Introduction to Optimization

Vincent Leclère (CERMICS, ENPC)

Autumn 2026

Why should I bother to learn this stuff?

- Many engineering decisions are inherently discrete: build or not, switch on or off, assign one resource to another, choose a route, schedule a task.
- MILP is used for problems such as routing, scheduling, facility location, unit commitment, network design and production planning.
- These problems are computationally difficult in theory, but modern MILP solvers can handle surprisingly large practical instances.
- Solver performance depends strongly on how you formulate the problem: mathematically equivalent models may differ enormously in solution time.

Contents

Contents

Mixed Integer Linear Programming (MILP)

A Mixed Integer Linear Program is an optimization problem of the form

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x_i \in \mathbb{Z} \qquad \forall i \in I \subseteq [n] \\ & x_j \in \mathbb{R} \qquad \forall j \in [n] \setminus I \end{aligned}$$

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

We are going to discuss more in detail the special case of Binary Integer Programs (BILP) where $I = [n]$ and $x_i \in \{0, 1\}$ for all $i \in [n]$.

Even BILPs are NP-hard in general: integrality fundamentally changes the computational difficulty.

Mixed Integer Linear Programming (MILP)

A Mixed Integer Linear Program is an optimization problem of the form

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x_i \in \mathbb{Z} \qquad \forall i \in I \subseteq [n] \\ & x_j \in \mathbb{R} \qquad \forall j \in [n] \setminus I \end{aligned}$$

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

We are going to discuss more in detail the special case of Binary Integer Programs (BILP) where $I = [n]$ and $x_i \in \{0, 1\}$ for all $i \in [n]$.

Even BILPs are NP-hard in general: integrality fundamentally changes the computational difficulty.

Mixed Integer Linear Programming (MILP)

A Mixed Integer Linear Program is an optimization problem of the form

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x_i \in \mathbb{Z} && \forall i \in I \subseteq [n] \\ & x_j \in \mathbb{R} && \forall j \in [n] \setminus I \end{aligned}$$

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

We are going to discuss more in detail the special case of Binary Integer Programs (BILP) where $I = [n]$ and $x_i \in \{0, 1\}$ for all $i \in [n]$.

Even BILPs are NP-hard in general: integrality fundamentally changes the computational difficulty.

Why study MILP?

- MILP is a very general modeling framework that can capture a wide range of combinatorial optimization problems.
- Many real-world problems in logistics, finance, manufacturing, and other fields can be formulated as MILP.

- Advances in MILP algorithms and solvers (in particular professional ones like Gurobi) have made it possible to solve large-scale instances efficiently.
- However, solvers are not magic: understanding the structure of MILP problems is crucial for effective modeling and solution.
- In particular there are multiple ways of formulating the same problem as a MILP, with very different computational performances.
- In addition there are widely used cases where we need to design specific algorithms (*e.g.*column generation) to solve MILP problems efficiently.

Why study MILP?

- MILP is a very general modeling framework that can capture a wide range of combinatorial optimization problems.
- Many real-world problems in logistics, finance, manufacturing, and other fields can be formulated as MILP.

- Advances in MILP algorithms and solvers (in particular professional ones like Gurobi) have made it possible to solve large-scale instances efficiently.
- However, solvers are not magic: understanding the structure of MILP problems is crucial for effective modeling and solution.
- In particular there are multiple ways of formulating the same problem as a MILP, with very different computational performances.
- In addition there are widely used cases where we need to design specific algorithms (*e.g.*column generation) to solve MILP problems efficiently.

Why study MILP?

- MILP is a very general modeling framework that can capture a wide range of combinatorial optimization problems.
- Many real-world problems in logistics, finance, manufacturing, and other fields can be formulated as MILP.

- Advances in MILP algorithms and solvers (in particular professional ones like Gurobi) have made it possible to solve large-scale instances efficiently.
- However, solvers are not magic: understanding the structure of MILP problems is crucial for effective modeling and solution.
- In particular there are multiple ways of formulating the same problem as a MILP, with very different computational performances.
- In addition there are widely used cases where we need to design specific algorithms (*e.g.*column generation) to solve MILP problems efficiently.

Why study MILP?

- MILP is a very general modeling framework that can capture a wide range of combinatorial optimization problems.
- Many real-world problems in logistics, finance, manufacturing, and other fields can be formulated as MILP.

- Advances in MILP algorithms and solvers (in particular professional ones like Gurobi) have made it possible to solve large-scale instances efficiently.
- However, solvers are not magic: understanding the structure of MILP problems is crucial for effective modeling and solution.
- In particular there are multiple ways of formulating the same problem as a MILP, with very different computational performances.
- In addition there are widely used cases where we need to design specific algorithms (*e.g.*column generation) to solve MILP problems efficiently.

Contents

Examples of MILP

- **Knapsack problem** (♡): given n items with weights w_i and values v_i, and a knapsack of capacity W, select items to maximize value without exceeding capacity.
- **Set Covering problem:** given a universe of elements and a collection of subsets, select the minimum number of subsets to cover all elements.
- **Facility Location problem:** decide where to open facilities to minimize costs while serving clients.
- **Scheduling problem:** assign tasks to time slots or resources while minimizing total completion time or maximizing resource utilization.
- **Traveling Salesman Problem (TSP):** find the shortest possible route that visits each city exactly once and returns to the origin city.
- **Vehicle Routing Problem (VRP):** determine optimal routes for a fleet of vehicles to deliver goods to a set of customers.

Examples of MILP (cont.)

- **Unit Commitment Problem:** decide which power generation units to turn on or off to meet electricity demand at minimum cost.
- **Network Design:** optimize the design of a network (e.g., telecommunications, transportation) to minimize costs while satisfying connectivity requirements.
- **Staffing Problem:** determine the optimal number of staff to hire and schedule to meet demand while minimizing costs.
- **Capital Budgeting:** select a portfolio of investment projects to maximize return while adhering to budget constraints.
- **Cutting Stock Problem:** minimize waste when cutting raw materials into smaller pieces to meet specific size requirements.
- **Crew Scheduling:** assign crew members to shifts while satisfying labor regulations and minimizing costs.
- **Disaster Relief Logistics:** optimize the distribution of relief supplies to affected areas while minimizing costs and response time.
- **Mine Planning:** determine the optimal extraction schedule of minerals to maximize profit while adhering to environmental and operational constraints.

Examples of MILP (cont.)

- **Production Lot-Sizing with Setups:** decide when and how much to produce across periods with inventory balance and setup costs.
- **Sports League Scheduling:** assign matches to dates/venues while enforcing fairness, rest, and venue constraints.
- **Facility Layout:** place departments/machines to minimize material handling cost subject to non-overlap and adjacency rules.
- **Blending/Recipe Design:** select ingredients and proportions to meet quality specs and demand at minimum cost (with ingredient on/off decisions).
- **Inventory Routing:** coordinate delivery schedules and quantities with vehicle routes to satisfy time-phased inventory constraints.
- **Transmission Expansion Planning:** choose lines to build/upgrade in a power grid to meet demand and reliability at minimum cost.
- **Maximum Independent Set (Conflict Selection):** select a subset of mutually compatible options to maximize total weight/value.
- ...

Contents

Contents

Knapsack problem example

- We have a list of n items, each with a weight w_i and a value v_i.
- We have a knapsack with a maximum weight limit W.
- Select items to maximize the summed value without exceeding the weight limit.

➥ obvious if you can cut the objects, NP-hard otherwise.

- We can formulate this as a Binary Integer Program:

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

Knapsack problem example

- We have a list of n items, each with a weight w_i and a value v_i.
- We have a knapsack with a maximum weight limit W.
- Select items to maximize the summed value without exceeding the weight limit.

➥ obvious if you can cut the objects, NP-hard otherwise.

- We can formulate this as a Binary Integer Program:

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

Branch and Bound for Binary Integer Programs

$$
\begin{aligned}
\min \quad & c^\top x \\
\text{s.t.} \quad & Ax \leq b \\
& x_i \in \{0,1\} \qquad \forall i \in [n]
\end{aligned}
$$

- There are a finite number of feasible solutions.
- The naive approach is to enumerate all feasible solutions and select the best one.
- However, there are 2^n binary assignments, so enumerating all is not feasible for large n.

➥ Branch and Bound is a systematic method to explore the solution space, without actually enumerating all solutions.

➥ The key idea is to compute bounds on the optimal solution in different regions of the solution space, and use these bounds to prune regions that cannot contain the optimal solution.

Branch and Bound for Binary Integer Programs

$$
\begin{aligned}
\min \quad & c^\top x \\
\text{s.t.} \quad & Ax \leq b \\
& x_i \in \{0,1\} \qquad\qquad \forall i \in [n]
\end{aligned}
$$

- There are a finite number of feasible solutions.
- The naive approach is to enumerate all feasible solutions and select the best one.

- However, there are 2^n binary assignments, so enumerating all is not feasible for large n.

➥ Branch and Bound is a systematic method to explore the solution space, without actually enumerating all solutions.

➥ The key idea is to compute bounds on the optimal solution in different regions of the solution space, and use these bounds to prune regions that cannot contain the optimal solution.

Branch and Bound for Binary Integer Programs

$$
\begin{aligned}
\min \quad & c^\top x \\
\text{s.t.} \quad & Ax \leq b \\
& x_i \in \{0,1\} \qquad\qquad \forall i \in [n]
\end{aligned}
$$

- There are a finite number of feasible solutions.
- The naive approach is to enumerate all feasible solutions and select the best one.

- However, there are 2^n binary assignments, so enumerating all is not feasible for large n.

➥ Branch and Bound is a systematic method to explore the solution space, without actually enumerating all solutions.

➥ The key idea is to compute bounds on the optimal solution in different regions of the solution space, and use these bounds to prune regions that cannot contain the optimal solution.

Binary tree representation of Binary Integer Programs

- We can represent the solution space of a Binary Integer Program as a binary tree.
- Each node in the tree represents a subproblem with additional constraints on the variables.
- The root node represents the original problem (no additional constraints).

- At each node, we can branch on a variable x_i that is not yet fixed to 0 or 1.
- This creates two child nodes: one with the constraint $x_i = 0$ and another with $x_i = 1$.

- This branching continues until we reach leaf nodes where all variables are fixed.
- Each leaf node (at most 2^n) represents a complete binary assignment, which may be feasible or infeasible.

Binary tree representation of Binary Integer Programs

- We can represent the solution space of a Binary Integer Program as a binary tree.
- Each node in the tree represents a subproblem with additional constraints on the variables.
- The root node represents the original problem (no additional constraints).

- At each node, we can branch on a variable x_i that is not yet fixed to 0 or 1.
- This creates two child nodes: one with the constraint $x_i = 0$ and another with $x_i = 1$.

- This branching continues until we reach leaf nodes where all variables are fixed.
- Each leaf node (at most 2^n) represents a complete binary assignment, which may be feasible or infeasible.

Binary tree representation of Binary Integer Programs

- We can represent the solution space of a Binary Integer Program as a binary tree.
- Each node in the tree represents a subproblem with additional constraints on the variables.
- The root node represents the original problem (no additional constraints).

- At each node, we can branch on a variable x_i that is not yet fixed to 0 or 1.
- This creates two child nodes: one with the constraint $x_i = 0$ and another with $x_i = 1$.

- This branching continues until we reach leaf nodes where all variables are fixed.
- Each leaf node (at most 2^n) represents a complete binary assignment, which may be feasible or infeasible.

Lower bounding with Linear Programming relaxation

At a certain node ν of the tree, we have a subproblem of the form

$$
\begin{aligned}
(\mathcal{P}_\nu) \quad & \min_{x \in \{0,1\}^n} && c^\top x \\
& \text{s.t.} && Ax \leq b \\
& && x_i = 0 && \forall i \in I_\nu^0 \\
& && x_i = 1 && \forall i \in I_\nu^1
\end{aligned}
$$

where $I_\nu = I_\nu^0 \cup I_\nu^1$ is the set of indices of variables fixed to 0 or 1.

A lower bound on the optimal value of $(\mathcal{P}_\nu)$ is the value of its linear programming relaxation:

$$
\begin{aligned}
(\mathcal{P}_\nu^{LP}) \quad & \min_{x \in [0,1]^n} && c^\top x \\
& \text{s.t.} && Ax \leq b \\
& && x_i = 0 && \forall i \in I_\nu^0 \\
& && x_i = 1 && \forall i \in I_\nu^1
\end{aligned}
$$

Lower bounding with Linear Programming relaxation

At a certain node ν of the tree, we have a subproblem of the form

$$
\begin{aligned}
(\mathcal{P}_\nu) \quad & \min_{x \in \{0,1\}^n} && c^\top x \\
& \text{s.t.} && Ax \leq b \\
& && x_i = 0 && \forall i \in I_\nu^0 \\
& && x_i = 1 && \forall i \in I_\nu^1
\end{aligned}
$$

where $I_\nu = I_\nu^0 \cup I_\nu^1$ is the set of indices of variables fixed to 0 or 1.

A lower bound on the optimal value of $(\mathcal{P}_\nu)$ is the value of its linear programming relaxation:

$$
\begin{aligned}
(\mathcal{P}_\nu^{LP}) \quad & \min_{x \in [0,1]^n} && c^\top x \\
& \text{s.t.} && Ax \leq b \\
& && x_i = 0 && \forall i \in I_\nu^0 \\
& && x_i = 1 && \forall i \in I_\nu^1
\end{aligned}
$$

Since the integer feasible set is contained in the relaxed feasible set

Branch and Bound algorithm outline

- Start at the root node with no variables fixed. The best integer-feasible solution found so far is called the incumbent; initialize its value to $+\infty$.
- At each node:
  1. Solve the LP relaxation to get a lower bound.
  2. If the LP is infeasible, prune the node.
  3. If the lower bound is greater than or equal to the incumbent value, prune the node.
  4. If the LP solution is binary (in particular, this always happens at a feasible leaf), update the incumbent.
  5. Otherwise, branch on a fractional variable and create two child nodes.
- Repeat until all nodes are processed or pruned.

Branch and Bound algorithm – pseudocode

Data: BIP: $\min\{c^\top x \mid Ax \leq b,\ x \in \{0,1\}^n\}$
Result: An optimal solution and its value, or a declaration that the BILP is infeasible
$z^* \leftarrow +\infty$, $x^* \leftarrow \varnothing$
Create root node ν_{root} with $I^0 = I^1 = \varnothing$ and push ν_{root} into list L
while $L \neq \varnothing$ **do**
  Extract a node ν from L
  Solve LP relaxation $(\mathcal{P}_\nu^{LP})$ for (x^{LP}, z^{LP})
  if *LP infeasible* **then**
    discard node
  else if $z^{LP} \geq z^*$ **then**
    discard node `// prune by bound`
  else if $x^{LP} \in \{0,1\}^n$ **then**
    Update $(x^*, z^*) \leftarrow (x^{LP}, z^{LP})$ `// integer feasible candidate`
  else
    Select an index i with $x_i^{LP} \notin \{0,1\}$ `// branching rule`
    Create child ν_0 with $I_{\nu_0}^0 = I_\nu^0 \cup \{i\}$, $I_{\nu_0}^1 = I_\nu^1$ and push ν_0 into L
    Create child ν_1 with $I_{\nu_1}^1 = I_\nu^1 \cup \{i\}$, $I_{\nu_1}^0 = I_\nu^0$ and push ν_1 into L
if $x^* = \varnothing$ **then**

Branch and bound on knapsack example (minimization formulation)

To keep the minimization convention used throughout the course, write knapsack as

$$\text{Min} -\sum_{i=1}^{n} v_i x_i \qquad \text{s.t.} \qquad \sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0,1\}.$$

- The knapsack linear relaxation (at any node) is very easy to solve: compute the efficiency v_i/w_i of each item and add them by decreasing efficiency up to maximum weight.
- The branch and bound algorithm would go as follows:
  1. Solve linear relaxation and choose the fractional object in the linear solution to branch on.
  2. Set the object to 1 (i.e. add it to the knapsack) and solve the new LP relaxation.
     - ⋆ If the LP lower bound is higher than current upper bound, prune the tree (i.e. no need to explore further solution with this object).
     - ⋆ If the solution is integer you know the best solution for $(\mathcal{P}_\nu)$, and update problem upper bound.
     - ⋆ Otherwise branch on the fractional solution and add two nodes to explore.
  3. Set the object to 0 (i.e. do not add it to the knapsack) and proceed similarly.

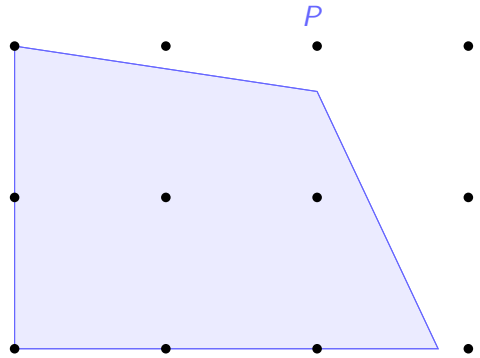
Contents

Polyhedron and integer hull

- For simplicity, consider first the pure-integer case.
- Let P be the polyhedron defined by the linear constraints.
- The integer-feasible points are $P \cap \mathbb{Z}^n$.
- Their convex hull

$$P^I := \text{conv}(P \cap \mathbb{Z}^n)$$

is the integer hull.

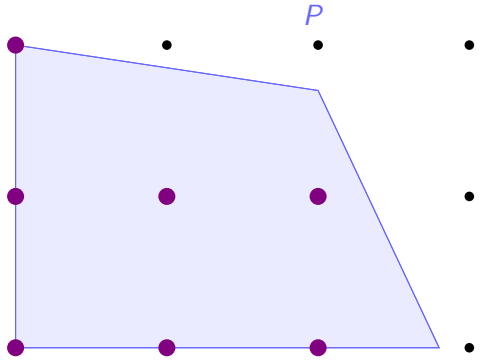


Polyhedron and integer hull

- For simplicity, consider first the pure-integer case.
- Let P be the polyhedron defined by the linear constraints.
- The integer-feasible points are $P \cap \mathbb{Z}^n$.
- Their convex hull

$$P^I := \text{conv}(P \cap \mathbb{Z}^n)$$

is the integer hull.

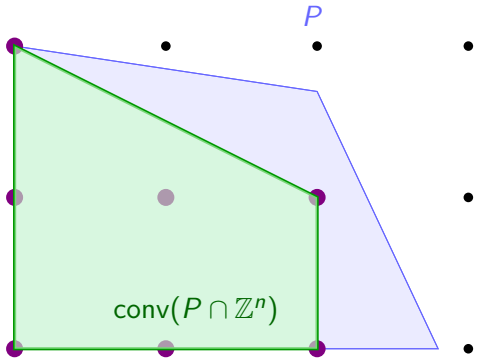


Polyhedron and integer hull

- For simplicity, consider first the pure-integer case.
- Let P be the polyhedron defined by the linear constraints.
- The integer-feasible points are $P \cap \mathbb{Z}^n$.
- Their convex hull

$$P^I := \text{conv}(P \cap \mathbb{Z}^n)$$

is the integer hull.

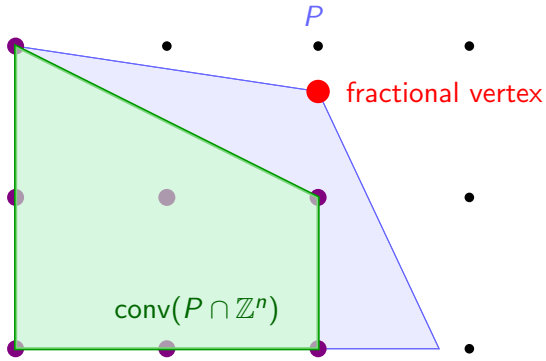


Polyhedron and integer hull

- For simplicity, consider first the pure-integer case.
- Let P be the polyhedron defined by the linear constraints.
- The integer-feasible points are $P \cap \mathbb{Z}^n$.
- Their convex hull

$$P^I := \text{conv}(P \cap \mathbb{Z}^n)$$

is the integer hull.



Polyhedron and integer hull

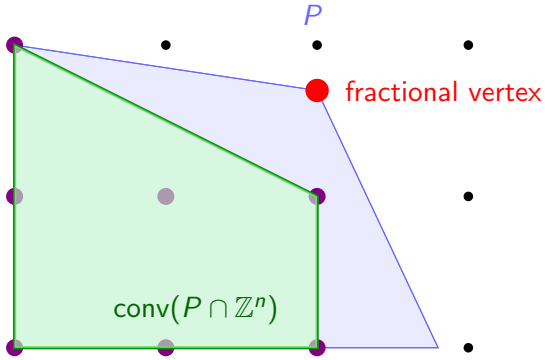
- For simplicity, consider first the pure-integer case.
- Let P be the polyhedron defined by the linear constraints.
- The integer-feasible points are $P \cap \mathbb{Z}^n$.
- Their convex hull

$$P^I := \text{conv}(P \cap \mathbb{Z}^n)$$

  is the integer hull.
- We have

$$\min_{x \in P} c^\top x \leq \min_{x \in P \cap \mathbb{Z}^n} c^\top x = \min_{x \in P^I} c^\top x.$$

- If the LP solution is integer, it is optimal for the integer problem.



Main ideas

- At each node of the branch-and-bound tree, we solve the LP relaxation to get a lower bound on the optimal value of the MILP in that subproblem.
- If the LP solution is integer, it is optimal for the MILP in that subproblem.
- If a general integer variable has a fractional LP value x_i^{LP}, branch with

$$x_i \leq \lfloor x_i^{LP} \rfloor \qquad \text{or} \qquad x_i \geq \lceil x_i^{LP} \rceil.$$

- We prune branches where the lower bound exceeds the incumbent value.

Practical consequences

- The efficiency of branch-and-bound depends on the quality of the LP relaxations, in particular on the integrality gap between the LP and integer optima.
- Stronger formulations (closer to the integer hull) lead to better bounds and more effective pruning $\rightsquigarrow$ sometimes reformulating the MILP with more variables and inequalities can significantly improve performance.
- Valid inequalities, *i.e.*linear constraints satisfied by all integer solutions, can be added to tighten the LP relaxation.
- The choice of branching variable can significantly affect the size of the search tree.

➥ Advanced techniques like cutting planes, heuristics, and preprocessing are often integrated into branch-and-bound algorithms (and solvers) to enhance performance.

➥ Finding problem-specific valid inequalities can improve the efficiency of your solver.

Contents

Modeling with MILP

- MILP is a flexible framework that can model a wide range of problems.
- Some common modeling techniques are required to express non-linear or logical relationships using linear constraints and integer variables.
- Binary variables are often used to represent decisions such as on/off, yes/no, or selection choices.

- Recall that there are many ways of formulating the same problem as a MILP, with very different computational performances.

Modeling with MILP

- MILP is a flexible framework that can model a wide range of problems.
- Some common modeling techniques are required to express non-linear or logical relationships using linear constraints and integer variables.
- Binary variables are often used to represent decisions such as on/off, yes/no, or selection choices.

- Recall that there are many ways of formulating the same problem as a MILP, with very different computational performances.

Binary variables

Binary variables are often used as indicator variables, for instance

$$b = \mathbb{1}_{\{x \in X\}}.$$

Terminology collision: this $0/1$ indicator is different from the extended-value indicator function I_X used earlier in convex analysis.

Here are some examples:

- production setting: $b_{i,t} = 1$ if unit i is on at time t;
- graph setting: $b_{i,j} = 1$ if the edge (i, j) is selected;
- planning setting: $b_{i,j} = 1$ if worker i is assigned to task j;
- ...

Big-M constraint

- Big-M constraints are used to model conditional relationships.
- They introduce a large constant M to "turn on" or "turn off" constraints based on the value of binary variables.
- For example, to model that x can be positive only if $b = 1$:

$$x \leq M \cdot b$$
$$x \geq 0$$
$$b \in \{0, 1\}$$

- The choice of M is crucial: it should be as small as possible while still being valid to avoid numerical instability and weak relaxations.

semi-continuous variables

A common constraint in production planning is the semi-continuous variable constraint: that is, a variable that is either zero or lies within a specified range $[\underline{x}, \overline{x}]$.

This can be modeled using a binary variable b as follows:

semi-continuous variables

A common constraint in production planning is the semi-continuous variable constraint: that is, a variable that is either zero or lies within a specified range $[\underline{x}, \overline{x}]$.

This can be modeled using a binary variable b as follows:

$$x = 0 \quad \text{or} \quad x \in [\underline{x}, \overline{x}] \qquad \Longleftrightarrow \qquad \begin{cases} x \leq \overline{x} \cdot b \\ x \geq \underline{x} \cdot b \\ b \in \{0, 1\} \end{cases}$$

Logical constraints

- Logical constraints model relationships between binary variables.
- They can be expressed using additional binary variables and linear inequalities.
- For example
  - "if $b_1 = 1$ then $b_2 = 1$":
  - " $b_1 = 1$ OR $b_2 = 1$":
  - "$b_1 = 1$ XOR $b_2 = 1$" (one and only one is 1):

Logical constraints

- Logical constraints model relationships between binary variables.
- They can be expressed using additional binary variables and linear inequalities.
- For example
  - "if $b_1 = 1$ then $b_2 = 1$":

$$b_2 \geq b_1$$
$$b_1, b_2 \in \{0, 1\}$$

  - " $b_1 = 1$ OR $b_2 = 1$":

$$b_1 + b_2 \geq 1$$
$$b_1, b_2 \in \{0, 1\}$$

  - "$b_1 = 1$ XOR $b_2 = 1$" (one and only one is 1):

$$b_1 + b_2 = 1$$
$$b_1, b_2 \in \{0, 1\}$$

Number of active variables

- Sometimes we want to limit the number of active continuous variables $x_i \geq 0$.
- Introduce activation binaries $b_i \in \{0, 1\}$ and linking constraints

$$x_i \leq M_i b_i.$$

Then

guarantees that at most k variables can be positive.

- Be careful: $\sum_i b_i = k$ selects exactly k activation binaries, but some corresponding x_i may still be zero.
- If $0 < \underline{x}_i \leq \overline{x}_i$ are known, use

$$\underline{x}_i b_i \leq x_i \leq \overline{x}_i b_i.$$

Then $\sum_i b_i = k$ enforces exactly k positive variables.

Number of active variables

- Sometimes we want to limit the number of active continuous variables $x_i \geq 0$.
- Introduce activation binaries $b_i \in \{0, 1\}$ and linking constraints

$$x_i \leq M_i b_i.$$

Then

$$\sum_{i=1}^{n} b_i \leq k$$

guarantees that at most k variables can be positive.

- Be careful: $\sum_i b_i = k$ selects exactly k activation binaries, but some corresponding x_i may still be zero.
- If $0 < \underline{x}_i \leq \overline{x}_i$ are known, use

$$\underline{x}_i b_i \leq x_i \leq \overline{x}_i b_i.$$

Then $\sum_i b_i = k$ enforces exactly k positive variables.

Piecewise linear functions – nonconvex case

Consider breakpoints $(x_0, f_0), \ldots, (x_k, f_k)$. For a general piecewise-linear function, we must select the active segment.

Introduce $b_i \in \{0, 1\}$ and $t_i \in [0, 1]$, $i \in [k]$:

$$\sum_{i=1}^{k} b_i = 1, \qquad 0 \le t_i \le b_i.$$

Then the graph is represented exactly by

$$x = \sum_{i=1}^{k} \Big(x_{i-1} b_i + (x_i - x_{i-1}) t_i \Big),$$

$$f = \sum_{i=1}^{k} \Big(f_{i-1} b_i + (f_i - f_{i-1}) t_i \Big).$$

Exactly one segment is selected, and t_i gives the position on that segment.

Piecewise linear functions – convex case

- For a convex piecewise-linear function, its epigraph can be represented without binary variables.
- Given breakpoints (x_i, f_i), introduce weights $\lambda_i \geq 0$ with

$$\sum_{i=1}^{k} \lambda_i = 1.$$

- The epigraph $t \geq f(x)$ is modeled by

$$x = \sum_{i=1}^{k} x_i \lambda_i, \qquad t \geq \sum_{i=1}^{k} f_i \lambda_i.$$

- In a minimization model where t is penalized, the inequality is tight at optimality: $t = f(x)$.
- Without convexity, this construction gives the convex envelope, not the original graph.

Binary products and McCormick inequalities

- The product $y = b_1 b_2$ of two binary variables $b_1, b_2 \in \{0, 1\}$ can be linearized with a *continuous* variable y:

$$
\begin{aligned}
y &\leq b_1 \\
y &\leq b_2 \\
y &\geq b_1 + b_2 - 1 \\
y &\geq 0
\end{aligned}
$$

- The same idea extends to the product $y = bx$ of a binary variable b and a continuous variable $x \in [\underline{x}, \overline{x}]$:

$$
\begin{aligned}
y &\leq \overline{x} \cdot b \\
y &\geq \underline{x} \cdot b \\
y &\leq x - \underline{x} \cdot (1 - b) \\
y &\geq x - \overline{x} \cdot (1 - b)
\end{aligned}
$$

Binary products and McCormick inequalities

- The product $y = b_1 b_2$ of two binary variables $b_1, b_2 \in \{0, 1\}$ can be linearized with a *continuous* variable y:

$$
\begin{aligned}
y &\leq b_1 \\
y &\leq b_2 \\
y &\geq b_1 + b_2 - 1 \\
y &\geq 0
\end{aligned}
$$

- The same idea extends to the product $y = bx$ of a binary variable b and a continuous variable $x \in [\underline{x}, \overline{x}]$:

$$
\begin{aligned}
y &\leq \overline{x} \cdot b \\
y &\geq \underline{x} \cdot b \\
y &\leq x - \underline{x} \cdot (1 - b) \\
y &\geq x - \overline{x} \cdot (1 - b)
\end{aligned}
$$

Contents

What you should know

- What a MILP/BILP is and why integrality makes optimization much harder.
- The role of the LP relaxation and the integrality gap.
- The principle of branch-and-bound: bound, branch, maintain an incumbent, prune.
- Why stronger formulations lead to better bounds and faster branch-and-bound.
- The role of binary variables and Big-(M) constraints in modeling discrete decisions.

What you should be able to do

- Formulate simple discrete decisions using binary and integer variables.
- Write and interpret the LP relaxation of a MILP.
- Execute a small branch-and-bound tree.
- Model basic logical/on-off relations with binary variables, including a valid Big-(M) formulation.