2025 Fall CSE5025

Combinatorial Optimization

组合优化

Instructor: 刘晟材

Lecture 5-1: Integer Programming for Combinatorial Optimization – Part I

Agenda for Today's Lecture

In this lecture, we will focus on

- What is integer programming?
- Why do we need integer programming?
- The modeling-and-solving paradigm of integer programing

Learning Objectives for this lecture

- Master the basic idea of integer programming
- Master the techniques for modeling CO problems as integer programs
- Know the and idea and techniques for solving integer programs

Why a New Technique?

In previous lectures, we have seen specific algorithms for specific problems:

- Set Cover: Greedy Algorithm (approximation) and DP (exact, exponential)
- Knapsack: DP (exact, pseudo-polynomial), FPTAS (approximation)
- TSP: Christofides (approximation), Held-Karp DP (exact, exponential)

However: These algorithms are problem-specific. A DP for Knapsack doesn't help with TSP. **Designing a new algorithm for every new problem is hard.**

Question:

- Is there a **general, unified framework** to define and solve all these problems?

Answer:

- Yes. **Integer Programming (整数规划)**

Three Core Components of a CO Problem

Decision Variables (决策变量): What are our decisions?

- *Example*: Which items to pick? Which sets to choose? Which path to take?

Constraints (约束): What are the rules we must follow?

- *Example*: Must cover all elements? Must not exceed capacity? Must visit every city once?

Objective Function (目标函数): What are we trying to optimize?

- *Example*: Minimize total cost? Maximize total value?

Integer program (IP) is a formal mathematical language for writing down these three components.

Formal Definition: Linear Programming

A **linear program (LP)** is the problem of optimizing a **linear objective function** subject to **a set of linear constraints**.

$$\text{Maximize } Z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \leq b_i \text{ for } i = 1, \ldots, m$$

$$x_j \geq 0 \text{ for } j = 1, \ldots, n$$

- **Key Property**: The variables x_j are continuous (real-valued).
- **Solvability:** LPs are "easy". They can be solved (to optimality) in **polynomial time** (e.g., via the Ellipsoid or Interior-Point methods).

Formal Definition: Integer Programming

An **integer program (IP)** is a LP with an added constraint: all of the variables must be integers.

This one change makes the problem NP-hard (**why**?).

$$\text{Maximize } Z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \leq b_i \text{ for } i = 1, \ldots, m$$

$$x_j \in \mathbb{Z} \text{ and } x_j \geq 0 \text{ for } j = 1, \ldots, n$$

IP Terminology

IP (Integer Program): All decision variables are integers.

MILP (Mixed Integer Linear Program):

- Some decision variables are integers
- Others are continuous.

BIP (Binary Integer Program) or 0/1 IP:

- All variables are binary, i.e., $x_j \in \{0, 1\}$.
- This is the common form for combinatorial optimization.
- Why? Because $x_j = 1$ or $x_j = 0$ naturally represents a "yes/no" decision, e.g.,
  - Yes, we pick item j.
  - No, we don't pick set j.

The Modeling-and-Solving Paradigm

Using IP splits our work into two distinct phases: Modeling and Solving.

Modeling: The Art of Formulation

- Take a real-world problem (like TSP).
- Translate its variables, objective, and constraints into the mathematical language of IP.
- This is a human-driven, intellectual task.

Solving: The Science of Algorithms

- Feed this standard mathematical formulation into a powerful, general-purpose IP solver (e.g., Gurobi, CPLEX).
- The solver uses built-in algorithms (e.g., Branch & Bound) to find the optimal x_j values.
- This is a computer-driven, automated task.

Modeling: Basic Logical Constraints

Let $x_j \in \{0, 1\}$ be our "yes/no" decision variables. We can build complex logic by combining them with simple linear inequalities.

Scenario 1: Selecting from a Set

- "**At least one**" of $\{x_1, x_2, x_3\}$ must be chosen: $x_1 + x_2 + x_3 \geq 1$
- "**At most one**" of $\{x_1, x_2, x_3\}$ can be chosen: $x_1 + x_2 + x_3 \leq 1$
- "**Exactly one**" of $\{x_1, x_2, x_3\}$ must be chosen: $x_1 + x_2 + x_3 = 1$
- "**Exactly k**" from n items must be chosen: $\sum_{j=1}^{n} x_j = k$

Scenario 2: Implications (IF-THEN):

- "**If we pick x_1, we must also pick x_2**" : $x_1 \leq x_2$ or $x_1 - x_2 \leq 0$
- "**If we pick x_1, we cannot pick x_2**" : $x_1 + x_2 \leq 1$
- "**We must pick x_1 *if and only if* we pick x_2**" : $x_1 = x_2$

Modeling: The Big M Method

Scenario 3: Conditional Constraints

- This is used to **turn constraints "on" or "off" based on a binary variable**.
- Example: "If we don't build the warehouse ($x = 0$), then we can't ship any goods ($y = 0$). And If we do build ($x = 1$), we can ship up to 10,000 units ($y \leq 10000$)."
- Let M be a large, constant number (a "Big M"), larger than any possible value of y.
- **Constraint: $y \leq x \cdot M$**
  - If $x = 1$, $y \leq M$ (constraint is "off", y is allowed).
  - If $x = 0$, $y \leq 0$ (constraint is "on", y is not allowed).

Modeling: Handling Linearizing Products

Scenario 4: Our objective or constraint contains $x_i \cdot x_j$, both are binary. This is not linear.

- Trick: Introduce a new binary variable z_{ij} to replace the product $x_i \cdot x_j$. And Add 3 new constraints to force $z_{ij} = x_i \cdot x_j$:

  1) $z_{ij} \leq x_i$

  2) $z_{ij} \leq x_j$

  3) $z_{ij} \geq x_i + x_j - 1$

IP Formulation: 0/1 KP (1)

Problem Description:

- Given a set of n items, $N = \{1, 2, \dots, n\}$, each with a weight $w_i > 0$ and a value $v_i > 0$, and a knapsack with a total weight capacity W
- Goal: Select items to maximize the total value without exceeding the capacity

Variables:

- We need to decide which items to pick. Define a decision variable $x_j \in \{0, 1\}$ for each j. $x_j = 1$ if we select item j; otherwise $x_j = 0$.

Objective:

- We want to maximize the total value of selected items: **Maximize** $Z = \sum_{j=1}^{n} v_j x_j$

Constraints:

- The rule: The total weight of the selected items must not exceed W

IP Formulation: 0/1 KP (2)

$$\text{Maximize } Z = \sum_{j=1}^{n} v_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} w_j x_j \leq W \text{ for } j = 1, \ldots, n$$

$$x_j \in \{0,1\} \text{ for } j = 1, \ldots, n$$

IP Formulation: Set Cover with Cost (1)

Problem description:

- Given a universe of m elements, $U = \{e_1, \dots, e_m\}$ and a collection of n subsets, $S = \{S_1, \dots, S_n\}$, each S_j is associated with a cost c_j.
- Goal: Find a set cover, i.e., a sub-collection $C \subseteq S$ that covers all elements in U, with the minimum total cost.

Variables:

- We need to decide which sets to pick.
- Define a decision variable $x_j \in \{0, 1\}$ for each S_j. $x_j = 1$ if we select S_j; otherwise $x_j = 0$.

IP Formulation: Set Cover with Cost (2)

Objective:

- We want to minimize the total cost of selected sets: **Minimize** $Z = \sum_{j=1}^{n} c_j x_j$

Constraints:

- The rule: Every element $e_i \in U$ must be covered.
- For a specific element e_i, it is covered if we pick at least one set S_j that contains e_i.
- Let A be an $m \times n$ matrix where $a_{ij} = 1$ if $e_i \in S_j$, and 0 otherwise.
- $\sum_{j=1}^{n} a_{ij} x_j \geq 1$ for each element $e_i \in U$.

IP Formulation: Set Cover with Cost (3)

$$\text{Minimize } Z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t. } \sum_{j=1}^{n} a_{ij} x_j \geq 1 \text{ for each element } e_i \ (i = 1, \dots, m)$$

$$x_j \in \{0,1\} \text{ for each set } S_j \ (j = 1, \dots, n)$$

- Any feasible solution $\boldsymbol{x}$ for this BIP corresponds 1-to-1 with a valid set cover.
- The optimal solution $\boldsymbol{x}^*$ for this BIP is the optimal set cover.

IP Formulation: Traveling Salesman Problem (1)

Problem Description:

- Given n cities and costs c_{ij} between city i and city j.
- Goal: Find a min-cost tour that visits every city exactly once and returns to the start.

Variables:

- This is harder. A "node-based" variable $x_i = 1$ (visit city i) doesn't work, as we must visit all of them.
- The decision is not which cities to visit, but in which order.
- Let's define variables for the edges (the path).
- Let $x_{ij} \in \{0, 1\}$ for every city pair $i \neq j$.
  - **$x_{ij} = 1$ if the tour goes directly from city i to city j.**
  - **$x_{ij} = 0$ otherwise.**

IP Formulation: Traveling Salesman Problem (2)

Objective:

- We want to minimize the total cost of the path we take.
- **Minimize** $Z = \sum_{i=1}^{n} \sum_{j \neq i} c_{ij} x_{ij}$

The "seemingly easy" Constraints:

- What rules must a valid tour follow?
  - Rule 1: Must enter every city j exactly once: $\sum_{i \neq j} x_{ij} = 1$ for each j
  - Rule 2: Must leave every city i exactly once: $\sum_{j \neq i} x_{ij} = 1$ for each i

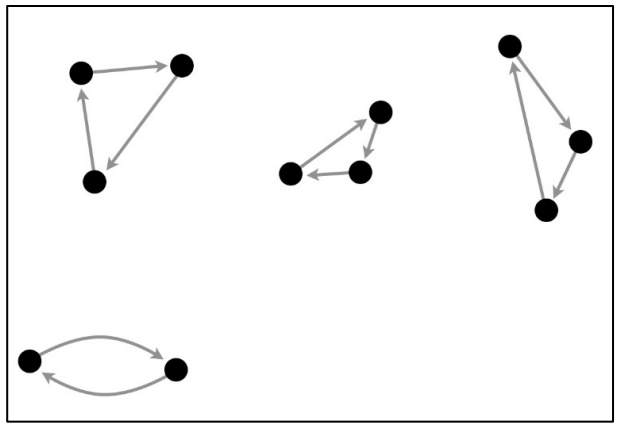
Are we done?

The Subtour Issue

The formulation on the previous slide is **incomplete**. It allows for solutions that satisfy the "enter/leave once" rules but are not a single tour.

Example: A "subtour"

- $1 \rightarrow 3 \rightarrow 5 \rightarrow 1$
- $2 \rightarrow 4 \rightarrow 2$
- ....



This solution is **feasible** under our current constraints, but it is **not** a truly valid TSP tour.

We need **Subtour Elimination Constraints (SECs)**.

Dantzig-Fulkerson-Johnson (DFJ, 1954) Formulation

Key Insight: For any proper subset (非空真子集) of cities $S \subseteq V$ (meaning the subset is not empty and is not V), the tour must leave that set at least once.

Subtour Elimination Constraints (SECs):

- Let S be any proper subset of V (e.g., $S = \{1, 2, 3\}$ in a 5-city TSP).
- In a valid TSP tour, the total number of edges x_{ij} for any $i \in S$ and any $j \notin S$ must be at least 1.
- $\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1$ for all $S \subseteq V, S \neq \emptyset, S \neq V$.

This single constraint (applied for all S) makes the subtours impossible.

- e.g., if $S = \{1, 2, 3\}$, $\sum_{i \in S} \sum_{j \notin S} x_{ij} = 0$ which violates the constraint.

The Full DFJ Formulation of TSP

$$\text{Minimize } Z = \sum_{i=1}^{n} \sum_{j \neq i} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{i \neq j} x_{ij} = 1 \quad \text{for each } j = 1, \ldots, n$$

$$\sum_{j \neq i} x_{ij} = 1 \quad \text{for each } i = 1, \ldots, n$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \quad \text{for all } S \subseteq V, S \neq \emptyset, S \neq V$$

$$x_{ij} \in \{0,1\} \text{ for all } i \neq j$$

DFJ Formulation of TSP is Exponentially Large

How many constraints are there in the DFJ formulation:

- There are $2^n - 2$ subset constraints

This is **exponentially large** number of constraints!

We can't just "write them all down" for the solver. That is to say, it is often computationally impossible to generate and load all the constraints into the solver (memory).

How to tackle this? We will touch this later in the next lecture.

A Compact Formulation of TSP

We model the tour as a path that delivers 1 unit of a "commodity" to every city.

Setup:

- Designate city 1 as the **Source**. City 1 "supplies" $n-1$ units of flow.
- Every other city $i \in \{2, \dots, n\}$ has a demand of 1 unit.

Decision Variables:

- $x_{ij} \in \{0, 1\}$ (same as before).
- $f_{ij} \geq 0$ and $f_{ij} \in \mathbb{Z}$, representing the amount of flow on edge (i, j).

How it works:

- Any subtour not containing node 1 has no source of flow.
- Its nodes (e.g., $4 -> 5 -> 4$) all have a demand of 1, but no supply.
- This violates flow conservation, so such subtours are impossible.

Flow-based Formulation of TSP

Flow Conservation Constraints (Subtour Elimination):

- Designate city 1 as the **Source**.
- City 1 "supplies" $n-1$ units of flow.
- Every other city $i \in \{2, \dots, n\}$ has a demand of 1 unit.
- $\sum_{k \neq i} f_{ki} - \sum_{j \neq i} f_{ij} = 1$ for each $i = 2, .., n$ (meaning: Flow In – Flow Out = 1 unit of Demand)
- $\sum_{j \neq 1} f_{1j} = n-1$ and $\sum_{j \neq 1} f_{j1} = 0$ (meaning: city 1 is the source)

Linking Constraints (Big M method):

- The flow f_{ij} can be non-zero only if $x_{ij} = 1$
- $f_{ij} \leq (n-1) \cdot x_{ij}$ for any $i \neq j$

Flow-based Formulation of TSP: The Full IP

$$\text{Minimize } Z = \sum_{i=1}^{n} \sum_{j \neq i} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum\nolimits_{i \neq j} x_{ij} = 1 \text{ for each } j = 1, \ldots, n$$

$$\sum\nolimits_{j \neq i} x_{ij} = 1 \text{ for each } i = 1, \ldots, n$$

$$\sum\nolimits_{k \neq i} f_{ki} - \sum\nolimits_{j \neq i} f_{ij} = 1 \text{ for each } i = 2, .., n$$

$$\textstyle\sum_{j \neq 1} f_{1j} = n - 1 \text{ and } \sum_{j \neq 1} f_{j1} = 0$$

$$f_{ij} \leq (n-1) \cdot x_{ij} \text{ for any } i \neq j$$

Flow-based Formulation: Analysis

Variable Count: $O(n^2)$ (number of variables x_{ij}) + $O(n^2)$ (number of variables f_{ij}).

Constraint Count: $O(n) + O(n) + O(n^2)$

This is a polynomial-sized (compact) formulation.

Flow-based formulation is also natural for modeling the vehicle routing problems (VRPs), though it may not be the most efficient one.

How Are IPs Solved?

Naive Idea: Brute-force

- We have n binary variables x_j.
- This gives 2^n possible solutions.
- We could check them all, discard the infeasible ones (that violate constraints), and find the best feasible one.
- This is just as bad as $O(2^n)$. Too slow!

Smarter Idea: Can we use the fact that LPs are easy to solve?

- What if we just ignore the integrality constraints, e.g., $x_j \in \{0, 1\}$?

Core Idea: LP Relaxation

This is the **one of the most important concept** for solving IPs.

We replace $x_j \in \{0,1\}$ with $0 \le x_j \le 1$ (or replace $x_j \in \{0,1,\dots,n\}$ with $0 \le x_j \le n$)

$$\text{Maximize } Z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \le b_i \text{ for } i = 1,\dots,m$$

$$x_j \in \{0,1\} \text{ for } j = 1,\dots,n$$

NP-Hard

$$\text{Maximize } Z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \le b_i \text{ for } i = 1,\dots,m$$

$$x_j \in [0,1] \text{ for } j = 1,\dots,n$$

Polynomial-time solvable.

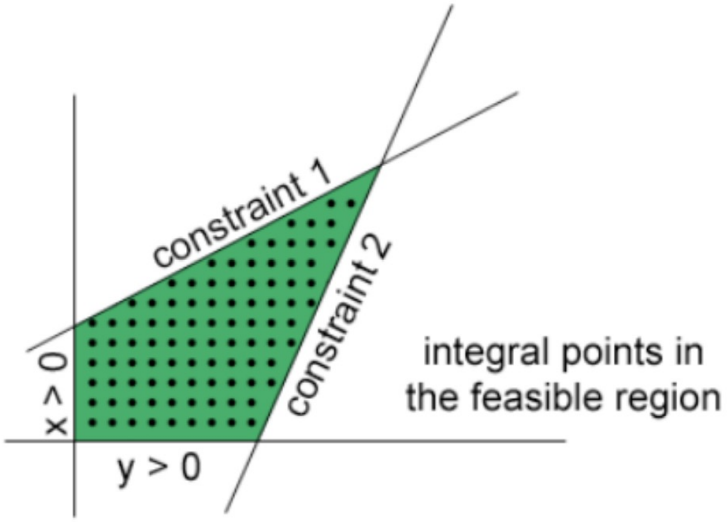
Geometric Intuition of LP-Relaxation

Let Z_{IP} be the optimal value of the IP.

Let Z_{LP} be the optimal value of the LP relaxation.

Fundamental Property:

- For Maximization problems: $Z_{LP} \geq Z_{IP}$
- For Minimization problems: $Z_{LP} \leq Z_{IP}$



Geometric Intuition:

- The constraints of LP define a "green feasible region" (a polytope)
- IP (Hard): We are searching for the **best integer point** (dots) inside this region.
- LP (Easy): We are searching for the **best point (of any kind)** in the green region.
  - The optimal LP solution will always be at a corner (vertex) of the polytope (多面体).

The LP relaxation provides an **optimistic bound** on the true integer solution.

Using the LP Relaxation (1)

When we solve the LP relaxation, two things can happen:

Case 1: The "Lucky" Case:

1) We solve the LP-relaxation (e.g., for 0/1 KP).

2) The optimal solution $\boldsymbol{x}^*$ comes back as all integers! (e.g., $x_1 = 1, x_2 = 0, x_3 = 1$)

3) $\boldsymbol{x}^*$ is a feasible solution for the original IP.

4) Because $Z_{LP} \geq Z_{IP}$, and we found an integer solution $\boldsymbol{x}^*$ with value Z_{LP}, we know Z_{IP} can't be any higher.

5) Conclusion: $Z_{LP} = Z_{IP}$. We have found the true optimal integer solution in polynomial time! This is called "the LP has an integral optimal solution".

Using the LP Relaxation (2)

Case 2: The "Normal" Case:

1) We solve the LP-relaxation (e.g., for 0/1 KP).

2) The optimal solution $\boldsymbol{x}^*$ comes back as fractional! (e.g., $x_1 = 1, x_2 = 0, x_3 = 0.5$)

3) What do we know?

- This is not a valid IP solution.
- The true integer optimum Z_{IP} is unknown, but it must be ≤ 150.

The Branch and Bound (B&B) Algorithm (1)

This is the algorithm that all modern IP solvers use.

It's essentially an intelligent "divide and conquer" search algorithm.

It has three main algorithmic components: **Branching, Bounding, and Pruning.**

Notations:

- Z_{IP}: the optimal value of the original IP (the one we want to solve).
- Z_{LP}: the optimal value of the LP relaxation of the original IP.
- Z_{INC}: the value of the incumbent solution (the best-so-far integer solution).
- Z_{LP}^{P}: the optimal value of the LP relaxation at node P in the B&B tree.

The Branch and Bound (B&B) Algorithm (2)

The "Branching" Idea:

- We solved the LP-relaxation of 0/1 KP and got a fractional value, e.g., $x_3 = 0.5$.
- We know that in the true optimal integer solution, x_3 is a non-negative integer.
- So, let's **branch** on this variable. We "divide" the problem into two new, smaller subproblems (child nodes):
  - Subproblem 1: The original LP + a new constraint: $x_3 \leq 0$ ($x_3 = 0$).
  - Subproblem 2: The original LP + a new constraint: $x_3 \geq 1$.
  - The true IP solution must be in one of these two subproblems.
- We **recursively** branch on each subproblem with fractional value: Each child node is defined by **the constraints of its parent node plus an additional, tighter linear constraint, which consequently reduces the feasible space.**

The Branch and Bound (B&B) Algorithm (3)

The "Bounding" Idea:

- This is the "intelligent" part of the algorithm.
- The **"Incumbent" (The Best-So-Far)**:
  - Let's say, in our search tree, we find our **first** valid integer solution (e.g., $x_1 = 1, x_3 = 0, \ldots$) by solving some LP-relaxation subproblem.
  - Suppose its value is $Z_{INC} = 130$ (for a maximization problem).
  - This Z_{INC} is now our "Incumbent" – the best integer solution found so far. We know the true optimum Z_{IP} must be $\geq Z_{INC} = 130$
- The "**Bound**" (**The Optimistic Estimate**):
  - Now, we go to another un-explored node P in the tree.
  - We solve its LP Relaxation and get its optimal value Z_{LP}^{P}.
  - Let's assume $Z_{LP}^{P} = 125$.

The Branch and Bound (B&B) Algorithm (4)

The "Pruning" Idea:

- This $Z_{LP}^{P} = 125$ is an optimistic estimate (bound). It means **no integer solution** in this entire branch (i.e., any subproblem created from this node) can ever have a value greater than 125.
- Our current incumbent ($Z_{INC} = 130$) is already better than the best possible outcome of this branch (125).
- **Therefore, we "prune" (cut off) this entire branch.** We can discard it without any further branching, saving exponential work.

Algorithmic Steps of B&B for Maximization

1) Initialize a "global incumbent" $Z_{INC} = -\infty$.

2) Add the root LP relaxation to a "**set**" of problems.

3) While **set** is not empty:

a. Select a problem P from the set.

b. Solve its LP relaxation, get Z_{LP}^{P} and $\boldsymbol{x}^{*}$.

c. Pruning 1 (by Bound): If $Z_{LP}^{P} \leq Z_{INC}$, prune this branch, and goto step 3).

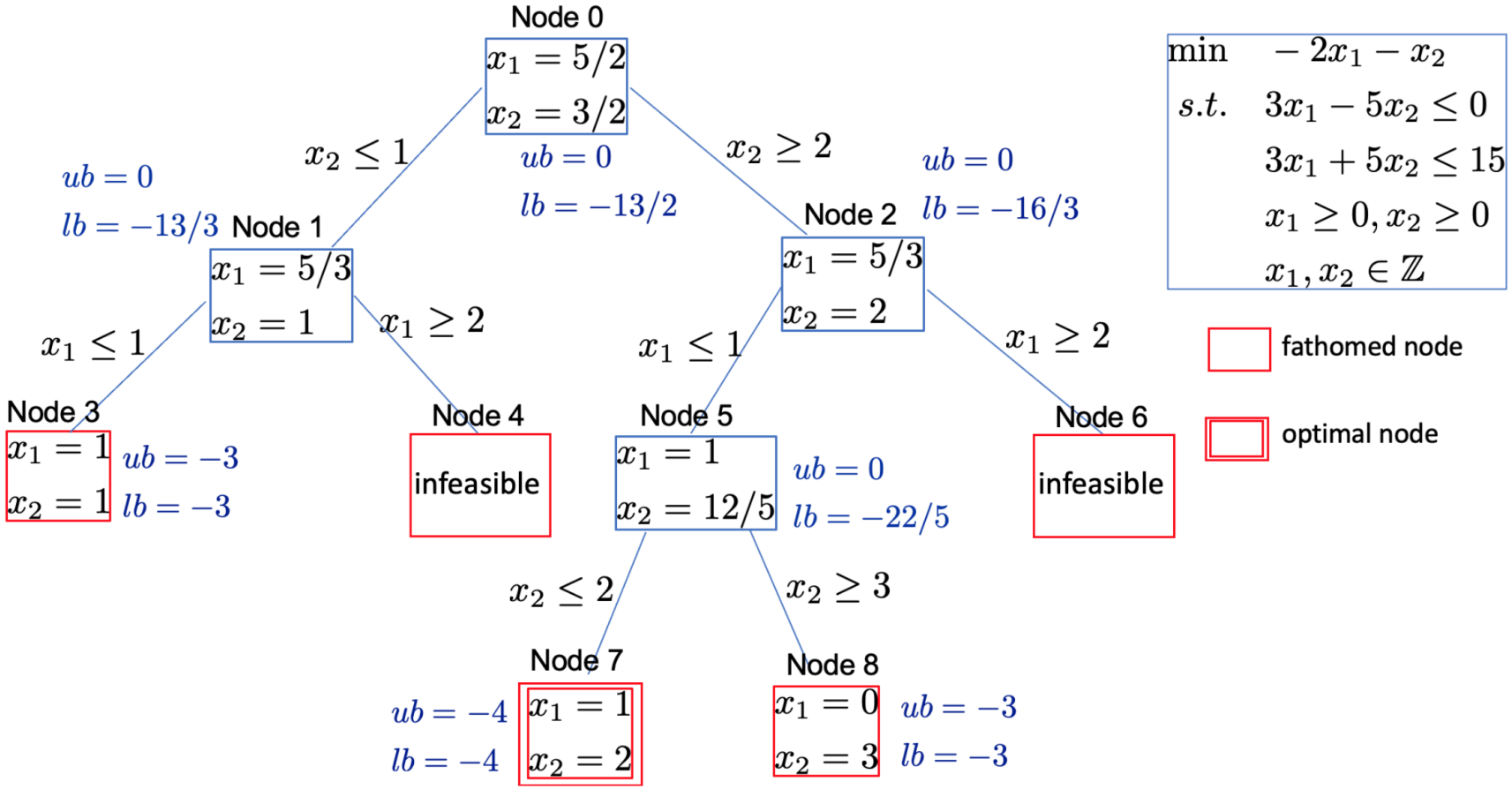
d. Pruning 2 (Infeasible): If $Z_{LP}^{P} = -\infty$, prune this branch and goto step 3).

e. If $\boldsymbol{x}^{*}$ are all integers, we have found a feasible integer solution. Check if $Z_{LP}^{P} > Z_{INC}$ and update the incumbent accordingly. Goto Step 3).

f. Branching: If $\boldsymbol{x}^{*}$ is fractional (e.g., $x_j =$ fractional value a). Create two new subproblems with added constraints: $x_j \leq \lfloor a \rfloor$ and $x_j \geq \lceil a \rceil$, respectively. Add both to the set.

4) Return Z_{INC}.

An Example for Minimization



- ub: upper bound (global incumbent)
- lb: lower bound (optimistic estimate)
- fathomed node: 已被充分探索、解决或永久排除的node