Optimization

Lecture 12: Integer linear programming — Branch-and-bound algorithms

Parin Chaipunya
KMUTT
└Mathematics @ Faculty of Science
└The Joint Graduate School of Energy and Environments

Areas of research:
- Multi-agent optimization: Bilevel programs, Game theory
- Optimization modeling: mainly focused on energy and environmental applications

parin.cha@kmutt.ac.th
parinchaipunya.com
github.com/parinchaipunya

Last update: Januaray 2026

Table of contents

Section 1

Integer linear programs

Integer linear programs

An **integer linear program** (briefly **ILP**) is a linear program with additional **integrality constraints** saying that some of the variables are only allowed to be integers. Formally, it takes the following form

$$\begin{cases} \min & f^t x & (1a) \\ \text{s.t.} & Ax \geq b & (1b) \\ & x \geq 0 & (1c) \\ & x_k \in \mathbb{Z} \quad (\forall k \in \mathcal{K}), & (1d) \end{cases}$$

where $\mathcal{K} \subset \{1, \ldots, n\}$ and n being the dimension of x. Those variables x_k's with $k \in \mathcal{K}$ are referred to as **integer variables**.

A typical type of integer variables is the **binary** variables, those taking either 0 or 1 as their values.

A binary variable x_k could be expressed as $\begin{cases} 0 \leq x_k \leq 1 \\ x_k \in \mathbb{Z} \end{cases}$, but more favorably as $x_k \in \{0, 1\}$.

Relaxed problems

Given any ILP of the form (1), we define its **relaxed problem** to be the LP restraining all the constraints but all the integrality requirements. That is, the relaxed problem reads

$$\begin{cases} \min & f^t x & \text{(2a)} \\ \text{s.t.} & Ax \geq b & \text{(2b)} \\ & x \geq 0\,. & \text{(2c)} \end{cases}$$

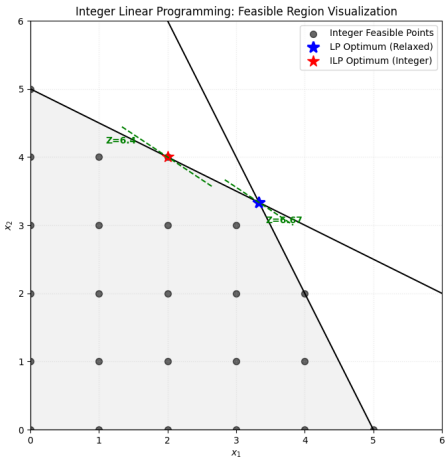
For convenience, we write F to denote the **relaxed feasible set** $F = \{x \mid Ax \geq b,\ x \geq 0\}$.

It is important to note that

$$z^*_{\text{relaxed}} \leq z^*,$$

where z^* and z^*_{relaxed} are, respectively, the optimal objective values of the original problem (1) and relaxed problem (2).

Some illustrations

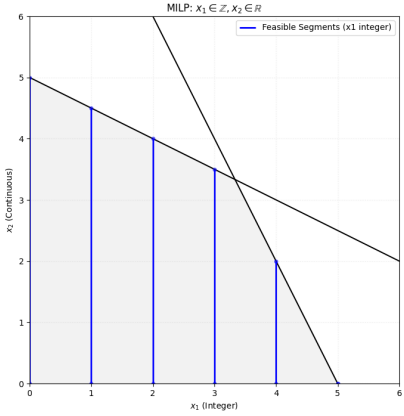


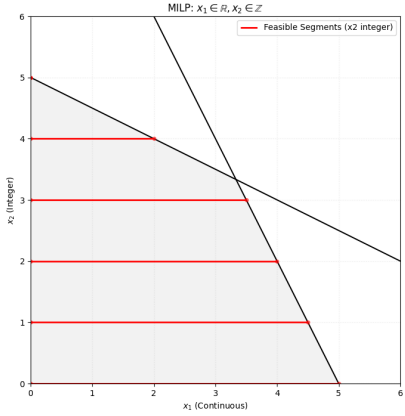
This figure illustrates the core idea of an integer program.

- Here, both variables x_1 and x_2 are integers.
- The feasible points are those dots lying inside the shaded area (the relaxed continuous feasible region).
- The relaxed optimal point can be different from the true integer optimal point.

Some illustrations

There are also problems where *some* of the variables are required to be integers. These are called **mixed-integer problems**.





An example (Production)

Example 1

A carpenter house produces tables and chairs.

- Each table takes 10 hours and each chair takes 3 hours to produce.
- Each table uses 2 blocks of woods and each chair takes 1 block.
- The workshop has a warehouse that stores up to 25 items of tables and chairs combined.
- Each table sells for 500 baht and each chair sells for 150 baht.

Questions

(a) How many tables and chairs should be produced within 100 laborhours to maximizes the sales?

(b) Why can't we just *round down* the continuous solution?

An example (Assignment)

Example 2

You are the head of a laboratory, and you want to assign three jobs to three assistants in your laboratory. Because your assistants have different backgrounds and trainings, the time it takes them to complete different tasks vary. The following table summarize the time (hours) that each assistant requires to complete the three jobs.

	Job A: Coding	Job B: Proofreading	Job C: Data cleaning
Assistant 1	9	2	7
Assistant 2	6	4	3
Assistnat 3	5	8	10

Questions

(a) What is the best way to assign each job to an assistant to minimize the time to complete all the jobs?

(b) Why can't we just *round down* the continuous solution?

Section 2

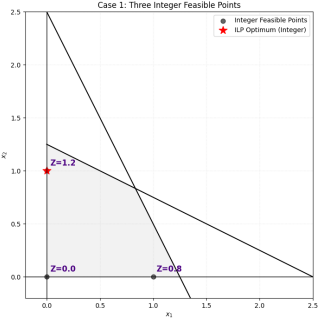
Branch-and-bound algorithms

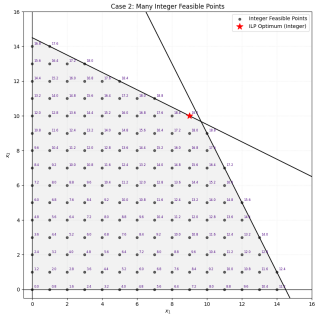
Subsection 1

Overview

Full enumeration (Brute-force)

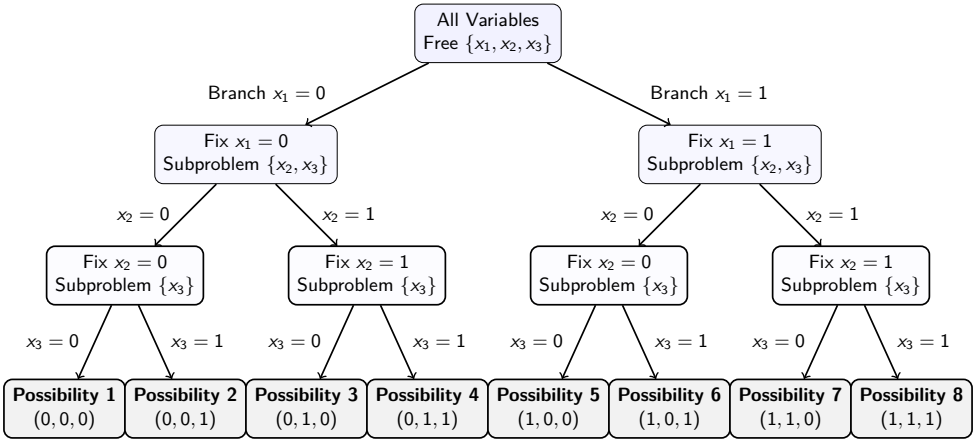
In the simplest cases, an ILP has a bounded relaxed feasible region F, and thus contains only finitely many integer grid points. In principle, one could evaluate the objective function at each of these points and select an optimal solution.





However, even in low dimensions, the number of such feasible points can be prohibitively large. This motivates the use of more efficient methods, such as branch-and-bound algorithms.

Brute-force as a tree



Branch-and-bound algorithms

Instead of writing the complete tree and examine all the nodes (*i.e.* brute-force), the **branch-and-bound algorithm** (briefly, **B&B algorithm**) suggests a way to explore further on promising nodes and to prune unfavorable branches.

The branch-and-bound algorithm is operated by iteratively applying three core steps, **starting from the fully relaexed problem**.

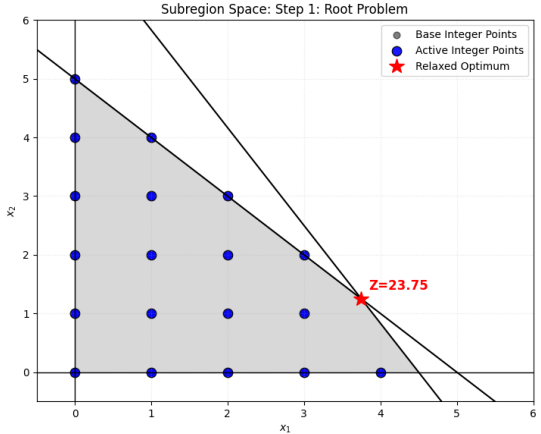
- **Branching.** This step happens at a selected node, where the feasible region is divided into smaller, mutually exclusive subproblems (nodes).
- **Bounding.** At each node, the guaranteed lower and upper bounds of the original problem are evaluated. These bounds allow us to make a decision whether to keep or discard a branch.
- **Pruning (or Fathoming).** If a node has a worse bound that the other one, then it is not certified to contain a solution. The whole branch is then discarded (or pruned).

Branch-and-bound algorithms (Visualization)

Let us make a visualization fo the branch-and-bound algorithm applied to the following maximization problem.

$$\begin{cases} \max & 5x_1 + 4x_2 & \\ \text{s.t.} & x_1 + x_2 & \leq 5 \\ & 10x_1 + 7x_2 & \leq 45 \\ & x_1, x_2 & \geq 0 \end{cases}$$

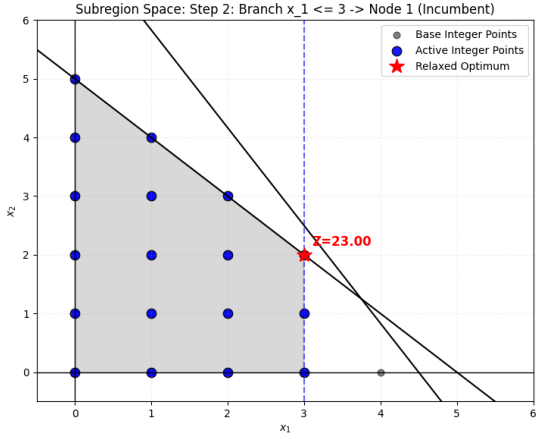
Branch-and-bound algorithms (Visualization)

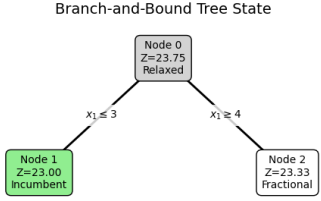


Branch-and-Bound Tree State

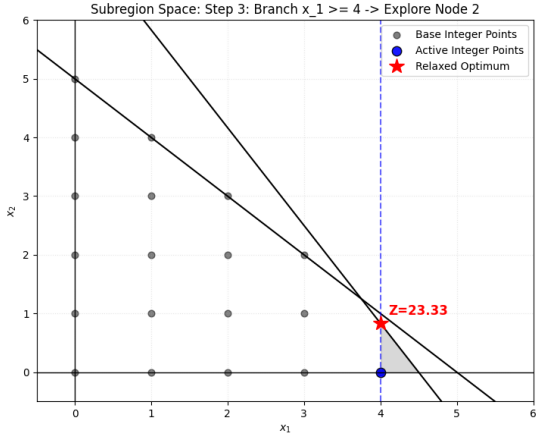
Node 0
Z=23.75
Relaxed

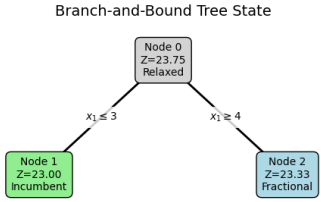
Branch-and-bound algorithms (Visualization)



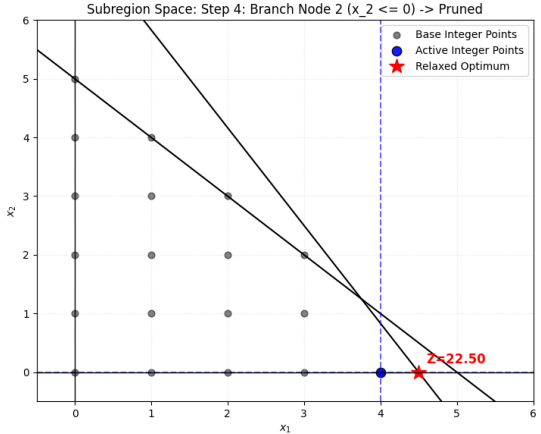


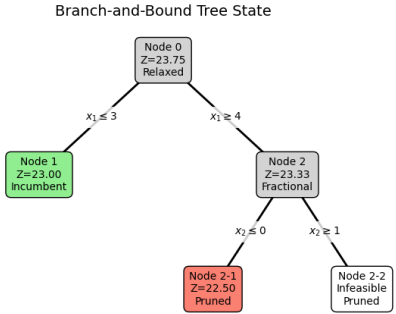
Branch-and-bound algorithms (Visualization)



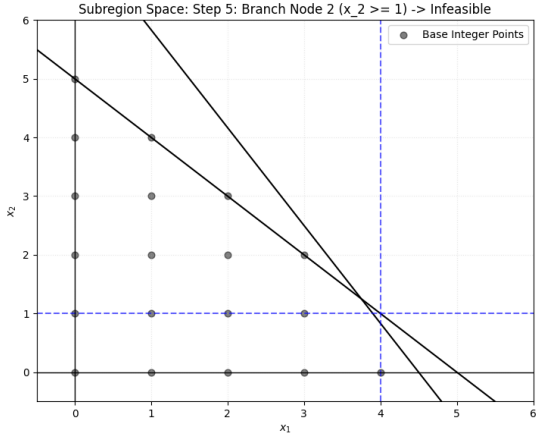


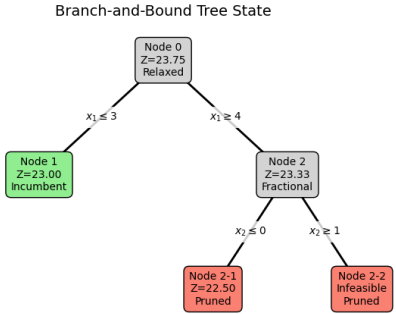
Branch-and-bound algorithms (Visualization)





Branch-and-bound algorithms (Visualization)





Subsection 2

The algorithm

Branch-and-bound algorithm

Consider the ILP defined in (1), and let F denote the relaxed feasibility region, *i.e.* the points x satisfying (1b)–(1c).

Since we shall face with many subproblems, we shall refer to the (1) itself as the **original problem**. Moreover, we emphasize the feasible points of the original problem as **ILP-feasible points**.

Branch-and-bound algorithm

Before we state the precise steps of the B&B algorithm, let us look at a breakdown of the core idea of the B&B algorithm.

- The algorithm starts from the totally relaxed problem and then generate a binary tree where each node has a lower bound attached to it.
- The upper bound is always global, and it is updated once a better ILP-feasible point is found.
- The algorithm is terminated if the gap between the upper and lower bounds is closed, or is close enough.

Branch-and-bound algorithm

The **branch-and-bound algorithm** (briefly, **B&B algorithm**) follows the steps below.

Step 1: Initialization

- Solve the fully relaxed problem $\min_{x \in F} f^t x$.
  If the optimal solution $\bar{x}$ obtained here satisfies the integrality condition (1d), STOP and $\bar{x}$ is the solution of the original problem.
  - The optimal objective value obtained here is recorded as the **global lower bound** z_L.
  - If an ILP-feasible point x_0 is known, set the **global upper bound** as $z_U = f^t x_0$.
    Otherwise, put $z_U = +\infty$.

Branch-and-bound algorithm

Step 2: Branching

- Choose an unpruned node h to branch on. Assume that $\bar{x}^h$ is the current relaxed optimal point at this node.
  - Choose a component $k \in \mathcal{K}$ such that $\bar{x}_k^h$ is not an integer, say $\lfloor \bar{x}_k^h \rfloor < \bar{x}_k^h < \lceil \bar{x}_i^h \rceil$.
  - Branch into two new nodes (namely $(h, 1)$ and $(h, 2)$) by adding a new constraint $\lfloor \bar{x}_i \rfloor \leq x$ to the first node, and adding $x \leq \lfloor \bar{x}_i \rfloor$ to the second.

Branch-and-bound algorithm

Step 3: Bounding

- At each $(h, j) \in \{(h, 1), (h, 2)\}$, solve the corresponding optimization subproblem to obtain the optimal value $\bar{z}^{h,j}$ and an optimal point $\bar{x}^{h,j}$.
- Record the lower bound at this node (h, j) by setting $z_L^{h,j} = \bar{z}^{h,j}$.

Branch-and-bound algorithm

Step 4: Pruning

- At each node h, if one of the following conditions is true
  - (a) $z_L^h \geq z_U$, or
  - (b) the subproblem at the node h is infeasible, or
  - (c) z_L^h is obtained at an ILP-feasible point $\bar{x}^h$ and $z_L^h < z_U$,

  then the node h is pruned.
- In case the node h is pruned by (c), the new global upper bound is set $z_U = z_L^h$.

Branch-and-bound algorithm

Step 5: Iterate

- If there are still unpruned nodes, repeat Steps 2–4.
  Otherwise, STOP and there is a node h^* such that $z_L^{h^*} = z_U$. Then the original problem has a minimum of $z_L^{h^*}$ which is attained at the corresponding point $\bar{x}^{h^*}$ at the node h^*.

Section 3

Examples

Subsection 1

The first example (Breadth first vs Depth first approaches)

The first example

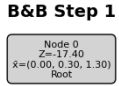
Let us consider the following ILP.

$$\begin{cases} \min & 3x_1 - 7x_2 - 12x_3 & & (3a) \\ \text{s.t.} & -3x_1 + 6x_2 + 8x_3 \leq 12 & & (3b) \\ & 6x_1 - 3x_2 + 7x_3 \leq 8 & & (3c) \\ & -6x_1 + 3x_2 + 3x_3 \leq 5 & & (3d) \\ & x_1, x_2, x_3 \geq 0 & & (3e) \\ & x_1, x_2, x_3 \in \mathbb{Z} & & (3f) \end{cases}$$

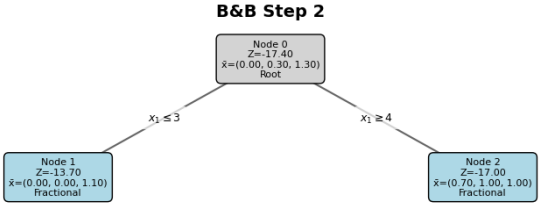
Following this will be the B&B tree solving (3).

We shall present two branching strategies, namely the **breadth first** and the **depth first** approaches.

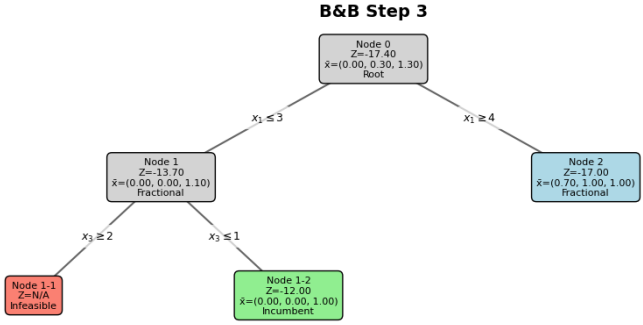
The first example [Breadth first]



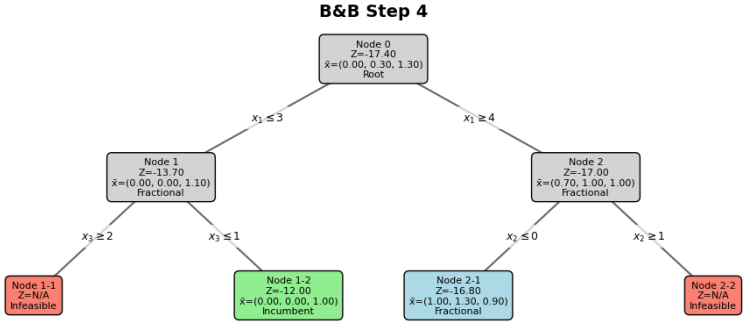
The first example [Breadth first]



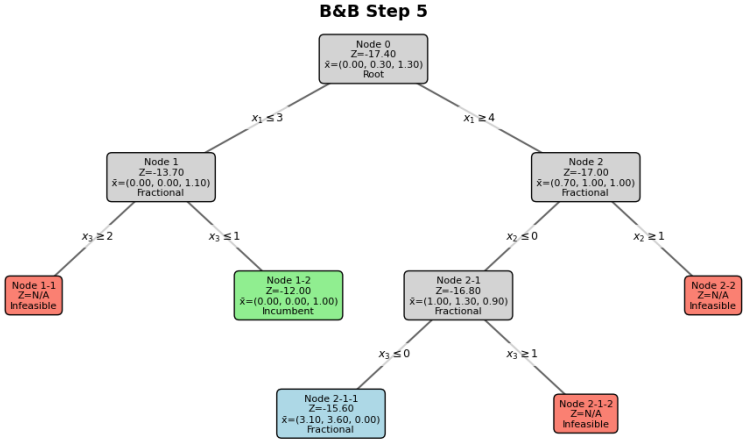
The first example [Breadth first]



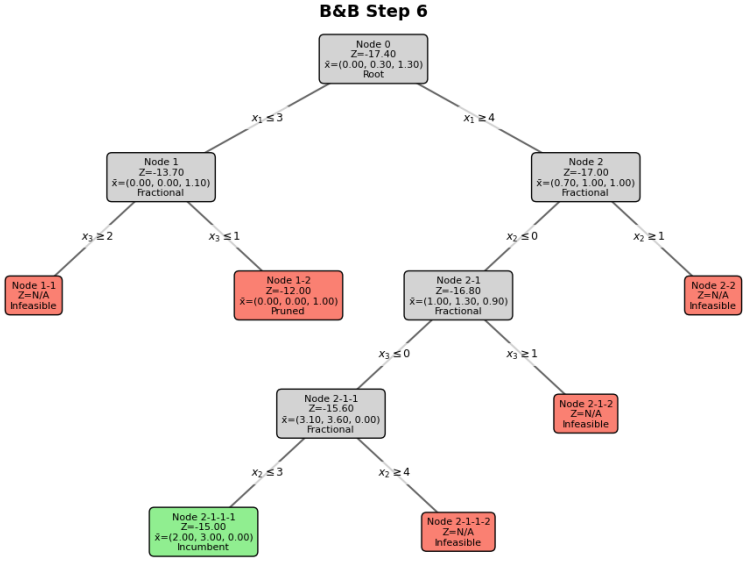
The first example [Breadth first]



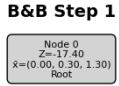
The first example [Breadth first]



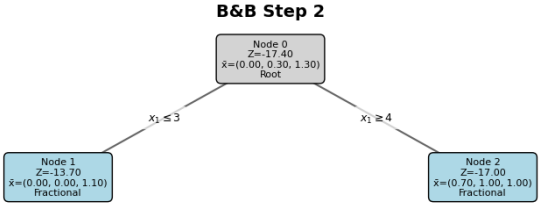
The first example [Breadth first]



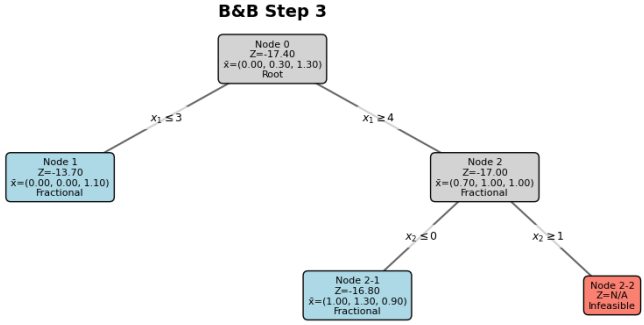
The first example [Depth first]



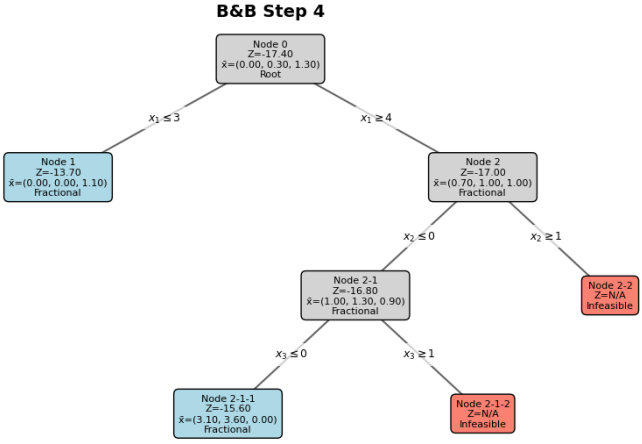
The first example [Depth first]



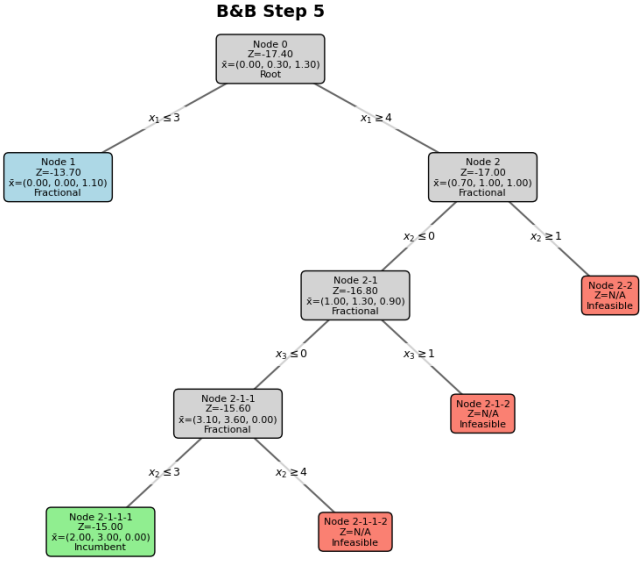
The first example [Depth first]



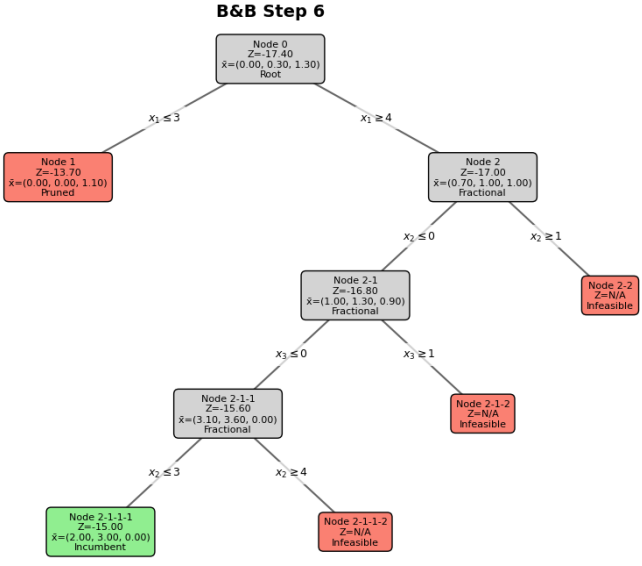
The first example [Depth first]



The first example [Depth first]



The first example [Depth first]



Breadth first vs Depth first

In general, either of the approaches has pros and cons and they do not promise a better performance over the other ones.

The **breadth first approach** explores the search tree level-by-level, resulting in a better *global view* of the tree. However, the approach usually ends up with a lot of nodes waiting to be explored so that it requires more memory.

In contrast, the **depth first approach** dives along one branch as far as possible before backtracking. This leaves smaller number of nodes to track and hence requires less memory. The depth first approach usually finds quickly an ILP-feasible point but not necessarily good bounds at the early stages.

Subsection 2

A binary example

A binary example

A minimalist hiker has a small bag that holds up to 7 kg and he has to decide which of the following three items to bring:

Item (i)	**Weight** (w_i)	**Utility** (v_i)
Portable grill	5 kg	10
Camera kit	3 kg	7
Luxury tent	4 kg	8

Which items to bring if he wants to maximize the utility?

A binary example

The ILP formulation of this problem reads

$$\begin{cases} \max & 10x_1 + 7x_2 + 8x_3 \\ \text{s.t.} & 5x_1 + 3x_2 + 4x_3 \leq 7 \\ & x_1, x_2, x_3 \in \{0, 1\}. \end{cases}$$

To make it compatible with the B&B method, we equivalently reformulate it into

$$\begin{cases} \max & 10x_1 + 7x_2 + 8x_3 & \text{(4a)} \\ \text{s.t.} & 5x_1 + 3x_2 + 4x_3 \leq 7 & \text{(4b)} \\ & x_1, x_2, x_3 \leq 1 & \text{(4c)} \\ & x_1, x_2, x_3 \geq 0 & \text{(4d)} \\ & x_1, x_2, x_3 \in \mathbb{Z}. & \text{(4e)} \end{cases}$$

A binary example

Note that when a variable x_i is fractional, $0 < x_i < 1$, the branching process splits a node into two cases:

$$x_i \leq 0 \quad \text{or} \quad x_i \geq 1\,.$$

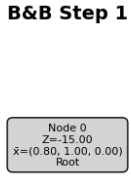
.

However, due to the constraints (4c) and (4d), this effectively reduces to fixing x_i at either

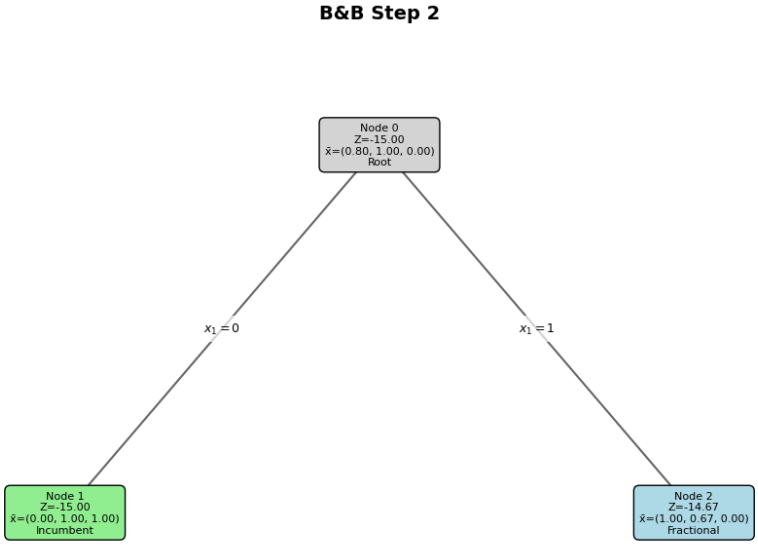
$$x_i = 0 \quad \text{or} \quad x_i = 1\,.$$

.

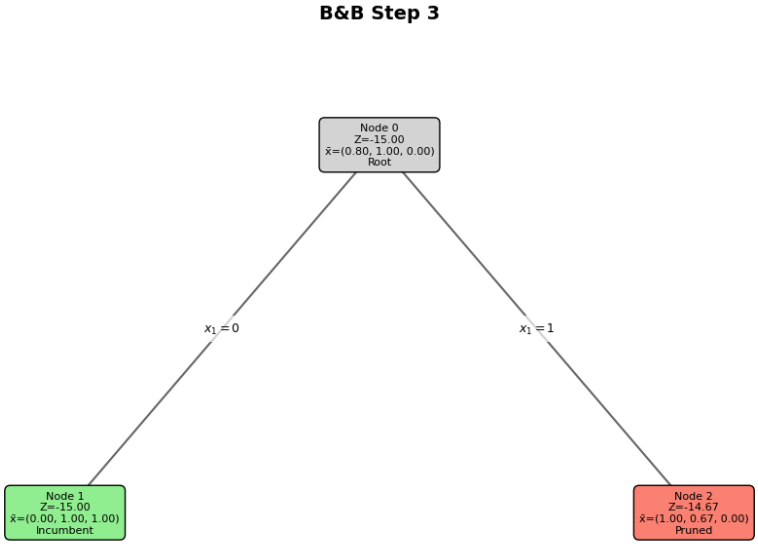
A binary example



A binary example



A binary example



Subsection 3

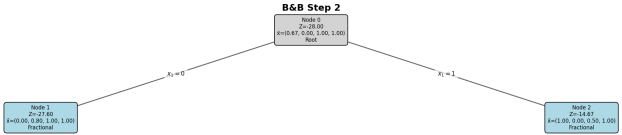
A bigger binary example

A bigger binary example

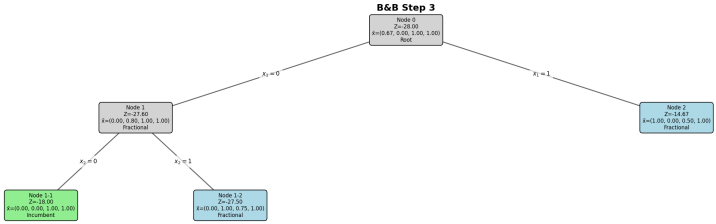
$$\begin{cases} \max & 15x_1 + 12x_2 + 10x_3 + 8x_4 & \\ \text{s.t.} & 6x_1 + 5x_2 + 4x_3 + 3x_4 & \leq 11 \\ & x_1, x_2, x_3, x_4 & \leq 1 \\ & x_1, x_2, x_3, x_4 & \geq 0 \\ & x_1, x_2, x_3, x_4 & \in \mathbb{Z}\,. \end{cases}$$

A bigger binary example

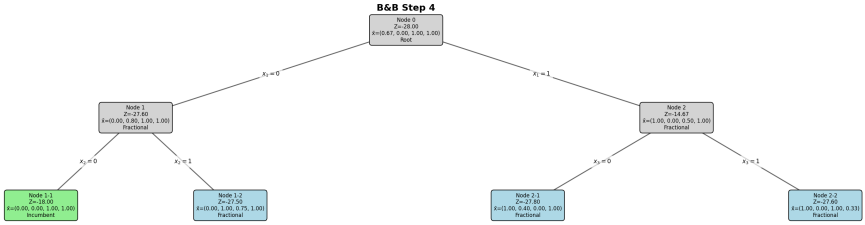
A bigger binary example



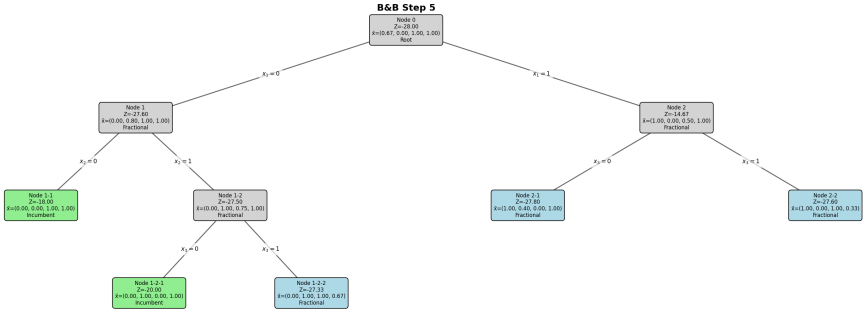
A bigger binary example



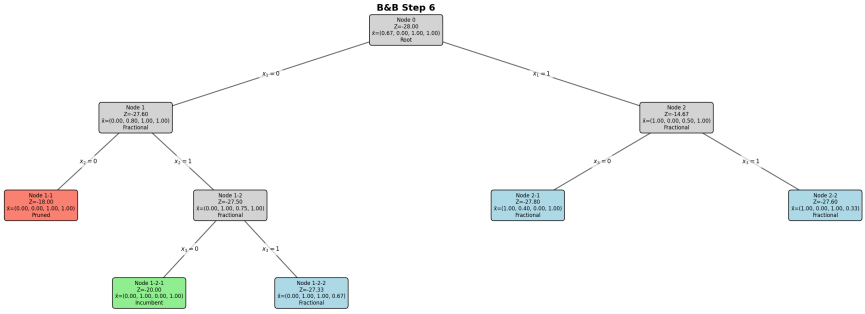
A bigger binary example



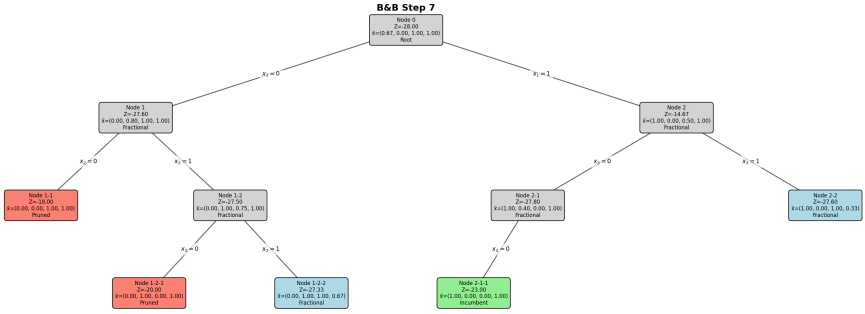
A bigger binary example



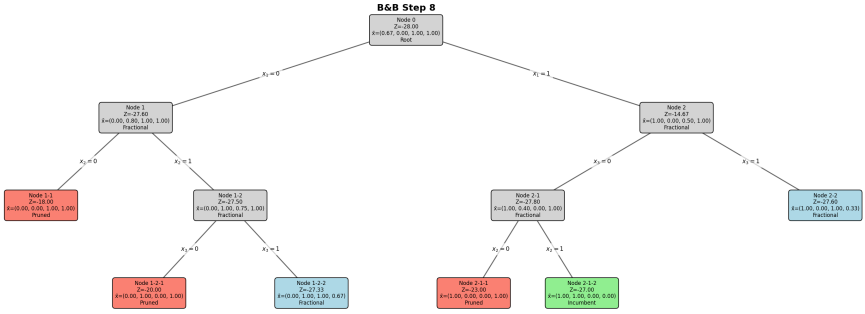
A bigger binary example



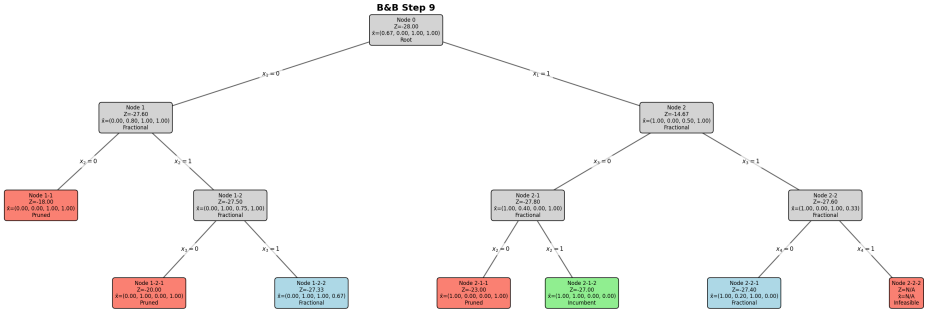
A bigger binary example



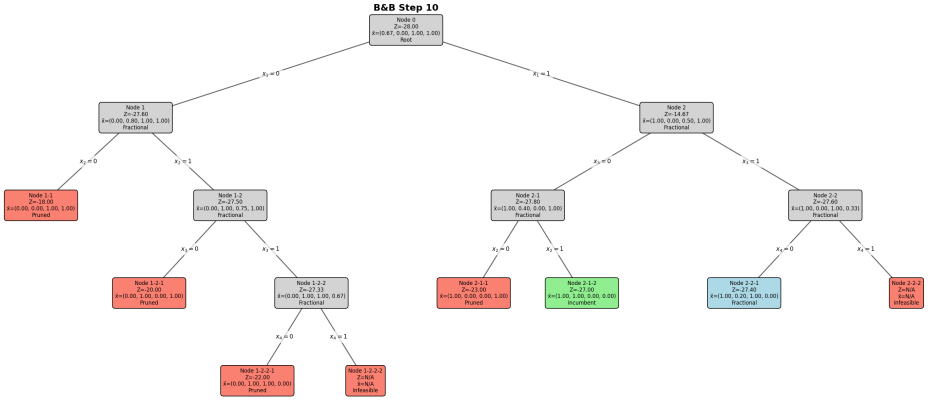
A bigger binary example



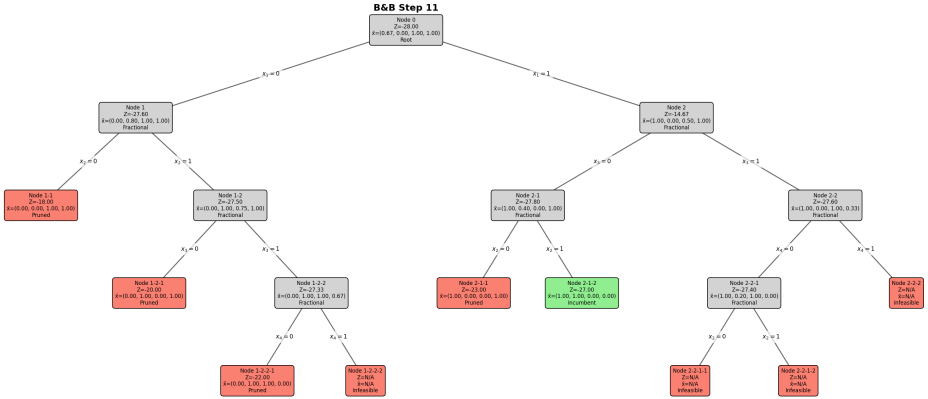
A bigger binary example



A bigger binary example



A bigger binary example



That's it!

Key takeaways.

- The branching process.
- The pruning methods.

Thank you.