^k$) is a k-face.
- A facet of a polyhedron $\mathcal{K}$ in $\mathbb{R}^n$ is a (n-1)-face.





Example



Let $\mathcal{P}$ represent this 3D-polyhedron:

- $\mathcal{P}$ itself is the only 3-face of $\mathcal{P}$ ($0^T x = 0,\ \forall x \in \mathcal{P}$).
- The triangles are the facets (or 2-faces) of $\mathcal{P}$.
- The edges are 1-faces of $\mathcal{P}$.
- The vertices are 0-faces of $\mathcal{P}$.
- The emptyset $\emptyset$ is the only (-1)-face of $\mathcal{P}$ (convention).

Theorem

Let $\mathcal{X}$ be a finite set of points in $\mathbb{R}^n$ and $x^0 \in \mathbb{R}^n \setminus conv(\mathcal{X})$. There exists $\alpha \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$ such that:

$$\begin{aligned} \alpha^T x &\leq \beta, \quad \forall x \in conv(\mathcal{X}) \\ \alpha^T x_0 &> \beta \end{aligned}$$

$\alpha^T x - \beta$ defines a **hyperplane** that separates x_0 from the convex hull $conv(\mathcal{X})$ of $\mathcal{X}$.

Let (P) be the following linear programs (with m constraints):

$$\max \left\{ c^T x \mid Ax \leq b,\ x \in \mathbb{N}^n \right\}$$

For all $1 \leq j \leq m$, $A_j x \leq b_j$ is a valid inequality for the convex hull of (P).

Chvatál-Gomory inequality

Consider $y \in \mathbb{R}_+^m$ and:

$$\alpha^T = \lfloor \sum_{j=1}^{m} y_j A_j \rfloor \in \mathbb{Z}^n$$

$$\beta = \lfloor \sum_{j=1}^{m} y_j b_j \rfloor \in \mathbb{Z}^n$$

Then $\alpha^T x \leq \beta$ is a valid inequality for the convex hull of (P).

In order to solve the linear relaxation (R) of an integer linear program (P), the simplex algorithm reformulate the problem by adding a set s of **slack** variables such that:

$$(R) = \max \left\{ c^T x \mid Ax + s = b,\ x \in \mathbb{R}^n,\ s \in \mathbb{R}^m \right\} = \max \left\{ c^T x \mid Du = b,\ u \in \mathbb{R}^{n+m} \right\}$$

The simplex algorithm proceeds by pivoting at each iteration, transforming the original system of linear equations into another one:

$$\overline{D}u = \overline{b},\ u \in \mathbb{R}^{n+m}$$

Consider constraint j of the final system (when the optimum of the linear relaxation has been found):

$$\sum_{i=1}^{n+m} \overline{d}_{ji} u_i = \overline{b}_j \tag{7}$$

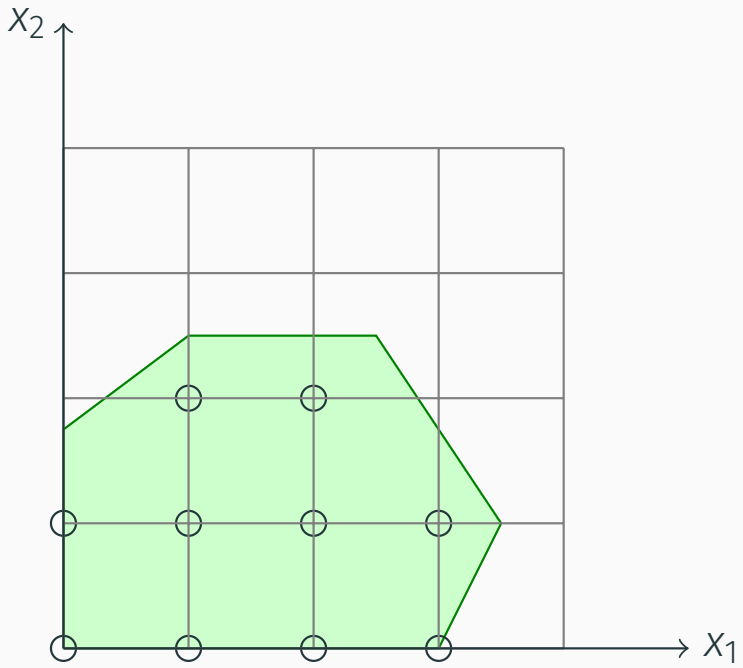
The following Gomory inequality is a **valid** inequality for the convex hull of (P):

$$\sum_{i=1}^{n+m} x_i \left(\overline{d}_{ji} - \lfloor \overline{d}_{ji} \rfloor\right) \geq \overline{b}_j - \lfloor \overline{b}_j \rfloor$$

We can create a **valid** Gomory inequality from (7) by dropping the integer parts (floor) of the coefficients $\overline{d}_{ji}$ and the right-hand-side $\overline{b}_j$, keeping only their **positive** fractional parts, and replacing the equality by an inequality $\geq$.

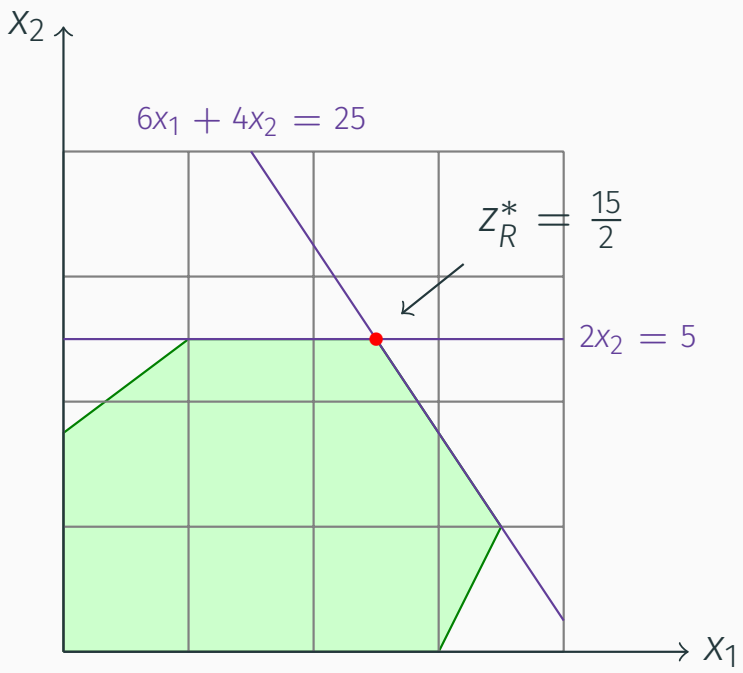
Problem

$$
\begin{aligned}
\text{max.} \quad & z = x_1 + 2x_2 \\
\text{s.t.} \quad & -3x_1 + 4x_2 \leq 7 \\
& 2x_2 \leq 5 \\
& 6x_1 + 4x_2 \leq 25 \\
& 2x_1 - x_2 \leq 6 \\
& x_1,\ x_2 \in \mathbb{N}
\end{aligned}
$$



Linear relaxation

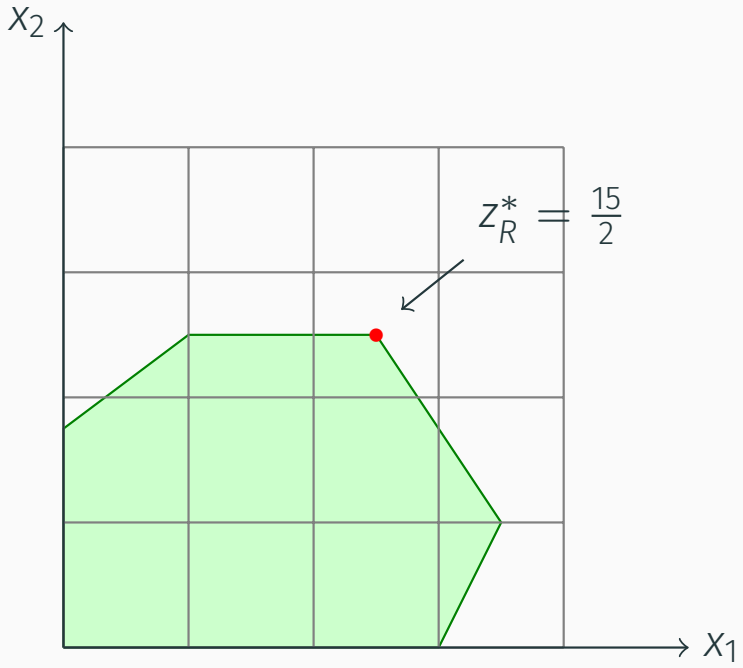
$$
\begin{aligned}
\text{max.} \quad & z = x_1 + 2x_2 \\
\text{s.t.} \quad & -3x_1 + 4x_2 + s_3 = 7 \\
& 2x_2 = 5 \\
& 6x_1 + 4x_2 = 25 \\
& 2x_1 - x_2 + s_6 = 6 \\
& x_1,\ x_2,\ s_3,\ s_4,\ s_5,\ s_6 \in \mathbb{R}^+
\end{aligned}
$$



The optimal solution of the relaxation is on the intersection of the second and third constraints, thus s_4 and s_5 are the non-basic variables and $s_4 = s_5 = 0$.

Final linear system (Simplex)

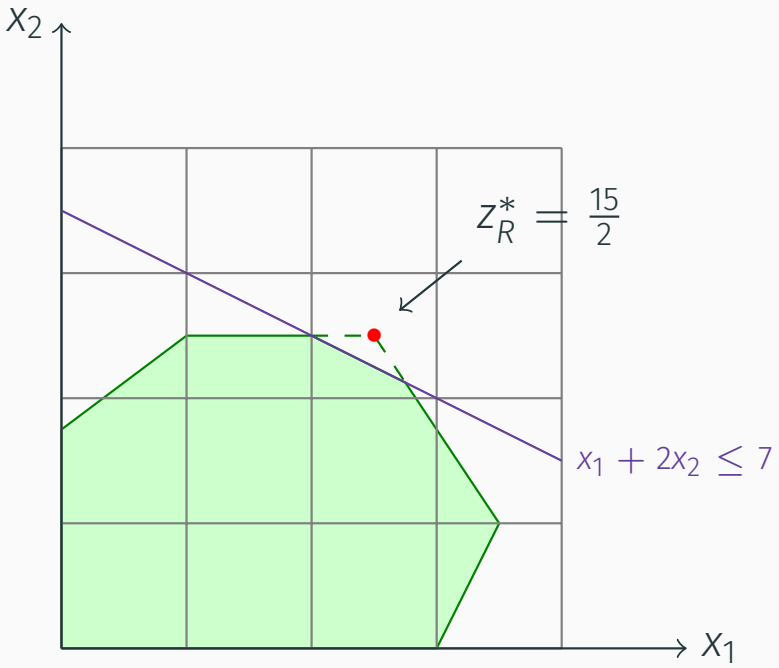
$$
\begin{aligned}
x_1 &= \frac{5}{2} + \frac{1}{3}s_4 - \frac{1}{6}s_5 \\
x_2 &= \frac{5}{2} - \frac{1}{2}s_4 + 0s_5 \\
s_3 &= \frac{9}{2} + 3s_4 - \frac{1}{2}s_5 \\
s_6 &= \frac{15}{2} - \frac{7}{6}s_4 + \frac{1}{3}s_5 \\
s_4 &= s_5 = 0
\end{aligned}
$$



Note: x_1, x_2, s_3, s_6 are the basic variables and s_4, s_5 are the non-basic variables.

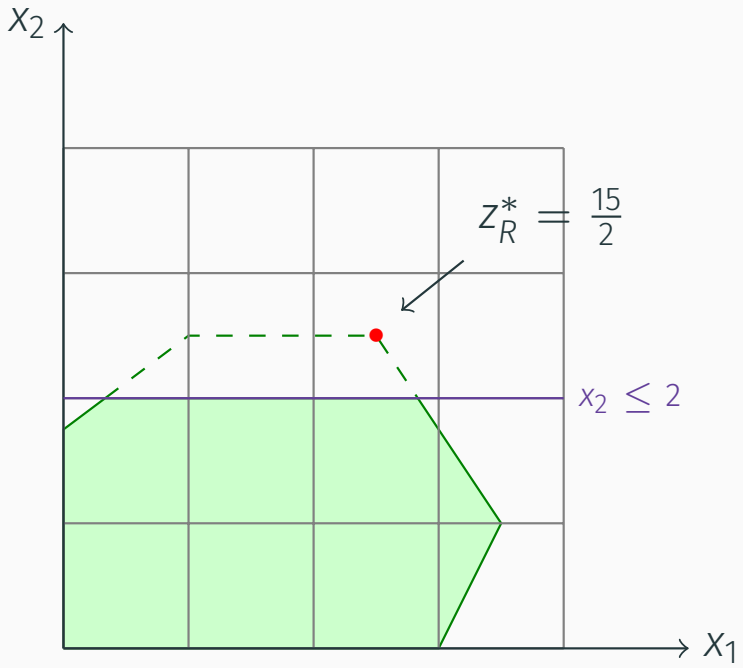
Gomory cut (1)

$$
\begin{aligned}
& s_4 = 5 - 2x_2 && (R) \\
& s_5 = 25 - 6x_1 - 4x_2 && (R) \\
& x_1 = \frac{5}{2} + \frac{1}{3}s_4 - \frac{1}{6}s_5 && (S) \\
\Leftrightarrow\ & x_1 - \frac{1}{3}s_4 + \frac{1}{6}s_5 = \frac{5}{2} \\
\Rightarrow\ & \frac{2}{3}s_4 + \frac{1}{6}s_5 \geq \frac{1}{2} \\
\Rightarrow\ & 4s_4 + s_5 \geq 3 \\
\Rightarrow\ & 45 - 6x_1 - 12x_2 \geq 3 \\
\Rightarrow\ & x_1 + 2x_2 \leq 7
\end{aligned}
$$



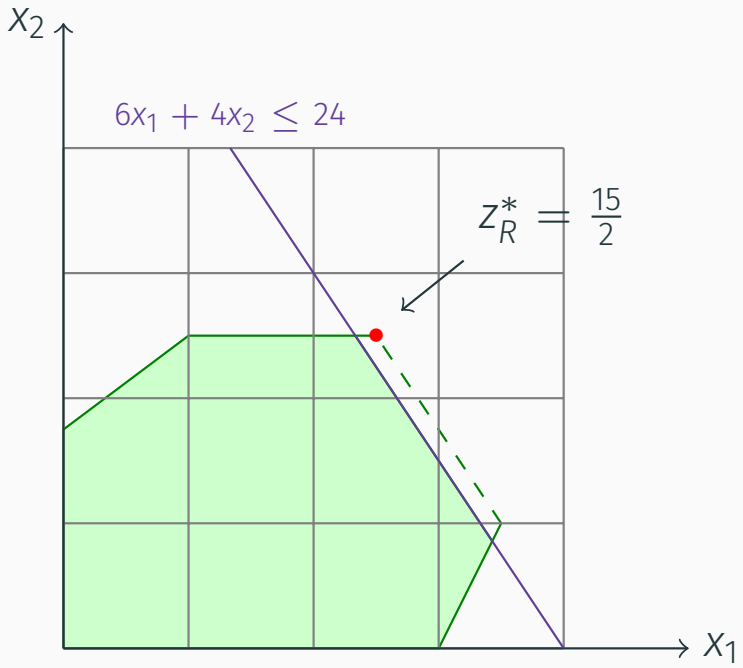
Gomory cut (2)

$$
\begin{aligned}
& s_4 = 5 - 2x_2 && (R) \\
& s_5 = 25 - 6x_1 - 4x_2 && (R) \\
& x_2 = \frac{5}{2} - \frac{1}{2}s_4 - 0s_5 && (S) \\
\Leftrightarrow \quad & x_2 + \frac{1}{2}s_4 = \frac{5}{2} \\
\Rightarrow \quad & \frac{1}{2}s_4 \geq \frac{1}{2} \;\Rightarrow\; s_4 \geq 1 \\
\Rightarrow \quad & 5 - 2x_2 \geq 1 \;\Rightarrow\; x_2 \leq 2
\end{aligned}
$$



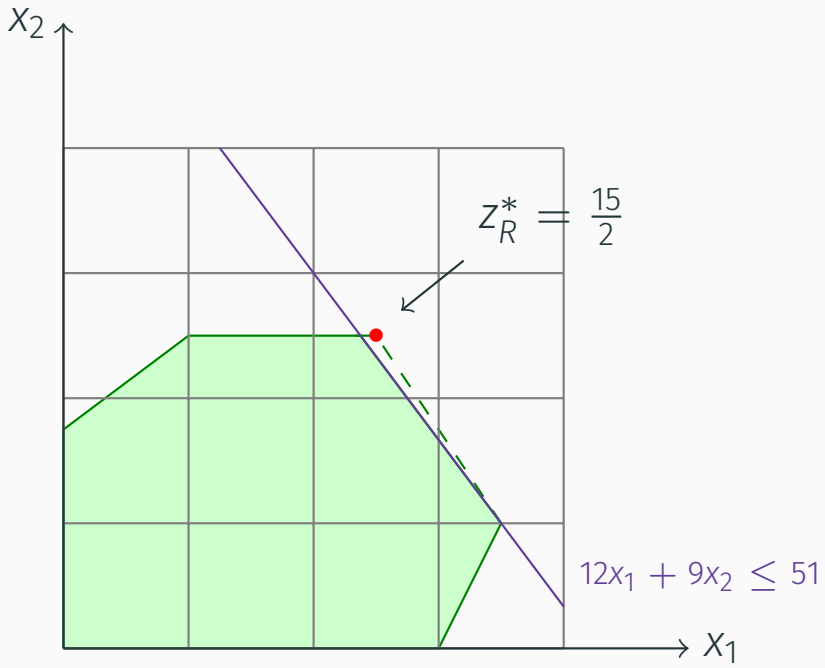
Gomory cut (3)

$$
\begin{aligned}
& s_4 = 5 - 2x_2 && (R) \\
& s_5 = 25 - 6x_1 - 4x_2 && (R) \\
& s_3 = \frac{9}{2} + 3s_4 - \frac{1}{2}s_5 && (S) \\
\Leftrightarrow \quad & s_3 - 3s_4 + \frac{1}{2}s_5 = \frac{9}{2} \\
\Rightarrow \quad & \frac{1}{2}s_5 \geq \frac{1}{2} \\
\Rightarrow \quad & s_5 \geq 1 \\
\Rightarrow \quad & 25 - 6x_1 - 4x_2 \geq 1 \\
\Rightarrow \quad & 6x_1 + 4x_2 \leq 24
\end{aligned}
$$



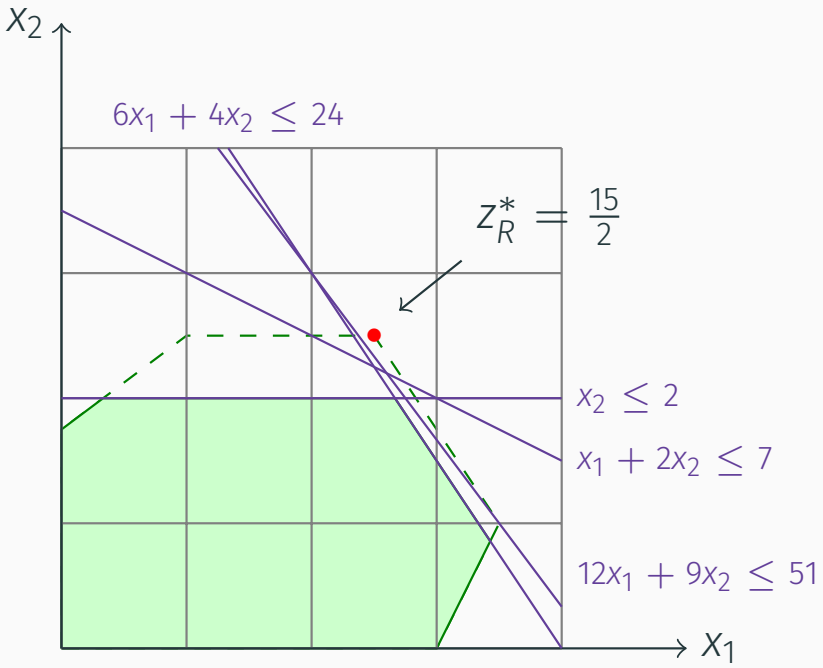
Gomory cut (4)

$$
\begin{aligned}
& s_4 = 5 - 2x_2 && (R) \\
& s_5 = 25 - 6x_1 - 4x_2 && (R) \\
& s_6 = \frac{7}{2} - \frac{7}{6}s_4 + \frac{1}{3}s_5 && (S) \\
\Leftrightarrow \quad & s_6 + \frac{7}{6}s_4 - \frac{1}{3}s_5 = \frac{7}{2} \\
\Rightarrow \quad & \frac{1}{6}s_4 + \frac{2}{3}s_5 \geq \frac{1}{2} \\
\Rightarrow \quad & s_4 + 4s_5 \geq 3 \\
\Rightarrow \quad & 105 - 24x_1 - 18x_2 \geq 3 \\
\Rightarrow \quad & 12x_1 + 9x_2 \leq 51
\end{aligned}
$$



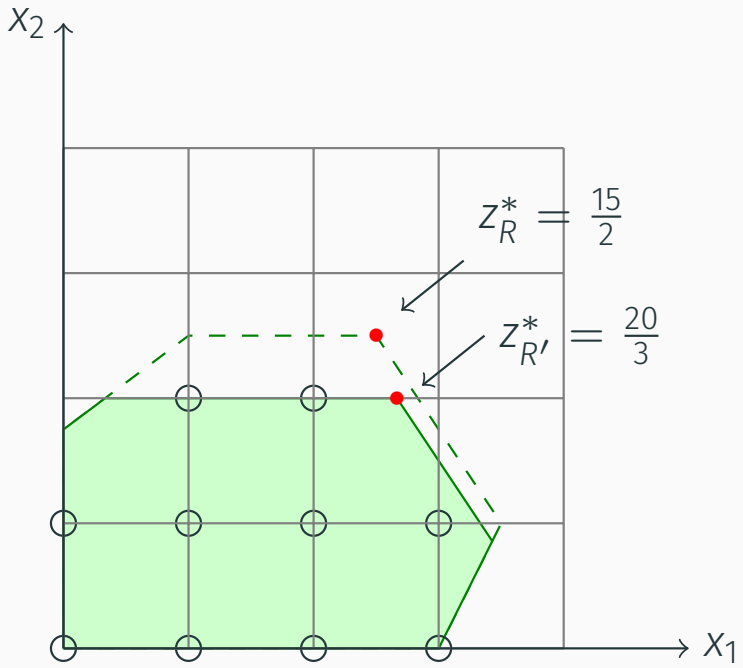
Gomory cut (1–4)

$$
\begin{aligned}
-3x_1 + 4x_2 &\leq 7 \\
2x_2 &\leq 5 \\
6x_1 + 4x_2 &\leq 25 \\
2x_1 - x_2 &\leq 6 \\
x_2 &\leq 2 \\
x_1 + 2x_2 &\leq 7 \\
6x_1 + 4x_2 &\leq 24 \\
12x_1 + 9x_2 &\leq 51 \\
x_1,\ x_2 &\in \mathbb{R}
\end{aligned}
$$



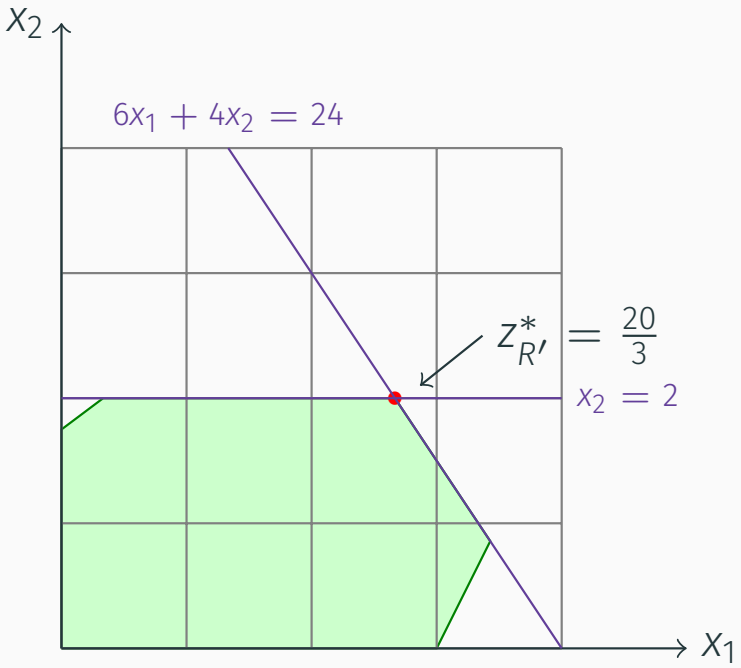
New ILP formulation

$$
\begin{aligned}
\text{max.} \quad & z = x_1 + 2x_2 \\
\text{s.t.} \quad & -3x_1 + 4x_2 \leq 7 \\
& 2x_1 - x_2 \leq 6 \\
& x_2 \leq 2 \\
& 6x_1 + 4x_2 \leq 24 \\
& x_1,\ x_2 \in \mathbb{N}
\end{aligned}
$$



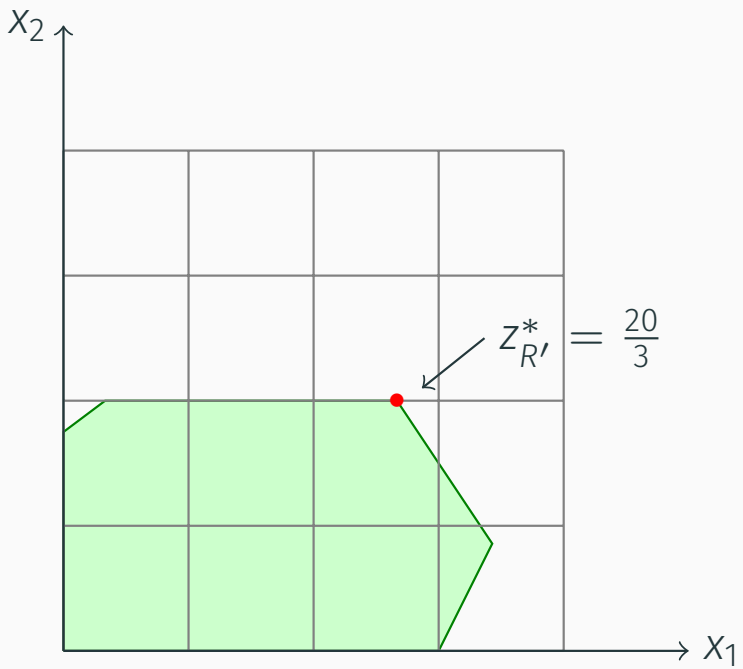
Linear relaxation

$$
\begin{aligned}
\text{max.} \quad & z = x_1 + 2x_2 \\
\text{s.t.} \quad & -3x_1 + 4x_2 + s_3 = 7 \\
& 2x_1 - x_2 + s_4 = 6 \\
& x_2 + s_5 = 2 \\
& 6x_1 + 4x_2 + s_6 = 24 \\
& x_1,\ x_2,\ s_3,\ s_4,\ s_5,\ s_6 \in \mathbb{R}^+
\end{aligned}
$$



Linear relaxation

$$x_1 = \frac{8}{3} + \frac{2}{3}s_5 - \frac{1}{6}s_6$$
$$x_2 = 2 - s_5 + 0s_6$$
$$s_3 = 7 + 6s_5 - \frac{1}{2}s_6$$
$$s_4 = \frac{8}{3} - \frac{7}{3}s_5 + \frac{1}{3}s_6$$
$$s_5 = s_6 = 0$$



x_2 and s_3 do not have fractional part so no cuts can be generated from the two corresponding constraints.

Linear relaxation

$$s_5 = 2 - x_2 \qquad (R)$$

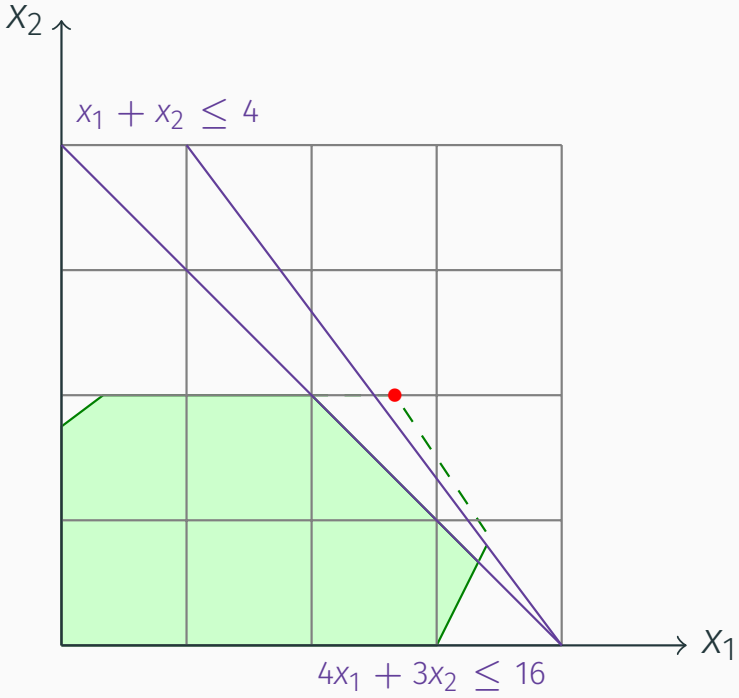
$$s_6 = 24 - 6x_1 - 4x_2 \qquad (R)$$

$$x_1 + x_2 \leq 4$$

$$x_2 = 2 - s_5 + 0s_6$$

$$s_3 = 7 + 6s_5 - \frac{1}{2}s_6$$

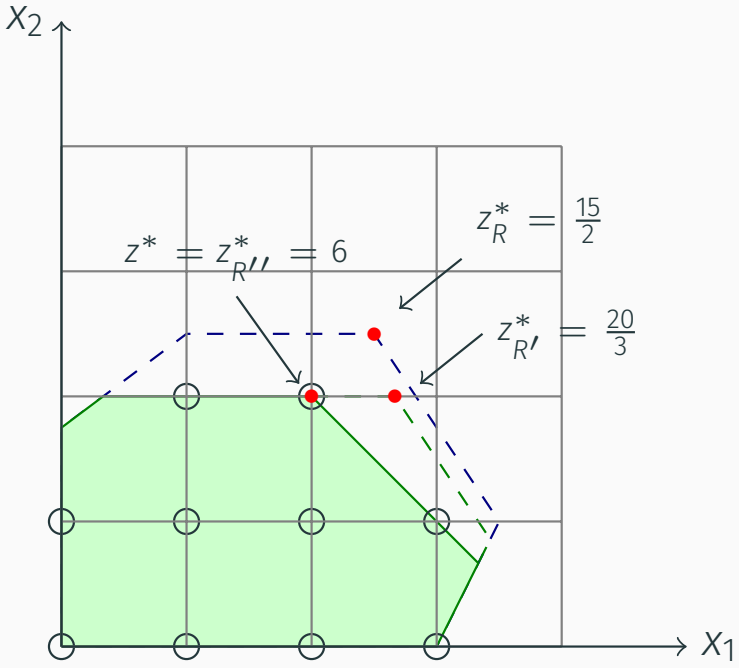
$$4x_1 + 3x_2 \leq 16$$



- $x_2 = 2 - s_5$ cannot be used to generate a new cut (all coefficients are already integers).
- $s_3 = 7 + 6s_5 - \frac{1}{2}s_6$ generates $s_6 \geq 0$, which is equivalent to $6x_1 + 4x_2 \leq 24$, which is already a constraint.

New ILP formulation

$$
\begin{aligned}
\text{max.} \quad & z = x_1 + 2x_2 \\
\text{s.t.} \quad & -3x_1 + 4x_2 \leq 7 \\
& 2x_1 - x_2 \leq 6 \\
& x_2 \leq 2 \\
& x_1 + x_2 \leq 4 \\
& x_1,\ x_2 \in \mathbb{N}
\end{aligned}
$$



We can stop because the optimal solution of the linear relaxation $z^*_{R''}$ is in $\mathbb{N}^2$.

How to generate cuts for generic programs? — Generic algorithm

Algorithm Generic cut-based algorithm

```
procedure CUTS(P)
    x ← null
    while x = null do
        x_R ← Simplex(Relax(P))                ▷ Apply Simplex to the linear relaxation.
        if x_R ∈ ℕ^n then                      ▷ Check if the solution is integral.
            x ← x_R                            ▷ The optimal solution has been found.
        else
            P ← P ∪ Gomory(x_R)                ▷ Add cuts to the problem.
        end if
    end while
    return x
end procedure
```

This algorithm is guaranteed to terminate in a finite number of steps, but it may takes a very long time to converge to the optimal solution.

Solving Integer Linear Programs

Decomposition methods

Decomposition methods

Integer linear programs can often be split in a **master problem** and a **subproblem** — These decompositions often depends on the form of the integer linear program.

- **Benders decomposition** — Constraints (rows) generation.
- **(Delayed) column generation** — Variables (columns) generation.
  - **Dantzig-Wolfe decomposition**

General idea:

1. Starts with a reduction of the original problem $(P) \rightarrow$ **Master problem**.
2. Solve the master problem.
3. Check the solution for optimality and feasibility in (P).
   - Check passed $\rightarrow$ The optimal solution has been found, **stop**.
   - Check failed $\rightarrow$ Create and solve the **subproblem** to generate new variables and/or constraints to add to the the **master problem**, and go bak to (2).

The reduced **master problem** and the **subproblem** must be "easy" to solve for the decomposition to be efficient.

Original problem (P)

$$\min \left\{ c^T x + f^T y \ : \ Ax + Fy = b, \ x \geq 0, \ y \in Y \right\}$$

Benders decomposition:

Master problem (MP)

$$\min \left\{ h(y) + f^T y \ : \ y \in Y \right\}$$

Primal subproblem $SP(\bar{y})$

$$h(\bar{y}) = \min \left\{ c^T x \ : \ Ax = b - F\bar{y}, \ x \geq 0 \right\}$$

Primal subproblem $SP(\bar{y})$

$$h(\bar{y}) = \min \left\{ c^T x \ : \ Ax = b - F\bar{y}, \ x \geq 0 \right\}$$

We need to select $\bar{y} \in Y$ such that $(SP(\bar{y}))$ admits feasible solutions:

- if $(SP(\bar{y}))$ is infeasible for all $\bar{y} \in Y$, then the original problem (P) is infeasible;
- if there exists $\bar{y}$ such that $SP(\bar{y})$ is unbounded, then the original problem (P) is unbounded;
- if (P) has an optimal solution (x^*, y^*), then x^* is an optimal solution of $(SP(y^*))$.

Primal subproblem $SP(\bar{y})$

$$h(\bar{y}) = \min \left\{ c^T x \ : \ Ax = b - F\bar{y}, \ x \geq 0 \right\}$$

We can use Farkas' lemma —

- Given $\bar{y} \in Y$, there exists $x \geq 0$ such that $Ax = b - F\bar{y}$ if and only if $u^T(b - F\bar{y}) \leq 0$ for all u such that $u^T A \leq 0$.

The polar cone $\{u \ : \ u^T A \leq 0\}$ is polyhedral and generated by a finite number of "rays" (vectors) $\rho_r, r \in \Gamma$. These vectors are independent from $\bar{y}$.

The primal subproblem $SP(\bar{y})$ is feasible if

$$\bar{y} \in \bar{Y} = \rho_r^T(b - F\bar{y}) \leq 0, \quad \forall r \in \Gamma.$$

$\bar{Y} \subseteq Y$ is the set of solutions from the master problem for which the primal subproblem is feasible.

Primal subproblem $SP(\bar{y})$

$$\min \left\{ c^T x \ : \ Ax = b - F\bar{y}, \ x \geq 0 \right\}$$

Dual subproblem $SD(\bar{y})$

$$\max \left\{ u^T (b - F\bar{y}) \ : \ u^T A \leq c, \ u \in \mathbb{R}^n \right\}$$

- The set of feasible solutions for $SD(\bar{y})$ does not depend on $\bar{y}$.
- The extreme rays of $SD(\bar{y})$ are the generative vectors of $\{u \ : \ u^T A \leq 0\}$, $\rho_r, r \in \Gamma$.
- The set of feasible solutions for $SD(\bar{y})$ has a finite number of extreme (corner) points $\omega_p, p \in \Omega$.

If $\bar{y} \in \bar{Y}$ — $(SP(\bar{y}))$ is feasible — then either

- $(SD(\bar{y}))$ is infeasible and $(SP(\bar{y}))$ is unbounded or;
- $(SD(\bar{y}))$ has a finite optimum, and does $(SP(\bar{y}))$.

We can reformulate (P) using the solution of $(SD(\bar{y}))$ [1] —

$$\min_{y \in \bar{Y}} \left\{ \max_{u} \left\{ u^T(b - Fy) \ : \ u^T A \leq c \right\} + f^T y \right\}$$

If $(SD(\bar{y}))$ has a solution, its own of its extreme points ω_p —

$$\min_{y \in \bar{Y}} \left\{ \max_{p \in \Omega} \left\{ \omega_p^T (b - Fy) \right\} + f^T y \right\}$$

Or —

$$\min_{\substack{y \in Y \\ \rho_r^T (b - Fy) \leq 0}} \left\{ \max_{p \in \Omega} \left\{ \omega_p^T (b - Fy) \right\} + f^T y \right\}$$

[1] We assign $-\infty$ to the $\max$ if $(SD(\bar{y}))$ is infeasible.

Benders master problem:

$$
\begin{aligned}
\text{min.} \quad & z + f^T y && && \text{(8a)}\\
\text{s.t.} \quad & z \geq \omega_p^T (b - Fy), && p \in \Omega && \text{(8b)}\\
& 0 \geq \rho_r^T (b - Fy), && r \in \Gamma && \text{(8c)}\\
& y \in S && && \text{(8d)}
\end{aligned}
$$

- (8b) — Optimality cuts.
- (8c) — Feasibility cuts, ensure that $(SP(y))$ is feasible.

There is a huge number of optimality (8b) and feasibility (8c) constraints — We start with a subset of these and add new ones when necessary.

Benders **master** problem (with only a subset of constraints):

$$\begin{aligned} \text{min.} \quad & z + f^T y && && (9a) \\ \text{s.t.} \quad & z \geq \omega_p^T (b - Fy), && p \in \Omega_1 && (9b) \\ & 0 \geq \rho_r^T (b - Fy), && r \in \Gamma_1 && (9c) \\ & y \in S && && (9d) \end{aligned}$$

1. Solve the **master** problem with subsets $\Omega_1 \subseteq \Omega$ and $\Gamma_1 \subseteq \Gamma$ of optimality and feasibility constraints $\rightarrow$ We obtain a new solution $\bar{y}$ which is a **lower bound** on the original problem (minimization problem).
2. Solve the dual **subproblem** $SD(\bar{y})$ using our new solution $\bar{y}$:
   - If the dual subproblem is **unbounded**, the primal subproblem is **infeasible**, so we can add new feasibility cuts to Γ_1 — We add the cuts corresponding to the extreme directions ρ_r along which the dual is unbounded.
   - Otherwise, we add a new optimal cut to Ω_1 using the current solution ω_p of the dual — If this cut is not violated by the current solution $\bar{y}$, then the optimal solution of the original problem has been found, otherwise we go back to (1).

Warehouse allocations — A company needs to supply a set of n clients and needs to open new warehouses (from a set of m possible warehouses). Opening a warehouse j costs f_j and supplying a client i from a warehouse j costs c_{ij} per supply unit. Which warehouses should be opened in order to satisfy all clients while minimizing the total cost?

$$
\begin{aligned}
\text{min.} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} + \sum_{j=1}^{m} f_j y_j & \\
\text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} = 1, & \forall i \in \{1, \ldots, n\} \\
& x_{ij} \leq y_j, & \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\} \\
& y_j \in \{0, 1\}, & \forall j \in \{1, \ldots, m\} \\
& x_{ij} \geq 0, & \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}
\end{aligned}
$$

Primal subproblem — $SP(\bar{y})$

$$\begin{aligned}
\text{min.} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \\
\text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} = 1, && \forall i \in \{1, \ldots, n\} \\
& x_{ij} \leq \bar{y}_j, && \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\} \\
& x_{ij} \geq 0, && \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}
\end{aligned}$$

Dual subproblem — $SP(\bar{y})$

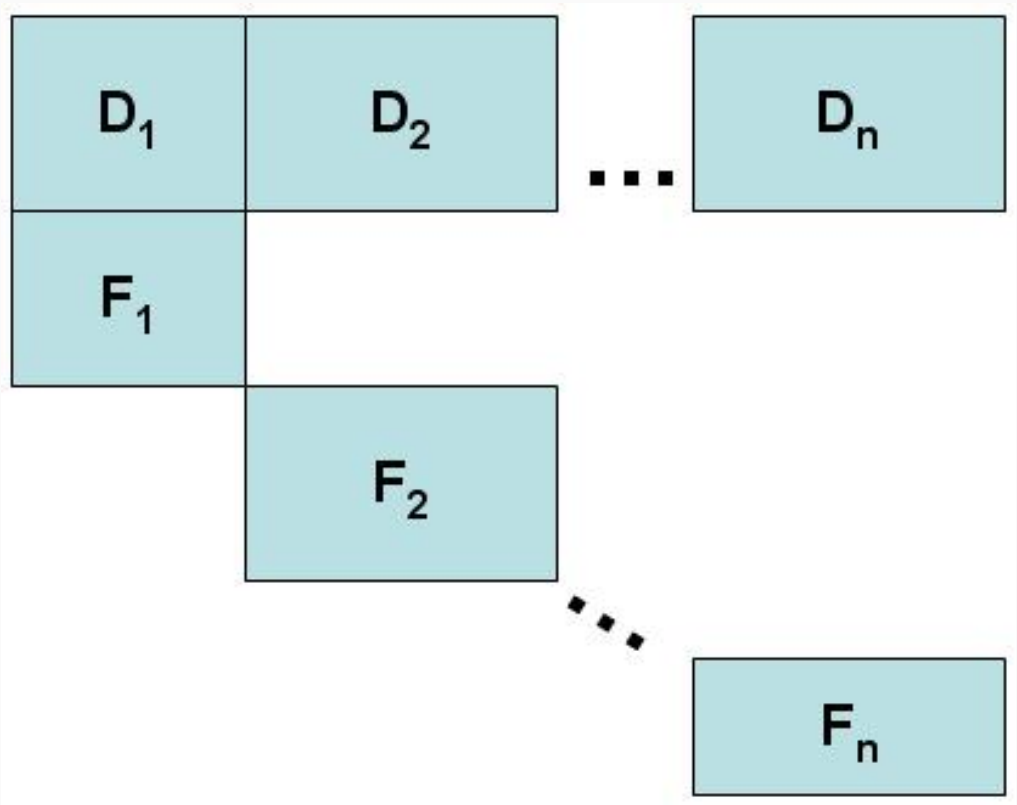
$$\begin{aligned}
\text{max.} \quad & \sum_{i=1}^{n} v_i - \sum_{i=1}^{n}\sum_{j=1}^{m} \bar{y}_j u_{ij} \\
\text{s.t.} \quad & v_i - u_{ij} \leq c_{ij}, && \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\} \\
& v_i \in \mathbb{R},\ u_{ij} \geq 0 && \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}
\end{aligned}$$

Master problem —

$$
\begin{aligned}
\text{min.} \quad & z + \sum_{j=1}^{m} f_j y_j \\
\text{s.t.} \quad & z \geq \sum_{i=1}^{n} v_i^p - \sum_{i=1}^{n} \sum_{j=1}^{m} u_{ij}^p y_j, && \forall p \in l_1 \\
& 0 \geq \sum_{i=1}^{n} v_i^r - \sum_{i=1}^{n} \sum_{j=1}^{n} u_{ij}^r y_j, && \forall r \in l_2 \\
& y_j \in \{0, 1\}, && \forall j \in \{1, \ldots, m\}
\end{aligned}
$$

- Adding optimality cuts is easy — We can retrieve v_i^p and u_{ij}^p from the solution of the dual subproblem.
- Adding feasibility cuts by hand is slightly more difficult — We need to find extreme directions along which the dual is unbounded, this may require solving another LP.

Dantzig-Wolfe decomposition can be used on integer linear program whose constraint matrix has the following block form:



The D matrix represent the coupling constraints and each F_i matrix represents an independent sub-matrix.

The original problem is reformulated into a master problem and n sub-problems — The number of variables (columns) in the master is typically exponential (each variable represents a solution of a subproblem).

The idea behind the Dantzig-Wolfe decomposition (and other column-generation methods) is that most variables will have a value of 0 in the optimal solution — We can start with a subset of the variables (columns) in the master problem and use solution(s) from the subproblem(s) to add new columns to the master problem as we go along.

- Dantzig-Wolfe decomposition is a "generic" methods that can be used on any integer linear program whose constraint matrix has the correct form.
- There are problem-specific Column-generation method — We need to define columns (variables), sub-problems, and the way we generate columns from sub-problems.

Vehicle Routing Problem — Given a fleet of vehicles, a depot and a set of customers, what is the optimal set of routes that deliver to every customer?

- $y_r \in \{0,\ 1\}$ — $y_r = 1$ if route r is used in the optimal solution.

Let c_r be the cost of route r and $\alpha_{ir} = 1$ if customer i is served by route r.

$$
\begin{aligned}
\text{min.} \quad & \sum_{r=1}^{R} c_r y_r \\
\text{s.t.} \quad & \sum_{r=1}^{R} \alpha_{ir} y_r \geq 1, && \forall i \in \{1,\ \ldots,\ n\} \\
& y_r \in \{0,\ 1\}, && \forall j \in \{1,\ \ldots,\ R\}
\end{aligned}
$$

There is an exponential number of routes (columns) $\rightarrow$ We cannot enumerate all of them and solve the full program, we need to generate them as we go along.

Bin Packing Problem — Given a set of n items with associated weights w_i, and a set of n bins of capacity W, what is the packing of items into bins that minimizes the number of bins used?

- $x_{ik} - x_{ik} = 1$ if item i is put inside bin k.
- $y_k - y_k = 1$ if bin k is used.

$$
\begin{aligned}
\text{min.} \quad & \sum_{k=1}^{n} y_k \\
\text{s.t.} \quad & \sum_{k=1}^{n} x_{ik} \geq 1, && \forall i \in \{1, \ \ldots, \ n\} \\
& \sum_{i=1}^{n} w_i x_{ik} \leq W y_k, && \forall k \in \{1, \ \ldots, \ n\} \\
& x_{ik}, y_k \in \{0, \ 1\}, && \forall i, k \in \{1, \ \ldots, \ n\}
\end{aligned}
$$

Bin Packing Problem — Given a set of n items with associated weights w_i, and a set of n bins of capacity W, what is the packing of items into bins that minimizes the number of bins used?

If we know all possible packing of any bin, we can reformulated the problem with an exponential number of variables.

A column x^t is a feasible solution of the (knapsack-like) problem:

$$\left\{ x^t \in \{0,\ 1\}^n \ : \ \sum_{i=1}^{n} w_i x_i^t \leq W \right\}$$

Let us assume there are T possible packings. For a packing t, $x_i^t = 1$ if item i is in the packing t.

Bin Packing Problem — Given a set of n items with associated weights w_i, and a set of n bins of capacity W, what is the packing of items into bins that minimizes the number of bins used?

- $\lambda_t - \lambda_t = 1$ if packing t is chosen.

$$
\begin{aligned}
\text{min.} \quad & \sum_{t=1}^{T} \lambda_t \\
\text{s.t.} \quad & \sum_{t=1}^{T} x_i^t \lambda_t \geq 1, && \forall i \in \{1, \ \ldots, \ n\} \\
& \lambda_t \in \{0, \ 1\}, && \forall t \in \{1, \ \ldots, \ T\}
\end{aligned}
$$

The associated subproblem (used to generate new columns) is a well-known problem.

Column generation — Bin Packing Problem example

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

We start with a subset of possible packings — All packings that only put a single item per bin.

$$\begin{aligned} \text{min.} \quad & \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 \\ \text{s.t.} \quad & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \end{pmatrix} \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \end{aligned}$$

Optimal solution: $z_0 = 5$, $\lambda = (1, 1, 1, 1, 1)$.

The dual of the master problem (with S columns) is:

$$\begin{aligned} \max. \quad & \sum_{i=1}^{n} \pi_i & \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i^s \pi_i \leq 1 & 1 \leq s \leq S \\ & \pi_i \geq 0 & 1 \leq i \leq n \end{aligned}$$

We want to find a column λ_s with negative reduced cost, i.e. whose corresponding constraints in the dual is violated:

$$\sum_{i=1}^{n} x_i^s \pi_i > 1 \iff \sum_{i=1}^{n} x_i^s \pi_i - 1 > 0$$

To find the column with maximum reduced cost, we want to solve:

$$\begin{aligned} \max. \quad & \sum_{i=1}^{n} x_i \pi_i^* - 1 \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0,\ 1\} \end{aligned}$$

At each step, we get a new solution for the **primal master problem** and thus new values for the **dual variables** π^*. We can then construct a new **subproblem** from these dual values π^*. If the optimum of the **subproblem** is 0 (or negative), there is no column with reduced negative cost.

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

The optimal dual solution for z_0 is $\pi^* = (1, 1, 1, 1, 1)$, thus the Knapsack subproblem associated with the solution is:

$$\begin{aligned} \text{max.} \quad & x_1 + x_2 + x_3 + x_4 + x_5 - 1 \\ \text{s.t.} \quad & x_1 + 2x_2 + 2x_3 + 3x_4 + 4x_5 \leq 6 \\ & x_i \in \{0,\ 1\} \end{aligned}$$

Optimal solution of the subproblem: $z_0^{SP} = 2$ — This generate a new column for our master problem $x = (1, 1, 1, 0, 0)$.

Column generation — Bin Packing Problem example

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

We add the new column to the master problem:

$$\text{min.} \quad \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6$$

$$\text{s.t.} \quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \lambda \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Optimal solution: $z_1 = 3$, $\lambda = (0, 0, 0, 1, 1, 1)$.

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

Knapsack subproblem associated with the solution[2]:

$$\begin{aligned} \max. \quad & x_3 + x_4 + x_5 - 1 \\ \text{s.t.} \quad & 2x_3 + 3x_4 + 4x_5 \leq 6 \\ & x_i \in \{0,\ 1\} \end{aligned}$$

Optimal solution of the subproblem: $z_1^{SP} = 1$ — This generate a new column for our master problem $x = (0, 0, 1, 1, 0)$.

[2] We can drop the x_i variables not in the objective from the constraint.

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

We add the new column to the master problem:

$$\min. \quad \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7$$

$$\text{s.t.} \quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \lambda \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Optimal solution: $z_2 = 3$, $\lambda = (0, 0, 0, 1, 1, 1, 0)$.

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

Knapsack subproblem associated with the solution:

$$\begin{aligned} \text{max.} \quad & x_1 + x_4 + x_5 - 1 \\ \text{s.t.} \quad & x_1 + 3x_4 + 4x_5 \leq 6 \\ & x_i \in \{0,\ 1\} \end{aligned}$$

Optimal solution of the subproblem: $z_2^{SP} = 1$ — This generate a new column for our master problem $x = (1, 0, 0, 1, 0)$.

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

We add the new column to the master problem:

$$\text{min.} \quad \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8$$

$$\text{s.t.} \quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \lambda \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Optimal solution: $z_3 = 3$, $\lambda = (0, \frac{1}{2}, 0, 0, 1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$.

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

Knapsack subproblem associated with the solution:

$$\begin{aligned} \text{max.} \quad & x_2 + x_4 + x_5 - 1 \\ \text{s.t.} \quad & 2x_2 + 3x_4 + 4x_5 \leq 6 \\ & x_i \in \{0,\ 1\} \end{aligned}$$

Optimal solution of the subproblem: $z_3^{SP} = 1$ — This generate a new column for our master problem $x = (0, 1, 0, 1, 0)$.

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

We add the new column to the master problem:

$$\text{min.} \quad \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 + \lambda_9$$

$$\text{s.t.} \quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \lambda \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Optimal solution: $z_4 = \frac{8}{3}$, $\lambda = (0, 0, 0, 0, 1, \frac{2}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

Column generation — Bin Packing Problem example

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

Knapsack subproblem associated with the solution:

$$\begin{aligned} \max. \quad & \frac{1}{3}x_1 + \frac{1}{3}x_2 + \frac{1}{3}x_3 + \frac{2}{3}x_4 + x_5 - 1 \\ \text{s.t.} \quad & x_1 + 2x_2 + 2x_3 + 3x_4 + 4x_5 \leq 6 \\ & x_i \in \{0,\ 1\} \end{aligned}$$

Optimal solution of the subproblem: $z_4^{SP} = 2$ — This generate a new column for our master problem $x = (1, 1, 0, 1, 0)$.

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

We add the new column to the master problem:

$$\text{min.} \quad \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 + \lambda_9 + \lambda_{10}$$

$$\text{s.t.} \quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \lambda \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Optimal solution: $z_6 = 2.5$, $\lambda = (0, 0, 0, 0, 1, \frac{1}{2}, \frac{1}{2}, 0, 0, \frac{1}{2})$.

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

Knapsack subproblem associated with the solution:

$$\begin{aligned} \max. \quad & \frac{1}{2}x_1 + \frac{1}{2}x_3 + \frac{1}{2}x_4 + x_5 - 1 \\ \text{s.t.} \quad & x_1 + 2x_3 + 3x_4 + 4x_5 \leq 6 \\ & x_i \in \{0,\ 1\} \end{aligned}$$

Optimal solution of the subproblem: $z_4^{SP} = 2$ — This generate a new column for our master problem $x = (1, 0, 1, 1, 0)$.

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

We add the new column to the master problem:

$$\text{min.} \quad \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 + \lambda_9 + \lambda_{10} + \lambda_{11}$$

$$\text{s.t.} \quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \lambda \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Optimal solution: $z_6 = 2.5$, $\lambda = (0, 0, 0, 0, 1, \frac{1}{2}, \frac{1}{2}, 0, 0, \frac{1}{2}, 0)$.

Column generation — Bin Packing Problem example

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

Knapsack subproblem associated with the solution:

$$\begin{aligned} \max. \quad & \frac{1}{2}x_2 + \frac{1}{2}x_3 + \frac{1}{2}x_4 + x_5 - 1 \\ \text{s.t.} \quad & 2x_2 + 2x_3 + 3x_4 + 4x_5 \leq 6 \\ & x_i \in \{0,\ 1\} \end{aligned}$$

Optimal solution of the subproblem: $z_4^{SP} = 1$ — This generate a new column for our master problem $x = (0, 1, 0, 0, 1)$.

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

We add the new column to the master problem:

$$\min. \quad \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 + \lambda_9 + \lambda_{10} + \lambda_{11} + \lambda_{12}$$

$$\text{s.t.} \quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \lambda \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Optimal solution: $z_6 = 2$, $\lambda = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1)$

Example: $W = 6$, $n = 5$, $w = (1, 2, 2, 3, 4)$.

Knapsack subproblem associated with the solution:

$$\begin{aligned} \max. \quad & x_4 + x_5 - 1 \\ \text{s.t.} \quad & 3x_4 + 4x_5 \leq 6 \\ & x_i \in \{0,\ 1\} \end{aligned}$$

Optimal solution of the subproblem: $z_4^{SP} = 0$ — This generate a new column for our master problem $x = (0, 0, 0, 1, 0)$.

The optimum of the subproblem is 0, thus the column x does not have a negative reduced-cost (in fact x is already in the master problem) $\rightarrow z_6$ is the optimal solution.

Modelling problems as integer linear programs

Linearization — A lots of elementary non-linear operations can be formulated as linear equations by adding variables and constraints to the model, but the resulting integer linear formulation may be pretty poor (weak relaxations).

Formulations — Most problems can be formulated in various way — For a given problem, most formulations have significant weaknesses and strengths, and should be solved in a suitable way.

In this section:

- How to "linearize" logical operators and simple non-linear operators?
- Examples of formulations for well-known problems (scheduling, travelling salesman, ...) — Tools and tips for modeling integer linear programs.
- Study case: **Resource-Constrained Project Scheduling Problem (RCPSP)**.

Modelling problems as integer linear programs

Modeling elementary operations

Modeling logical operators

We can model a logical proposition $P \in \{true,\ false\}$ by a binary variable $x_P \in \{0,\ 1\}$.

- Conjunction $- A = B \wedge C$:

$$\begin{cases} x_A \leq x_B \\ x_A \leq x_C \\ x_A \geq x_B + x_C - 1 \end{cases}$$

- Disjunction $- A = B \vee C$:

$$\begin{cases} x_A \geq x_B \\ x_A \geq x_C \\ x_A \leq x_B + x_C \end{cases}$$

Modeling logical operators

We can model a logical proposition $P \in \{true,\ false\}$ by a binary variable $x_P \in \{0,\ 1\}$.

- Implication $- A \Rightarrow B$:

  $x_B \geq x_A$

- Equivalence $- A \Leftrightarrow B$:

  $x_A = x_B$

- Negation $- A = \neg B$:

  $x_A + x_B = 1$

Modeling logical operators

- $f : [0, C]^2 \to \{0, 1\}, \; z = f(x, y) = (x \leq y)$

$$\begin{cases} x \leq y + C(1 - z) \\ y \leq x + Cz \\ z \in \{0, 1\} \\ 0 \leq x, y \leq C \end{cases}$$

- $f : [0, C] \to \{0, 1\}, \; z = f(x) = (x > c)$

$$\begin{cases} cz \leq x \\ x \leq Cz \\ z \in \{0, 1\} \\ 0 \leq x \leq C \end{cases}$$

Modeling non-linear functions

Some non-linear functions can be formulated automatically in a linear way by adding some integer variables.

- $f_1 \ : \ \{0,\ 1\}^2 \to \{0,\ 1\}\,,\ f_1(x,y) = xy$

$$\begin{cases} z \leq x \\ z \leq y \\ z \geq x + y - 1 \\ x, y, z \in \{0,\ 1\} \end{cases}$$

- $f_2 \ : \ \{0,\ 1\} \times [0,\ a] \to \mathbb{R},\ f_2(x,y) = xy$

$$\begin{cases} z \leq ax \\ z \leq y \\ z \geq y + y - (1-x)a \\ x \in \{0,\ 1\} \\ 0 \leq y, z \leq a \end{cases}$$

- $f_3 \ : \ [0,\ C]^2 \to [0,\ C]\,,\ f_3(x,y) = \min(x,y)$

$$\begin{cases} z \leq x \\ z \leq y \\ x \leq y + Cw \\ y \leq x + C(1-w) \\ z \geq (1-w)x + wy \\ w \in \{0,\ 1\} \\ 0 \leq x, y, z \leq C \end{cases}$$

If z has a positive coefficient in the objective function of a maximization problem (i.e. we want to maximize z), this formulation reduces to $z \leq x$ and $z \leq y$.

- $f_4 \ : \ [a,\ b] \to \mathbb{R},\ f_4(x) = |x|$ with $a < 0 < b$

$$\begin{cases} z = x^+ + x^- \\ 0 \leq x^+ \leq by \\ 0 \leq x^- \leq |a|\,(1-y) \\ a \leq x^+ - x^- \leq b \\ z \in \mathbb{R},\ y \in \{0,\ 1\} \end{cases}$$

Modelling problems as integer linear programs

Examples

Single machine scheduling — A machine needs to produce a set L of n items. Each item i has a manufacturing time p_i, a release time r_i (time at which its components are available) and a deadline d_i (time at which the time should be manufactured). The factory can only produce one item at a time, and manufacturing process cannot be interrupted (non-preemptive machine), what is the schedule that minimize the overdue deliveries?

- x_{ij} — Binary variable indicating if i is manufactured before j.
- s_i — Continuous variable indicating the starting time of task i.
- y_i — Binary variable indicating if the item i is overdue.

$$\min. \quad \sum_{j=1}^{n} y_i \tag{10a}$$

$$\text{s.t.} \quad s_j \geq s_i + p_i - M(1 - x_{ij}), \qquad \forall i, j \in L,\ i \neq j \tag{10b}$$

$$s_i + p_i \leq d_i + My_i, \qquad \forall i \in L \tag{10c}$$

$$x_{ij} + x_{ji} = 1, \qquad \forall i, j \in L,\ i \neq j \tag{10d}$$

$$x_{ij} \in \{0,\ 1\}, \qquad \forall i, j \in L \tag{10e}$$

$$y_i \in \{0,\ 1\}, \qquad \forall i \in L \tag{10f}$$

$$s_i \geq r_i, \qquad \forall i \in L \tag{10g}$$

- (10d) Two items must be ordered — i is before j or j is before i.
- (10c) $y_i = 1$ if $s_i + p_i > d_i - M$ must be big enough to inhibit the constraint when $y_i = 0$.
- (10b) If j is after i ($x_{ij} = 1$), then $s_j \geq s_i + p_i - M$ must be big enough to inhibit the constraint if j is before i ($x_{ij} = 0$).

Single machine scheduling —

We want to formulate a set of linear (in)-equalities that would enforce, for any pair of items/tasks i and j:

"If i is ordered before j, then j must start after the end of i."

$$x_{ij} = 1 \;\Rightarrow\; s_j \geq s_i + p_i$$

- We can formulate this using the following linear equation:

$$s_j \geq s_i + p_i - M(1 - x_{ij})$$

- When $x_{ij} = 1$, we get $s_j \geq s_i + p_i$ as required. When $x_{ij} = 0$, we get:

$$s_j \geq s_i + p_i - M$$

- $\rightarrow$ We want that constraint to be **inhibited** when $x_{ij} = 0$, so we need to choose a correct value for M, e.g.:

$$M = \max_i r_i + \sum_i p_i$$

Single machine scheduling —

We want to formulate a set of linear (in)-equalities that would enforce, for any items/tasks i:

"If i is processed after its deadline, i is delayed."

$$s_i + p_i > d_i \;\Leftrightarrow\; y_i = 1$$

- We can formulate one direction of the equivalence using the following linear equation:
$$s_i + p_i > d_i \;\Rightarrow\; y_i = 1 \;:\; s_i + p_i \leq d_i + M' y_i$$
- When $s_i + p_i > d_i$, we **have to** have $y_i = 1$, otherwise the constraint would be violated.
- What about the other direction?
$$\big(y_i = 1 \;\Rightarrow\; s_i + p_i > d_i\big) \Longleftrightarrow \big(s_i + p_i \leq d_i \;\Rightarrow\; y_i = 0\big)$$
- We do not need an (in)-equality this because for each i, y_i has a **positive coefficient** in the objective function of a **minimization** problem, thus the model will try to set $y_i = 0$ whenever it is possible.

Aircraft landing scheduling problem — Given a fleet P of aircraft and a set of lanes R, what is the schedule that minimizes the fees due to plane landing too early or too late?

- e_i — Earliest landing time for aircraft i.
- d_i — Latest landing time for aircraft i.
- t_i — Desired landing time for aircraft i.
- s_{ij} — Separation time between aircraft i and j.

If a plane lands before or after its desired landing time, unit fees have to be paid — g_i for landing too early, h_i for landing too late.

Aircraft landing scheduling problem — Given a fleet P of aircraft and a set of lanes R, what is the schedule that minimizes the fees due to plane landing too early or too late?

- T_i — Landing time of aircraft i.
- E_i, L_i — Advance and delay of aircraft i.
- $x_{ij} - x_{ij} = 1$ if aircraft j lands after aircraft i.
- $y_{ir} - y_{ir} = 1$ if aircraft i lands on lane r.

$$
\begin{aligned}
\text{min.} \quad & \sum_{i \in P} (h_i E_i + g_i L_i) & \\
\text{s.t.} \quad & e_i \leq T_i \leq d_i, & \forall i \in P \\
& L_i \geq T_i - t_i, & \forall i \in P \\
& E_i \geq t_i - T_i, & \forall i \in P \\
& T_j \geq T_i + s_{ij} x_{ij} - M_{ij}(1 - x_{ij}), & \forall i, j \in P \\
& x_{ij} + x_{ji} \geq y_{ir} + y_{jr} - 1, & \forall i, j \in P, \forall r \in R \\
& T_i, E_i, L_i \in \mathbb{R}^+, x_{ij} \in \{0,\ 1\}, y_{ir} \in \{0,\ 1\}, & \forall i, j \in P, \forall r \in R
\end{aligned}
$$

Aircraft landing scheduling problem — Given a fleet P of aircraft and a set of lanes R, what is the schedule that minimizes the fees due to plane landing too early or too late?

$$x_{ij} + x_{ji} \geq y_{ir} + y_{jr} - 1, \quad \forall i, j \in P, \forall r \in R$$

If both aircraft land on the same lane r, $y_{ir} = y_{jr} = 1$, they need to be scheduled:

$$\forall i, j \in P \ : \ \exists r \in R, \ y_{ir} = 1 \wedge y_{jr} = 1 \ \Rightarrow \ x_{ij} + x_{ji} = 1$$

Aircraft landing scheduling problem — Given a fleet P of aircraft and a set of lanes R, what is the schedule that minimizes the fees due to plane landing too early or too late?

$$T_j \geq T_i + s_{ij}x_{ij} - M_{ij}(1 - x_{ij})$$

If aircraft j lands after aircraft i, $x_{ij} = 1$, they must be separated by at least s_{ij}:

$$\forall i, j \in P \ : \ x_{ij} = 1 \ \Rightarrow \ T_j \geq T_i + s_{ij}$$

To inhibit the constraint when $x_{ij} = 0$, we need the following to be always verified:

$$T_j \geq T_i - M_{ij}M_{ij} \geq T_i - T_j$$

Since we know that for any plane i, $e_i \leq T_i \leq d_i$, we can choose:

$$M_{ij} = d_i - e_j \geq T_i - T_j$$

Modeling ILP — Extended and compact formulations

Most problems can be formulated as integer linear programs in multiple ways — Different formulations lead to different resolution techniques.

- Pseudo-polynomial or extended formulations:
  - Better *LP* relaxations, early node pruning in branch & bounds.
  - Large *MILP* — Possibly pseudo-polynomial or even exponential number of binary variables and/or constraints $\Rightarrow$ Needs of specific methods (column generation, branch & cuts, ...).
- Polynomial-size or compact formulations:
  - Fast node evaluation due to the small size of the relaxation.
  - Weaker relaxations, more nodes need to be explored, cuts need to be generated.

Travelling salesman problem (TSP) — Given a list of n cities and the distances c_{ij} between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

Miller-Tucker-Zemlin (MTZ) formulation —

- $x_{ij} \in \{0, 1\}$ — Binary variable indicating if we go directly from city i to city j.
- $u_i \in \{1, \ldots, n\}$ — Subtour elimination variables — u_i is the position of city i in the tour.

$$\min. \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{11a}$$

$$\text{s.t.} \quad \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1, \qquad \forall i \in \{1, \ldots, n\} \tag{11b}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1, \qquad \forall j \in \{1, \ldots, n\} \tag{11c}$$

$$u_1 = 1 \tag{11d}$$

$$2 \leq u_i \leq n, \qquad \forall i \in \{1, \ldots, n\} \tag{11e}$$

$$u_i - u_j + 1 \leq (n-1)(1 - x_{ij}), \qquad \forall i, j \in \{2, \ldots, n\} \tag{11f}$$

$$x_{ij} \in \{0, 1\}, \; u_i \in \mathbb{N} \qquad \forall i, j \in \{1, \ldots, n\} \tag{11g}$$

Flow formulation —

- $x_{ij} \in \{0,\ 1\}$ — Binary variable indicating if we go directly from city i to city j.
- $y_{ij} \in \mathbb{R}_*^+$ — Subtour elimination variables — y_{ij} is the *flow* on arc (i, j), each city produces one unit of flow.

$$\text{min.} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{12a}$$

$$\text{s.t.} \quad \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1, \qquad \forall i \in \{1,\ \ldots,\ n\} \tag{12b}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1, \qquad \forall j \in \{1,\ \ldots,\ n\} \tag{12c}$$

$$\sum_{j=2}^{n} y_{1j} = 1 \tag{12d}$$

$$\sum_{j=1}^{n} y_{ij} = \sum_{j=1}^{n} y_{ji} + 1, \qquad \forall i \in \{2,\ \ldots,\ n\} \tag{12e}$$

$$y_{ij} \leq n x_{ij}, \qquad \forall i, j \in \{1,\ \ldots,\ n\} \tag{12f}$$

$$x_{ij} \in \{0,\ 1\}\ ,\ y_{ij} \in \mathbb{R}_*^+ \qquad \forall i, j \in \{1,\ \ldots,\ n\} \tag{12g}$$

Subtour formulation —

- $x_{ij} \in \{0,\ 1\}$ — Binary variable indicating if we go directly from city i to city j.

$$\min. \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{13a}$$

$$\text{s.t.} \quad \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1, \qquad \forall i \in \{1,\ \ldots,\ n\} \tag{13b}$$

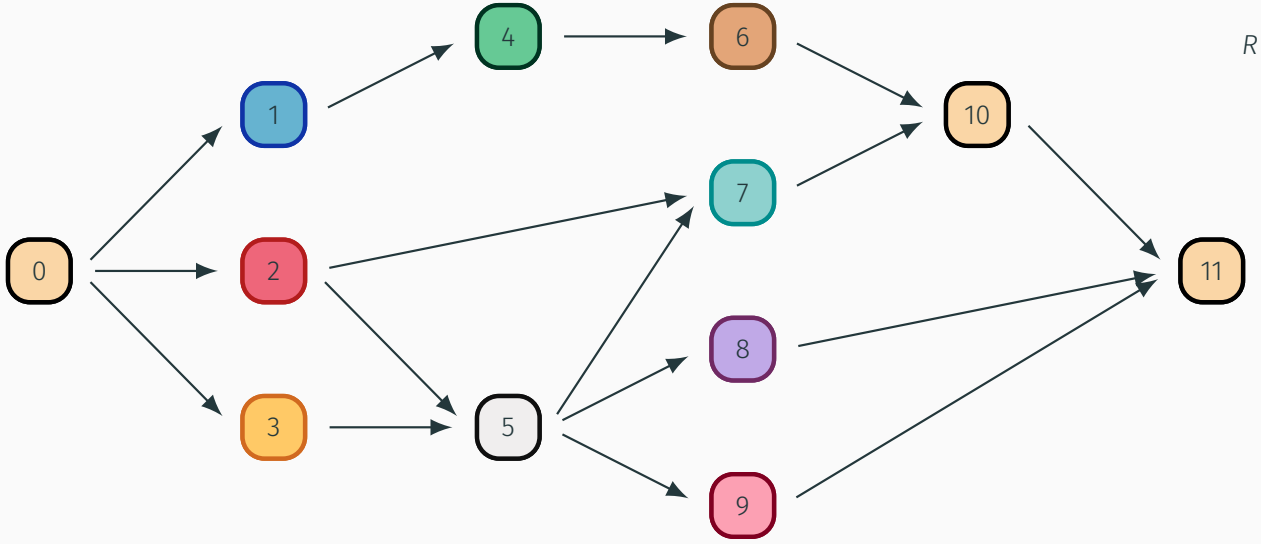
$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1, \qquad \forall j \in \{1,\ \ldots,\ n\} \tag{13c}$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \qquad S \subset \{1,\ \ldots,\ n\}\,,\ S \neq \emptyset \tag{13d}$$

$$x_{ij} \in \{0,\ 1\} \qquad \forall i, j \in \{1,\ \ldots,\ n\} \tag{13e}$$

Resource-Constrained Project Scheduling Problem (RCPSP) —

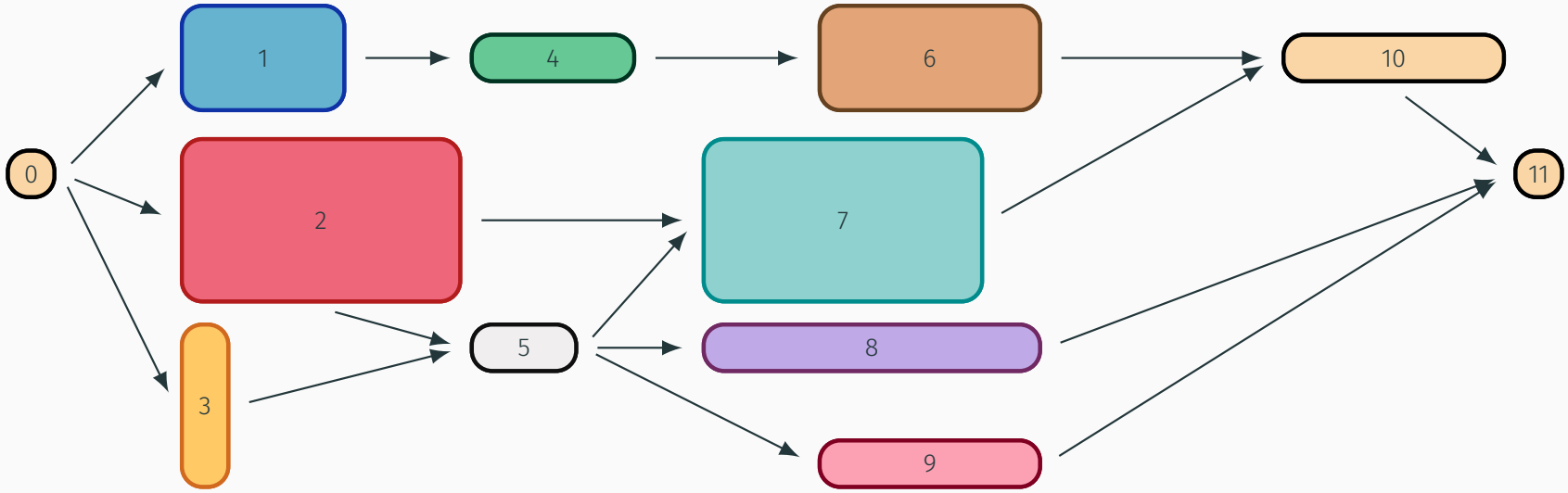
- R — Set of resources with a **limited constant** availability $B_k \geq 0$.
- A — Set of activities with duration $p_i \geq 0$ and resource requirements $b_{ik} \geq 0$ for each resource $k \in R$.
- E — Set of precedence constraints $(i, j),\ i, j \in A,\ i < j$, this can be represented as a *directed acyclic graph*.
- $\mathcal{T}$ — Time interval (scheduling horizon).



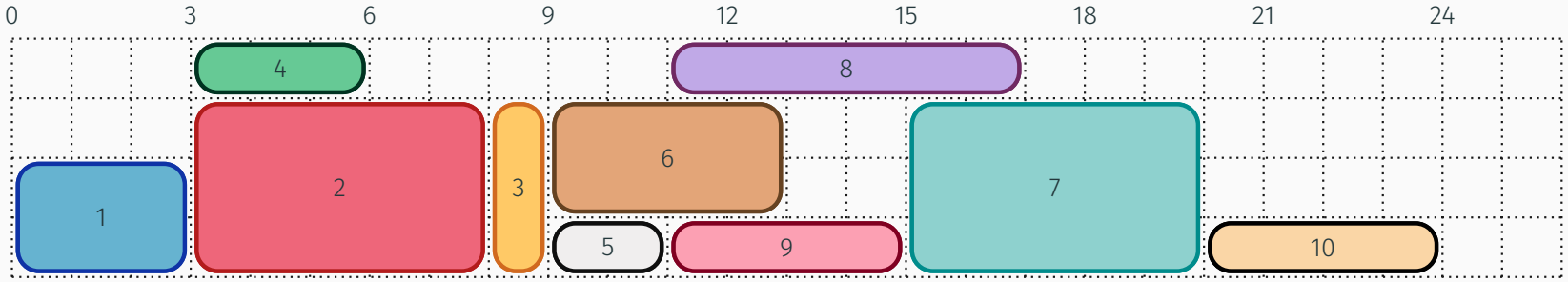
$R = \{1\}, B_1 = 4, \mathcal{T} = [0, 30)$.

i	p_i	b_1
1	3	2
2	5	3
3	1	3
4	3	1
5	2	1
6	4	2
7	5	3
8	6	1
9	4	1
10	4	1

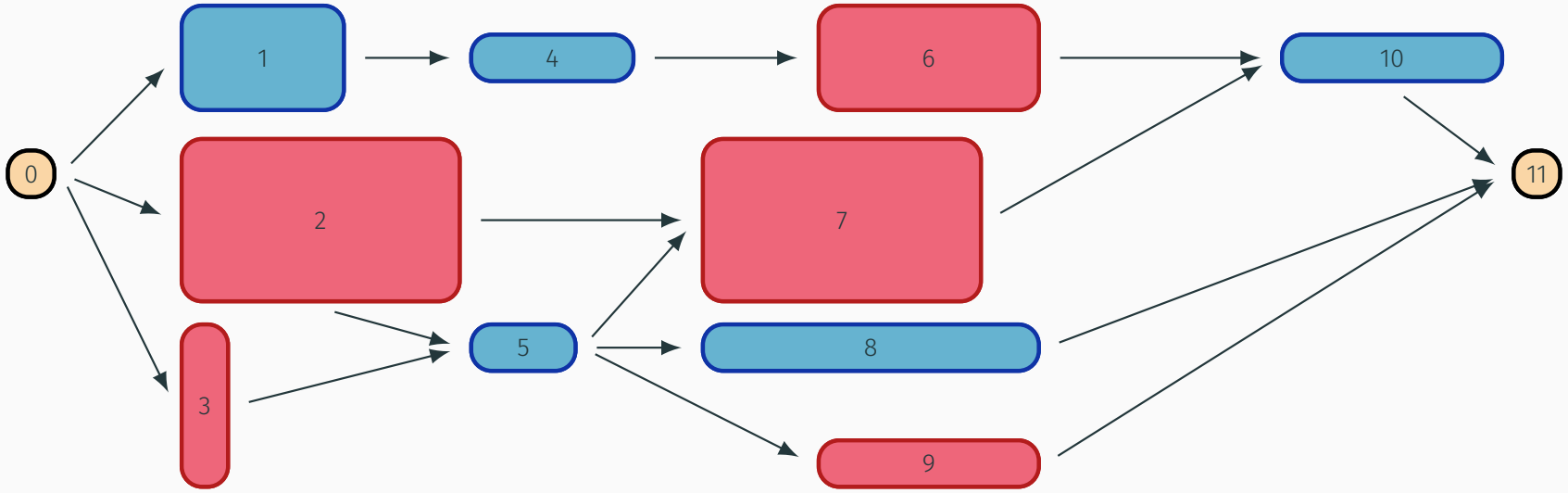
Example with $A = \{1, \ldots, 10\}$ and a single resource B of capacity $B_1 = 4$:



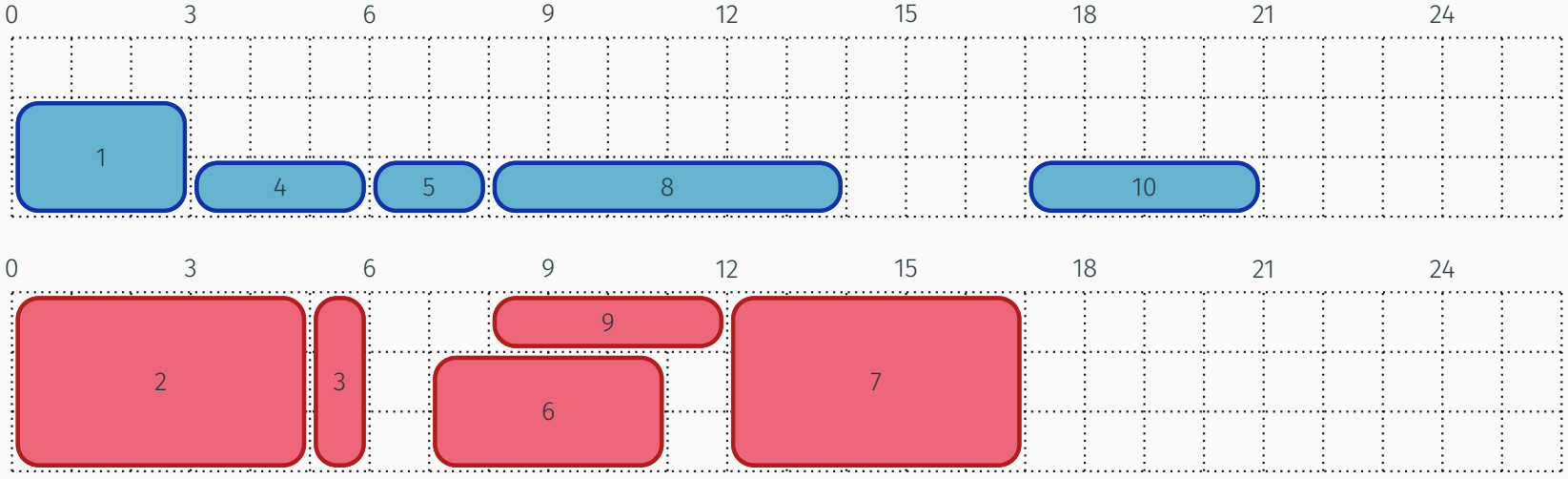
Feasible solution:



Example with $A = \{1, \ldots, 10\}$ and two resources of capacity $B_1 = B_2 = 3$:



Feasible solution:



RCPSP —

It is often useful to start with a conceptual (i.e. non-linear) mathematical formulation.

- $S_i,\ i \in A$ — Start time of task i.
- $C_{\max}$ — Makespan or total project duration.

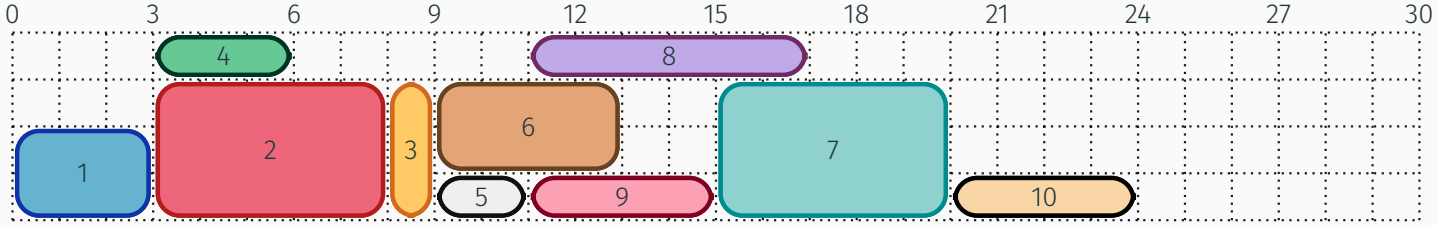
$$\begin{aligned}
\text{min.} \quad & C_{\max} = \max_{i \in A} S_i + p_i && && \text{(14a)}\\
\text{s.t.} \quad & S_j \geq S_i + p_i, && \forall (i,j) \in E && \text{(14b)}\\
& \sum_{i \in A(t)} b_{ik} \geq B_k, && \forall t \in \mathcal{T},\ \forall k \in R && \text{(14c)}\\
& S_j \geq 0, && \forall i \in A && \text{(14d)}
\end{aligned}$$

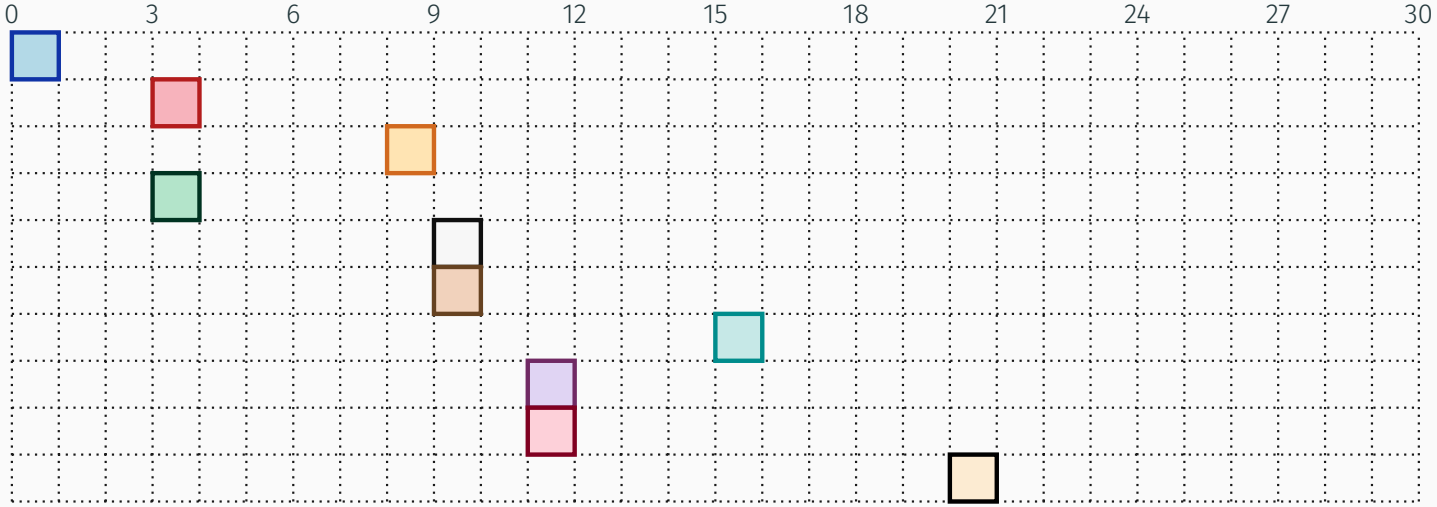
- (14b) — Precedence constraints.
- (14c) — Resource constraints, $A(t) = \left\{ j \in A \ : \ t \in \left[S_j,\ S_j + p_j \right) \right\}$.

RCPSP — Aggregated time-indexed formulation [Pritsker, Waiters, et al. 1969]

For integer data (processing time), S_i is integral.

- $x_{it} \in \{0,\ 1\} - x_{ij} = 1 \Leftrightarrow\ S_i = t,\ t \in T = \mathcal{T} \cap \mathbb{N}$, pseudo-polynomial number of variables ($|A|\ |T|$).





RCPSP — Aggregated time-indexed formulation (DT) [Pritsker, Waiters, et al. 1969]

- $x_{it} \in \{0,\ 1\} - x_{ij} = 1 \Leftrightarrow\ S_i = t,\ t \in T = \mathcal{T} \cap \mathbb{N}$, pseudo-polynomial number of variables ($|A|\ |T|$).

Link with the conceptual formulation:

- $S_i = \sum_{t \in T} t x_{it}$
- $A(t) = \{i \in A\ :\ \exists \tau \in \{t - p_i + 1,\ \ldots,\ t\}\ ,\ x_{i\tau} = 1\}$

$$
\begin{aligned}
\text{min.} \quad & \sum_{t \in T} t x_{n+1,t} & \\
\text{s.t.} \quad & \sum_{t \in T} t x_{jt} - \sum_{t \in T} t x_{it} \geq p_i, & \forall (i,j) \in E \\
& \sum_{i \in A} \sum_{\tau = t - p_i + 1}^{t} b_{ik} x_{i\tau} \leq B_k, & \forall t \in T,\ \forall k \in R \\
& \sum_{t \in T} x_{it} = 1, & \forall i \in A \\
& x_{it} \in \{0,\ 1\} & \forall i \in A,\ \forall t \in T
\end{aligned}
$$

RCPSP — Disaggregated time-indexed formulation (DDT) [Christofides et al. 1987]

The relaxation of the aggregated time-indexed formulation is pretty weak — The lower bound on the makespan is often not better than a trivial lower bound.

The model can be reinforced by disaggregation of the precedence constraints, i.e. replacing precedence constraints by:

$$\sum_{\tau=0}^{t-p_i} x_{i\tau} - \sum_{\tau=0}^{t} x_{j\tau} \geq 0, \quad \forall (i,j) \in E,\ \forall t \in T$$

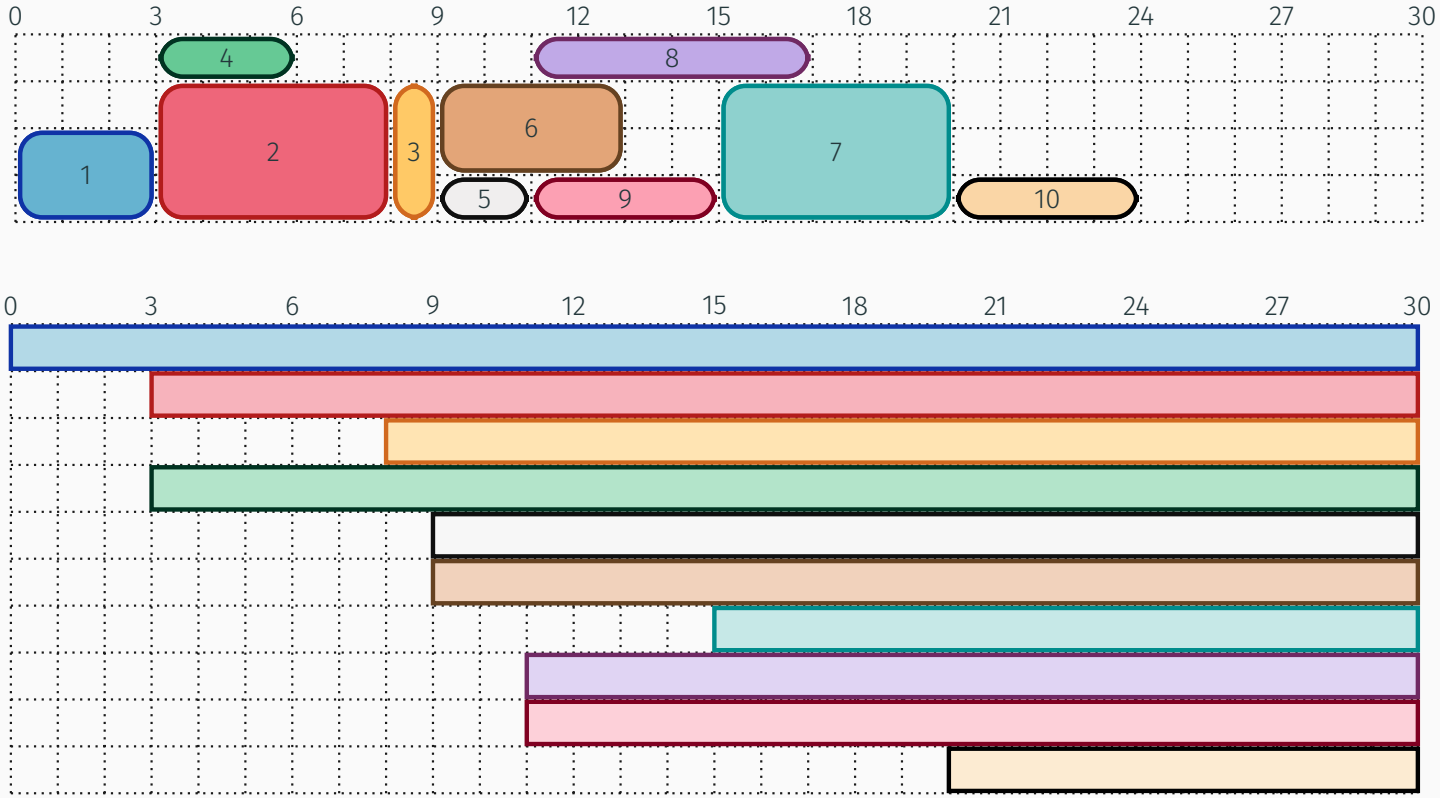
This model the set of constraints:

$$\forall (i,j) \in E,\ \forall t \in T\ :\ S_j \leq t \ \Rightarrow\ S_i \leq t - p_i$$

- The constraint matrix without resource constraints is totally unimodular.
- Total unimodularity is preserved by lagrangean relaxation of the resource constraints — Efficiently computable by a max-flow algorithm [Möhring et al. 2003].

RCPSP — Time-indexed formulation with step variables (SDDT) [Pritsker and Watters 1968]

- $\xi_{it} \in \{0, 1\} - x_{ij} = 1 \Leftrightarrow S_i \leq t, \ t \in T = \mathcal{T} \cap \mathbb{N}$, pseudo-polynomial number of variables ($|A|\ |T|$).



RCPSP — Time-indexed formulation with step variables (SDDT) [Pritsker and Watters 1968]

The time-indexed formulation with step variables (SDDT) can be obtained by transforming the desaggregated time-indexed formulation with pulse variables (DDT) using the following:

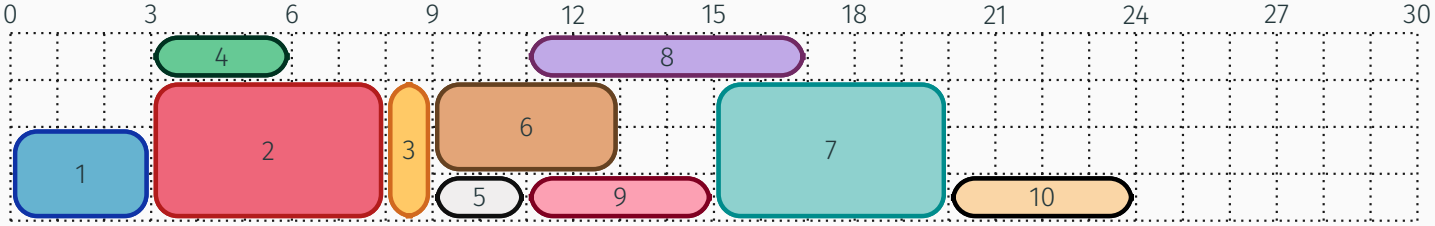
$$\xi_{it} = \sum_{\tau=0}^{t} x_{it}, \quad \forall i \in A, \ \forall t \in T$$

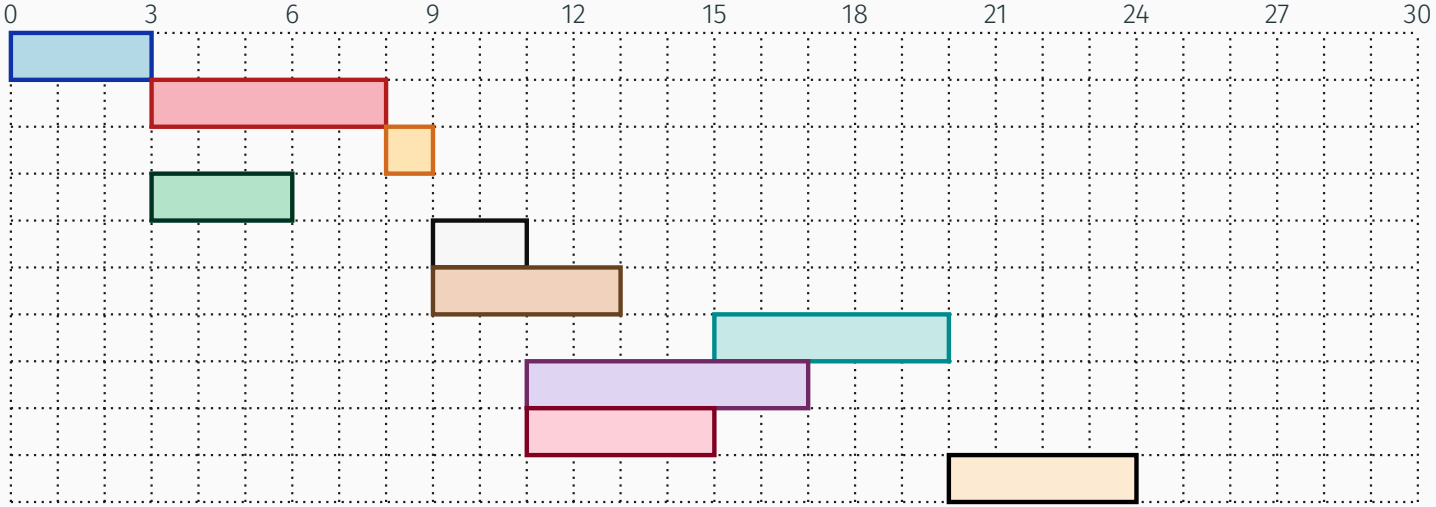
- This is a non-singular transformation (NST) — Formulations that can be obtained by a NST are strictly equivalent. They have the same set of feasible solutions and the same relaxation value.
- There is another formulation using $\xi' = 1 \Leftrightarrow S_i + p_i \leq t$ which is equivalent (by NST transformation) to (SDDT) [Bianco et al. 2013].

RCPSP — On/Off time-indexed formulation (OODDT) [Pritsker, Waiters, et al. 1969]

- $\mu_{it} \in \{0,\ 1\} - \mu_{ij} = 1 \Leftrightarrow\ t \in [S_i,\ S_i + p_i[$, **pseudo-polynomial** number of variables ($|A|\ |T|$).

It is possible to create a (OODDT) formulation equivalent to (DDT) using a non-singular transformation [Artigues 2013].





RCPSP — Extended time-indexed formulations [Hardin et al. 2008; Christofides et al. 1987; Mingozzi et al. 1998].

- *Minimal Forbidden Set (MFS)*, F — Minimal set of activities that cannot be scheduled in parallel due to resource constraints.

$$\sum_{i \in F} b_{ik} > B_k \quad \text{and} \quad \forall j \in F, \sum_{i \in F \setminus \{j\}} \leq B_k$$

  We can create a **forbidden set-based** formulation by adding cover-like cuts for each MFS — This formulation would have an exponential number of constraints.

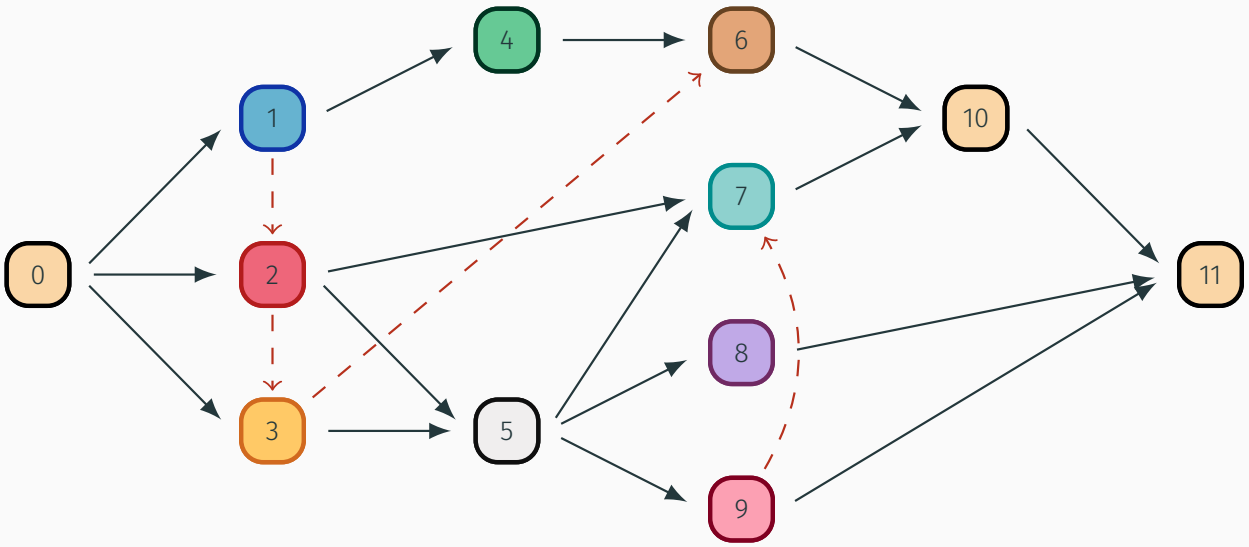
- *Feasible Set (FS)*— Set of activities that can be scheduled in parallel. A **feasible subset-based** formulation can be obtained from (DDT) by replacing the resource constraints by an exponential number of feasible set constraints.

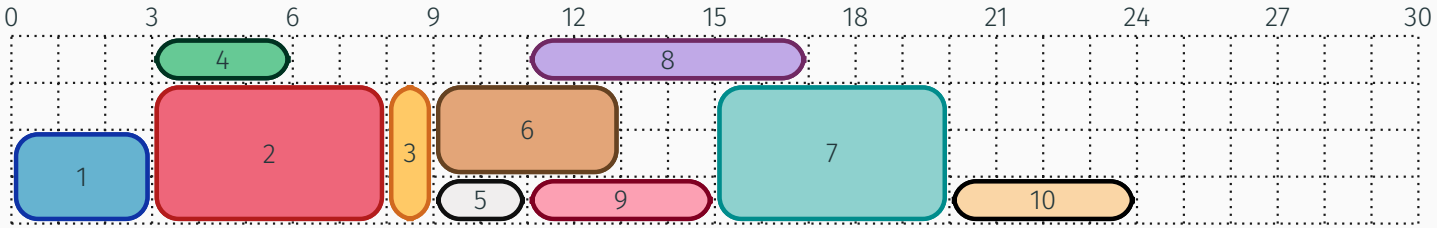
Note: This is a non-exhaustive list of possible extended time-indexed formulations.

RCPSP — Sequencing (strict ordering) formulations.

Adding precedence constraints such that all resource conflicts are resolved —

$$z_{ij} = 1 \Leftrightarrow S_j \geq S_i + p_i, \quad \forall i, j \in A$$





RCPSP — Sequence formulation based on minimal forbidden sets [Olaguibel et al. 1993].

- S_i — Start time of task i.
- $z_{ij} \in \{0,\ 1\}$ — $z_{ij} = 1$ if task i is before j in the schedule.

$$
\begin{aligned}
\text{min.} \quad & S_{n+1} & \\
\text{s.t.} \quad & z_{ij} + z_{ji} \leq 1, & \forall i, j \in A,\ i < j \\
& z_{ij} + z_{jh} - z_{ih} \leq 1, & \forall i, j, j \in A,\ i \neq j \neq h \\
& z_{ij} = 1, & \forall (i, j) \in E \\
& S_j + (1 - z_{ij}) M_{ij} \geq S_i + p_i, & \forall i, j \in A,\ i \neq j \\
& \sum_{\substack{i,j \in F \\ i \neq j}} z_{ij} \geq 1, & \forall F \in \mathcal{F} \\
& z_{ij} \in \{0,\ 1\} & \forall i, j \in A
\end{aligned}
$$

We need an exponential number of forbidden sets constraints — This formulation is extended.

RCPSP — Sequence formulation based on resource flow [Artigues et al. 2003].

- S_i — Start time of task i.
- $z_{ij} \in \{0,\ 1\}$ — $z_{ij} = 1$ if task i is before j in the schedule.
- Φ_{ij}^k — Units of resource k transferred from i to j.

We can replace the forbidden set constraints in the previous formulation by:

$$\Phi_{ij}^k \leq \min(b_{ik},\ b_{jk}) z_{ij}, \qquad \forall i, j \in A,\ \forall k \in R,\ i \neq j$$

$$\sum_{j \in V \setminus \{i\}} \Phi_{ij}^k = b_{ik}, \qquad \forall i \in A,\ j \neq n+1$$

$$\sum_{i \in V \setminus \{j\}} \Phi_{ij}^k = b_{jk}, \qquad \forall j \in A,\ j \neq 0$$

$$0 \leq \Phi_{ij}^k \leq min(b_{ik}, b_{jk}), \qquad \forall i, j \in A,\ \forall k \in R,\ i \neq n+1,\ j \neq 0$$

We need $\mathcal{O}(|A|^2\ |R|)$ additional continuous variables and constraints — This formulation is compact.

RCPSP — Event-based formulations [Lasserre et al. 1992; Dauzère-Pérès et al. 1995; Pinto et al. 1995].

Start and End event variables —

- ε — Set of remarkable events.
- $t_e \geq 0$ — Event date representing the start and end of at least one activity.
- $a_{ie}^{-} \in \{0,\ 1\}$ — Start assignment variables, $a_{ie}^{-} = 1 \Leftrightarrow S_i = t_e$.
- $a_{ie}^{+} \in \{0,\ 1\}$ — End assignment variables, $a_{ie}^{+} = 1 \Leftrightarrow S_i + p_i = t_e$.

We need $2(n+1)\,|\varepsilon|$ binary variables.

RCPSP — Event-based formulations.

On/Off event variables (OOE) [Koné et al. 2011]. —

- ε — Set of remarkable events.
- $t_e \geq 0$ — Event date representing the start of at least one activity.
- $a_{ie} \in \{0,\ 1\}$ — On/off binary variables, $a_{ie} = 1 \Leftrightarrow [S_i,\ S_i + p_i] \cap [t_e,\ t_e + 1] \neq \emptyset$.

We need $n\,|\varepsilon|$ binary variables.

RCPSP — Event-based formulations — On/Off event variables (OOE) [Koné et al. 2011].

$$
\begin{aligned}
\text{min.} \quad & C_{max} \\
\text{s.t.} \quad & C_{max} \geq t_e + (a_{ie} - a_{i,e-1})p_i && \forall e \in \varepsilon,\ \forall i \in A \\
& t_{e+1} \geq t_e, && \forall e \in \varepsilon,\ e \neq n \\
& t_0 = 0 \\
& t_f \geq t_e + (a_{ie} - a_{i,e-1} - a_{if} + a_{i,f-1} - 1)p_i, && \forall e, f \in \varepsilon,\ \forall i \in A,\ f > e \neq 0 \\
& \sum_{f=0}^{e-1} a_{if} \geq e(1 - a_{ie} + a_{i,e-1}) && \forall e \in \varepsilon,\ \forall i \in A \\
& \sum_{f=e}^{n-1} a_{if} \geq e(1 + a_{ie} - a_{i,e-1}) && \forall e \in \varepsilon,\ \forall i \in A \\
& \sum_{e \in \varepsilon} a_{ie} \geq 1, && \forall i \in A \\
& a_{ie} + \sum_{f=0}^{e} a_{je} \leq 1 + (1 - a_{ie})e, && \forall e \in \varepsilon,\ \forall (i,j) \in E \\
& \sum_{i=0}^{n-1} r_{ik} a_{ie} \leq B_k, && \forall e \in \varepsilon,\ \forall k \in R \\
& t_e \geq 0,\ a_{ie} \in \{0,\ 1\}, && \forall e \in \varepsilon,\ \forall i \in A
\end{aligned}
$$

References I

Artigues, Christian (2013). **A note on time-indexed formulations for the resource-constrained project scheduling problem**. LAAS report 13206. Toulouse, France: Laboratoire d'Analyse et d'Architectures des Systèmes, LAAS-CNRS.

Artigues, Christian, Philippe Michelon, and Stéphane Reusser (2003). "Insertion techniques for static and dynamic resource-constrained project scheduling". In: **European Journal of Operational Research** 149.2, pp. 249–267.

Bianco, Lucio and Massimiliano Caramia (2013). "A new formulation for the project scheduling problem under limited resources". In: **Flexible Services and Manufacturing Journal** 25.1-2, pp. 6–24.

Christofides, Nicos, Ramon Alvarez-Valdés, and José M Tamarit (1987). "Project scheduling with resource constraints: A branch and bound approach". In: **European Journal of Operational Research** 29.3, pp. 262–273.

Dauzère-Pérès, S and JB Lasserre (1995). "A new mixed-integer formulation of the flow-shop sequencing problem". In: **2nd Workshop on models and algorithms for planning and scheduling problems, Wernigerode, Allemagne**.

Hardin, Jill R, George L Nemhauser, and Martin WP Savelsbergh (2008). "Strong valid inequalities for the resource-constrained scheduling problem with uniform resource requirements". In: **Discrete Optimization** 5.1, pp. 19–35.

References II

- Koné, Oumar et al. (2011). "Event-based MILP models for resource-constrained project scheduling problems". In: Computers & Operations Research 38.1, pp. 3–13.
- Lasserre, Jean B and Maurice Queyranne (1992). "Generic Scheduling Polyhedra and a New Mixed-Integer Formulation for Single-Machine Scheduling.". In: IPCO, pp. 136–149.
- Mingozzi, Aristide et al. (1998). "An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation". In: Management science 44.5, pp. 714–729.
- Möhring, Rolf H et al. (2003). "Solving project scheduling problems by minimum cut computations". In: Management Science 49.3, pp. 330–350.
- Olaguibel, Ramon Alvarez-Valdes and JoseManuel Tamarit Goerlich (1993). "The project scheduling polyhedron: dimension, facets and lifting theorems". In: European Journal of Operational Research 67.2, pp. 204–220.
- Pinto, Jose M and Ignacio E Grossmann (1995). "A continuous time mixed integer linear programming model for short term scheduling of multistage batch plants". In: Industrial & Engineering Chemistry Research 34.9, pp. 3037–3051.

References III

Pritsker, A Alan B, Lawrence J Waiters, and Philip M Wolfe (1969). "Multiproject scheduling with limited resources: A zero-one programming approach". In: Management science 16.1, pp. 93–108.

Pritsker, A Alan B and Lawrence J Watters (1968). A zero-one programming approach to scheduling with limited resources. Tech. rep. RAND CORP SANTA MONICA CALIF.