Design and Analysis of Algorithms

Branch-and-Bound Algorithm

Si Wu

School of CSE, SCUT

cswusi@scut.edu.cn

TA: 1684350406@qq.com

Topics

- **Complete Enumeration**
- **Branch-and-Bound Algorithm**

Is Integer Programming (IP) easy to solve?

- At the first glance, IP may be easy to solve since the number of feasible solution is smaller and is limited.

- If the feasible region is bounded, the number of feasible solution is finite. But, the number is simply too large. With *n* variables (*0/1*), there are 2^n solutions to be considered.

- The corner point is (generally) no longer feasible

Overview

- Enumerating all solutions is too slow for most problems.
- Branch-and-bound starts the same as enumerating, but it cuts out a lot of the enumeration whenever possible.
- Branch-and-bound is the starting point for all solution techniques for integer programming.

Capital Budgeting Example

Investment budget = $14,000

Investment	1	2	3	4	5
Cash Required	$5,000	$4,000	$7,000	$3,000	$6,000
Present Value	$12,000	$11,000	$13,000	$8,000	$15,000

Capital Budgeting Example

Investment budget = $14,000

Investment	1	2	3	4	5
Cash Required	$5,000	$4,000	$7,000	$3,000	$6,000
Present Value	$12,000	$11,000	$13,000	$8,000	$15,000

$$\begin{aligned} \max \quad & 12x_1 + 11x_2 + 13x_3 + 8x_4 + 15x_5 \\ \text{s.t.} \quad & 5x_1 + 4x_2 + 7x_3 + 3x_4 + 6x_5 \le 14 \\ & x_j \in \{0, 1\} \text{ for each } j = 1 \text{ to } 5 \end{aligned}$$

Complete Enumeration

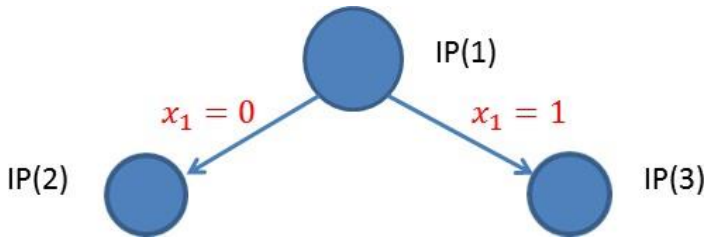
- Systematically considers all possible values of the decision variables.
  If there are n binary variables, there are 2^n different ways.
- Usual idea: iteratively break the problem into two.
  At the first iteration, we consider separately the case that $x_1 = 0$ and $x_1 = 1$.
- Each node of the tree represents the original problem plus additional constraints.

An Enumeration Tree

IP(1)

$$\begin{aligned} \max \quad & 12x_1 + 11x_2 + 13x_3 + 8x_4 + 15x_5 \\ \text{s.t.} \quad & 5x_1 + 4x_2 + 7x_3 + 3x_4 + 6x_5 \le 14 \\ & x_j \in \{0, 1\} \text{ for each } j = 1 \text{ to } 5 \end{aligned}$$

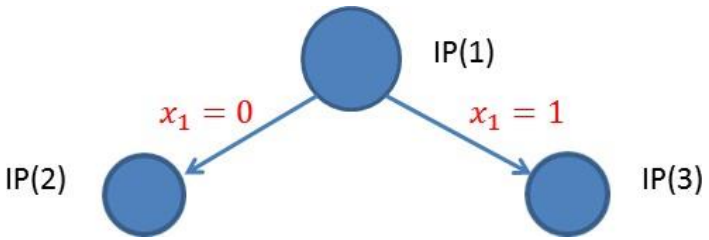
An Enumeration Tree



$$\begin{aligned} \max \quad & 12x_1 + 11x_2 + 13x_3 + 8x_4 + 15x_5 \\ \text{s.t.} \quad & 5x_1 + 4x_2 + 7x_3 + 3x_4 + 6x_5 \le 14 \\ & x_j \in \{0, 1\} \text{ for each } j = 1 \text{ to } 5 \end{aligned}$$

- We refer to node 2 and 3 as the children of node 1 in the enumeration tree.
- We refer to node 1 as the parent of node 2 and 3.

An Enumeration Tree

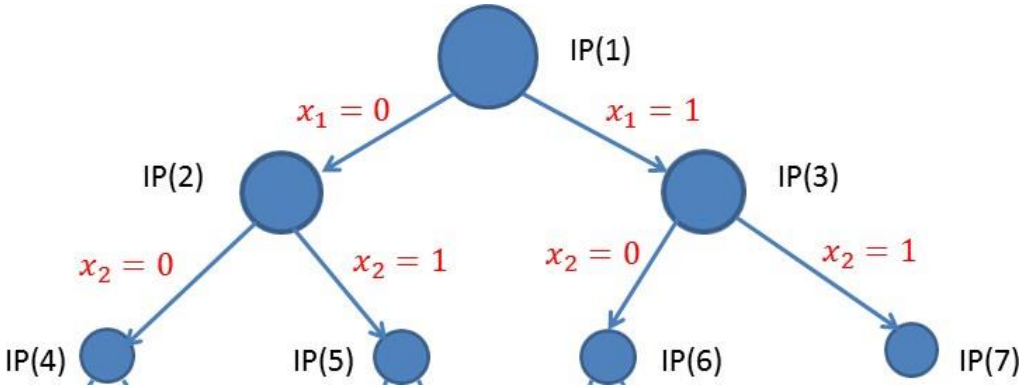


$$\begin{aligned} \max \quad & 12x_1 + 11x_2 + 13x_3 + 8x_4 + 15x_5 \\ \text{s.t.} \quad & 5x_1 + 4x_2 + 7x_3 + 3x_4 + 6x_5 \le 14 \\ & x_j \in \{0, 1\} \text{ for each } j = 1 \text{ to } 5 \end{aligned}$$

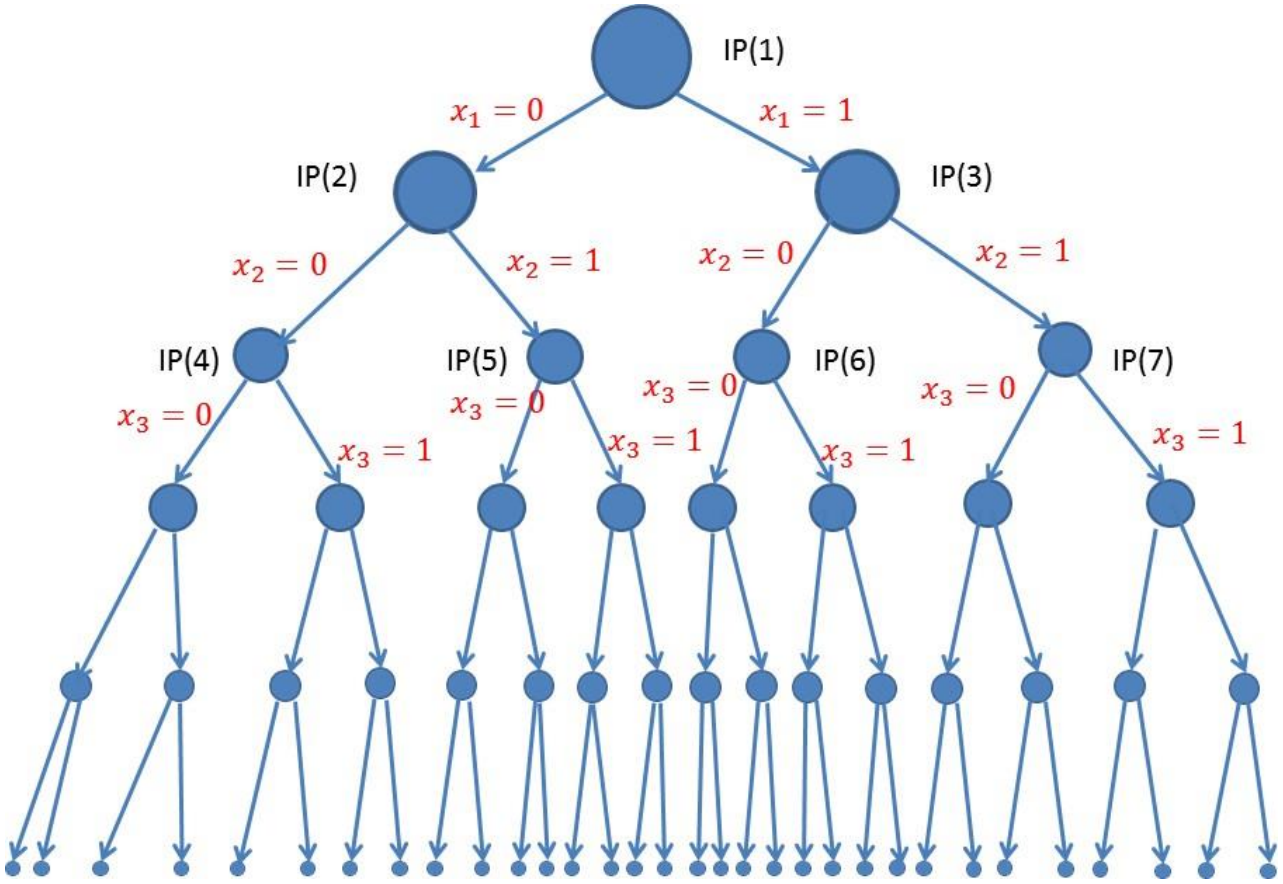
Which of the following is false?

- IP(1) is the original integer program.
- IP(3) is obtained from IP(1) by adding the constraint $x_1 = 1$.
- It is possible that there is some solution that is feasible for both IP(2) and IP(3).

An Enumeration Tree



An Enumeration Tree



Number of leaves of the tree : 32.

⇒ *If there are n variables, the number of leaves is 2^n.*

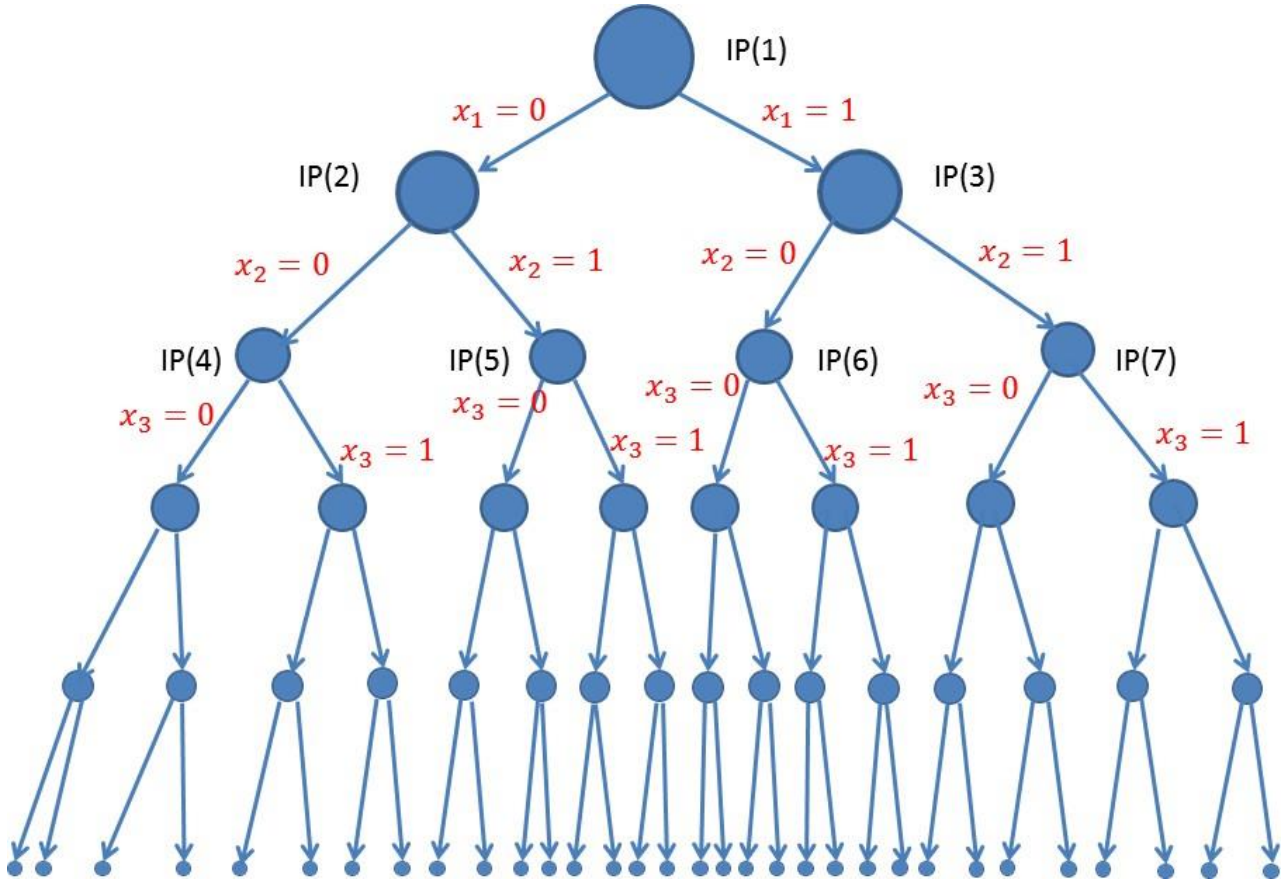
On Complete Enumeration

- Suppose that we could evaluate 1 billion solutions per second.
- Let n = number of binary variables.
- Solution times
  - $n = 30$, 1 second
  - $n = 40$, 17 minutes
  - $n = 50$, 11.6 days
  - $n = 60$, 31 years
  - $n = 70$, 31,000 years

How to solve large size integer program faster?

Only a tiny fraction of the feasible solutions actually need to be examined.

Subtrees of An Enumeration Tree



If we can eliminate an entire subtree in one step, we can eliminate a fraction of all complete solutions at a single step.

Branch-and-Bound Algorithm

- Branch-and-bound algorithm are the most popular methods for solving integer programming problems.
- **Basic Idea:** Enumeration procedure can always find the optimal solution for any bounded IP problem, but it takes too much time. So, we consider the partial enumeration, that is divide-and-conquer.
- They enumerate the entire solution space but only implicitly,called *implicit enumeration algorithms.*
- *Bounding*, *branching*, and *fathoming* are the three components of Branch-and-bound technique.
- Running time grows exponentially with the problem size, but *small-to-moderate* size problems can be solved in reasonable time.

A Simpler Problem to Work with

$$
\begin{aligned}
\text{Max} \quad & 24x_1 + 2x_2 + 20x_3 + 4x_4 \\
\text{s.t.} \quad & 8x_1 + x_2 + 5x_3 + 4x_4 \leq 9 \\
& x_i \in \{0, 1\} \text{ for } i = 1 \text{ to } 4
\end{aligned}
\qquad \text{IP(1)}
$$

Linear Programming (LP) Relaxation

LP Relaxation: The LP obtained by omitting all integer or 0-1 constraints on variables is called the LP relaxation of IP.

IP:

$$\begin{aligned} \text{max (or min)} \quad & cx \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \text{ and integers} \end{aligned}$$

LP Relaxation:

$$\begin{aligned} \text{max (or min)} \quad & cx \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

IP and LP Relaxation

Since LP relaxation is less constrained than IP, the following are immediate:

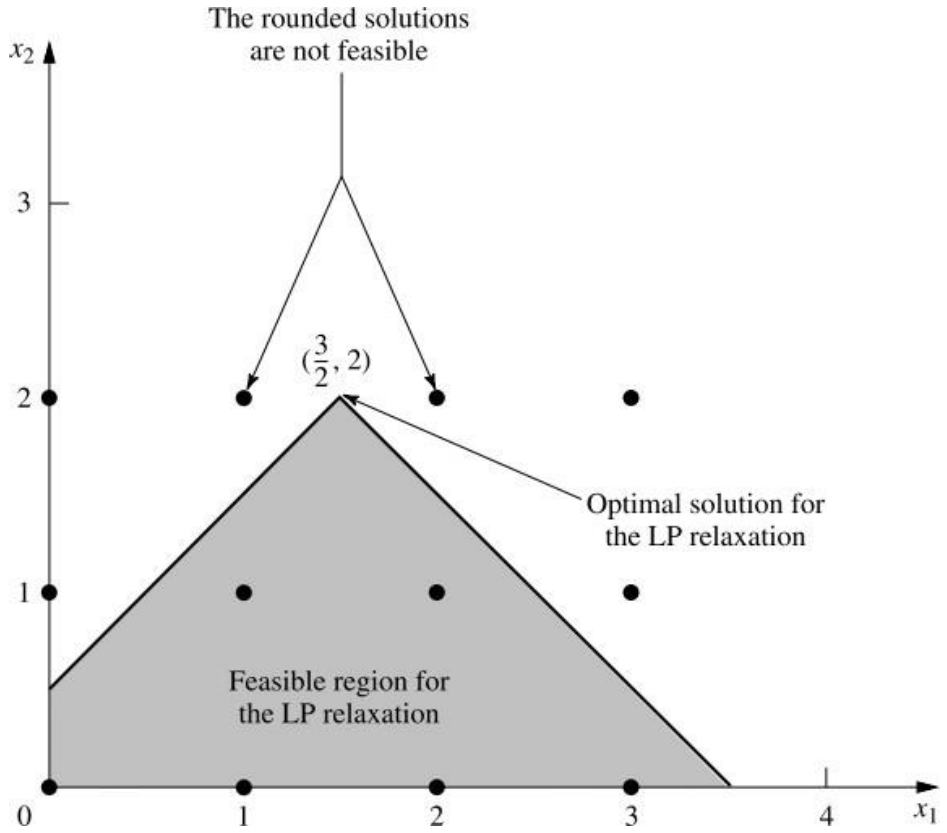
- If IP is a minimization/maximization problem, the optimal objective value for LP relaxation is less/greater than or equal to the optimal objective for IP.
- If LP relaxation is infeasible, then so is IP.
- If LP relaxation is optimized by integer variables, then that the solution is feasible and optimal for IP.

So, solving LP relaxation does give some information:

It gives a bound on the optimal value, and if we are lucky, may give the optimal solution to IP.

Why can't we solve the LP relaxation and round it up/down to get the required integer solution?

- Rounding does not guarantee the feasibility.



Max $z = x_2$

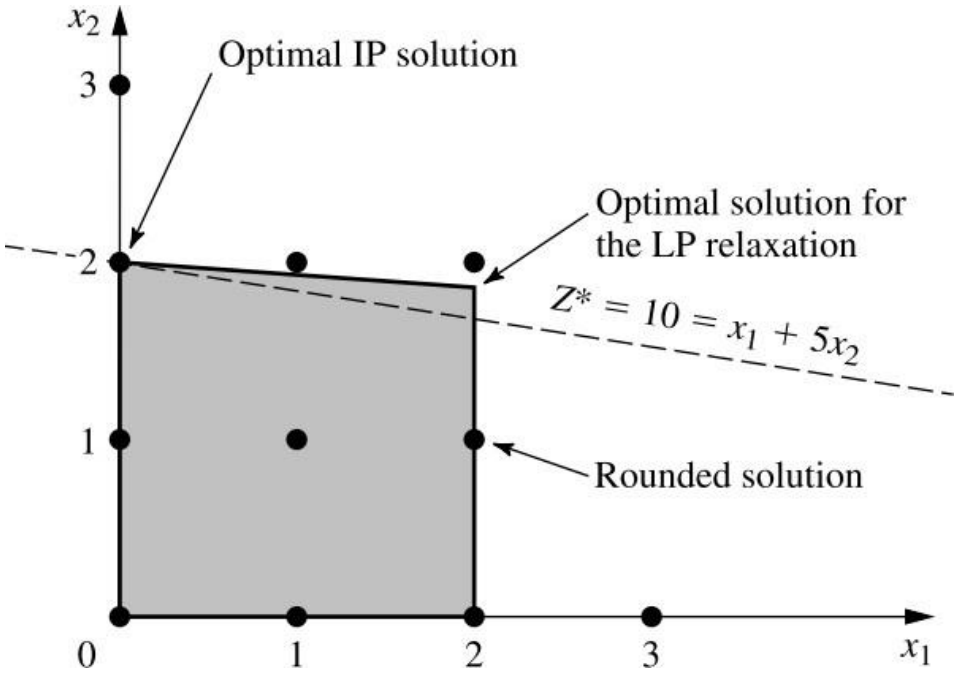
s.t. $-x_1 + x_2 \leq 0.5$

$x_1 + x_2 \leq 3.5$

$x_1, x_2 \geq 0$ and are integers

Why can't we solve the LP relaxation and round it up/down to get the required integer solution?

- Rounding does not guarantee the optimality.



Max $z = x_1 + 5x_2$

s.t. $x_1 + 10x_2 \leq 20$

$x_1 \leq 2$

$x_1, x_2 \geq 0$ and are integers

Branch-and-Bound

The essential idea: search the enumeration tree, but at each node:

- Solve the LP relaxation at the node
- Eliminate the subtree (fathom it) if
  - The solution is integer (there is no need to go further) or
  - There is no feasible solution or
  - The best solution in the subtree cannot be as good as the best available solution (the incumbent).

Bounding

- For each (sub)problem, we need to obtain a bound on how good its best feasible solution can be.
- Usually, the bound is obtained by solving the LP relaxation. LP relaxation of IP(1):

$$
\begin{aligned}
\text{Max} \quad & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\
\text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\
& 0 \leq x_i \leq 1 \text{ for } i = 1 \text{ to } 4
\end{aligned}
\qquad \text{LP(1)}
$$

Bounding

- For each (sub)problem, we need to obtain a bound on how good its best feasible solution can be.
- Usually, the bound is obtained by solving the LP relaxation. LP relaxation of IP(1):

$$\begin{aligned} \text{Max} \quad & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & 0 \leq x_i \leq 1 \text{ for } i = 1 \text{ to } 4 \end{aligned} \qquad \text{LP(1)}$$

- The LP relaxation of the knapsack problem can be solved using a "greedy algorithm".

  Think of the objective in terms of dollars, and consider the constraint as bound on the weight.

Solving the LP relaxation (LP(1))

$$
\begin{aligned}
\text{Max} \quad & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\
\text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 \le 14 \\
& 0 \le x_i \le 1 \text{ for } i = 1 \text{ to } 4
\end{aligned}
\qquad \text{LP(1)}
$$

item	1	2	3	4
unit value	$1.6	$1.571	$1.5	$1.333

Consider the unit value of the four items. Put items into the knapsack in decreasing order of unit value. What do you get?

Solving the LP relaxation (LP(1))

$$
\begin{aligned}
\text{Max} \quad & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\
\text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 \le 14 \\
& 0 \le x_i \le 1 \text{ for } i = 1 \text{ to } 4
\end{aligned}
\qquad \text{LP(1)}
$$

item	1	2	3	4
unit value	\$1.6	\$1.571	\$1.5	\$1.333

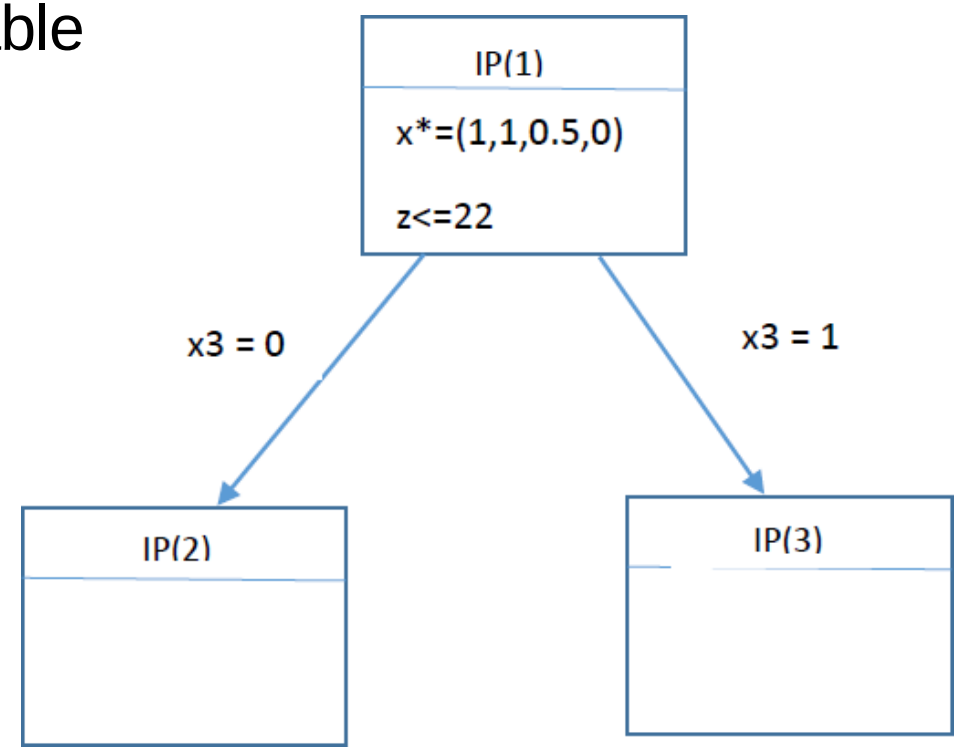
Consider the unit value of the four items. Put items into the knapsack in decreasing order of unit value. What do you get?

$\Rightarrow (x_1, x_2, x_3, x_4) = (1, 1, 0.5, 0)$, with $z = 22$

$\Rightarrow$ No integer solution will have value lager than 22.

Branching

- Partitioning the entire set of feasible solutions into smaller and smaller subsets.
  - Set "*No incumbent* with objective value $z^* = -\infty$".
  x_3 - branching variable



Note that any optimal solution to the overall problem must be feasible to one of the subproblems.

More on the Incumbent

- The incumbent is the feasible solution for the IP. It is the best solution so far in the B&B search.
- As Branch-and-bound proceeds, new solutions will be evaluated. If a new solution is better than the current incumbent, it replaces the current incumbent.
- So, the incumbent is always the best solution seen so far.

Solving the LP relaxation (LP(2))

IP(2): $x_3 = 0$

$$
\begin{array}{lll}
\text{Max} & 8x_1 + 11x_2 + 6 \times 0 + 4x_4 & \\
\text{s.t.} & 5x_1 + 7x_2 + 4 \times 0 + 3x_4 \leq 14 & \text{LP(2)} \\
 & 0 \leq x_i \leq 1 \text{ for } i = 1, 2, 4 &
\end{array}
$$

item	1	2	3	4
unit value	\$1.6	\$1.571	\$0	\$1.333

$\Rightarrow (x_1, x_2, x_3, x_4) = (1, 1, 0, \frac{2}{3})$, with $z = 21\frac{2}{3}$

$\Rightarrow$ No integer solution for this subproblem will have value lager than 21, but we don't have any feasible integer solution.

Solving the LP relaxation (LP(3))

IP(3): $x_3 = 1$

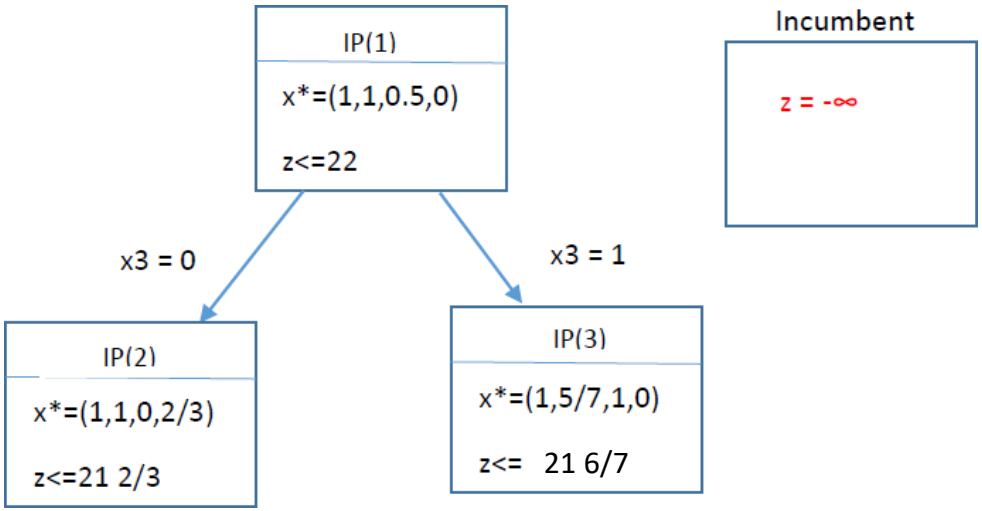
$$\begin{aligned} \text{Max} \quad & 8x_1 + 11x_2 + 6 \times 1 + 4x_4 \\ \text{s.t.} \quad & 5x_1 + 7x_2 + 4 \times 1 + 3x_4 \leq 14 \\ & 0 \leq x_i \leq 1 \text{ for } i = 1, 2, 4 \end{aligned} \qquad \text{LP(3)}$$

item	1	2	4
unit value	\$1.6	\$1.571	\$1.333

$\Rightarrow (x_1, x_2, x_3, x_4) = (1, \frac{5}{7}, 1, 0)$, with $z = 21\frac{6}{7}$

$\Rightarrow$ No integer solution for this subproblem will have value lager than 21, but we don't have any feasible integer solution.

Enumeration Tree

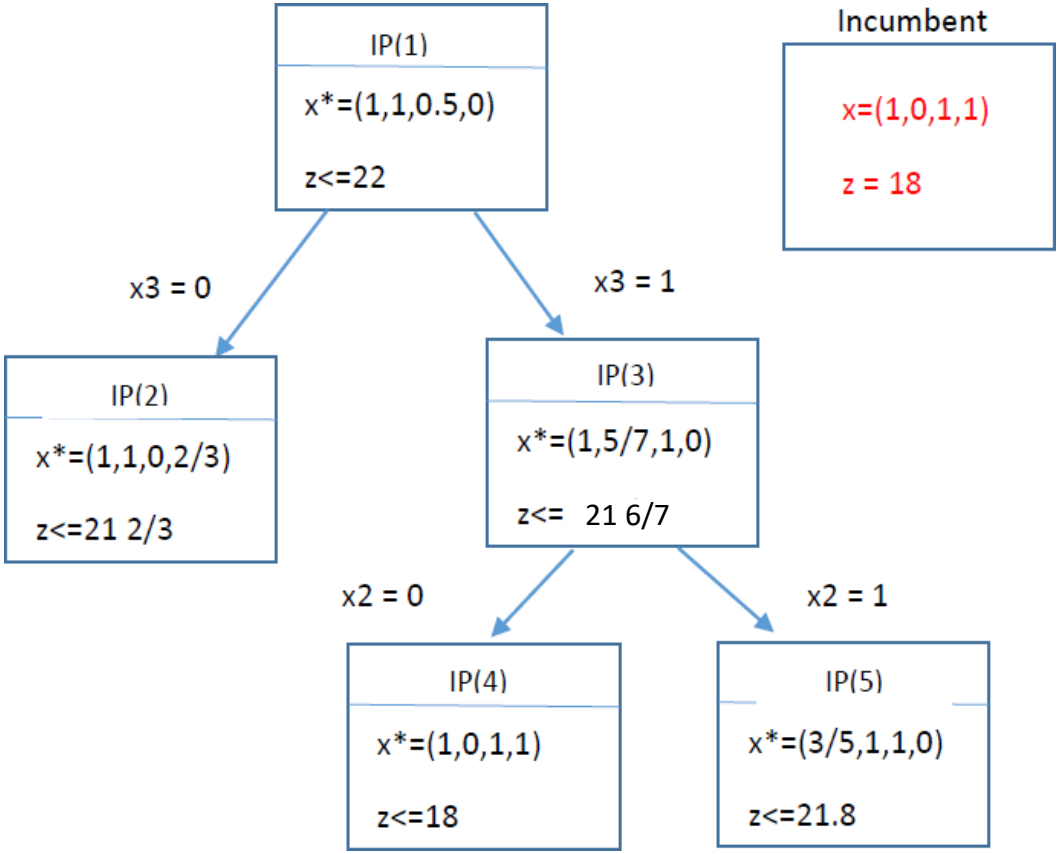


We do not have any feasible integer solution. So, we will take a subproblem and branch on one of its variables.
In general, we will choose the subproblem as follows:

- choose an active subproblem, which so far only means we have not chosen before, and
- choose the subproblem with the highest solution value (for maximization) (lowest for minimization).

⇒ Choose IP(3) and branch on x_2.

Enumeration Tree



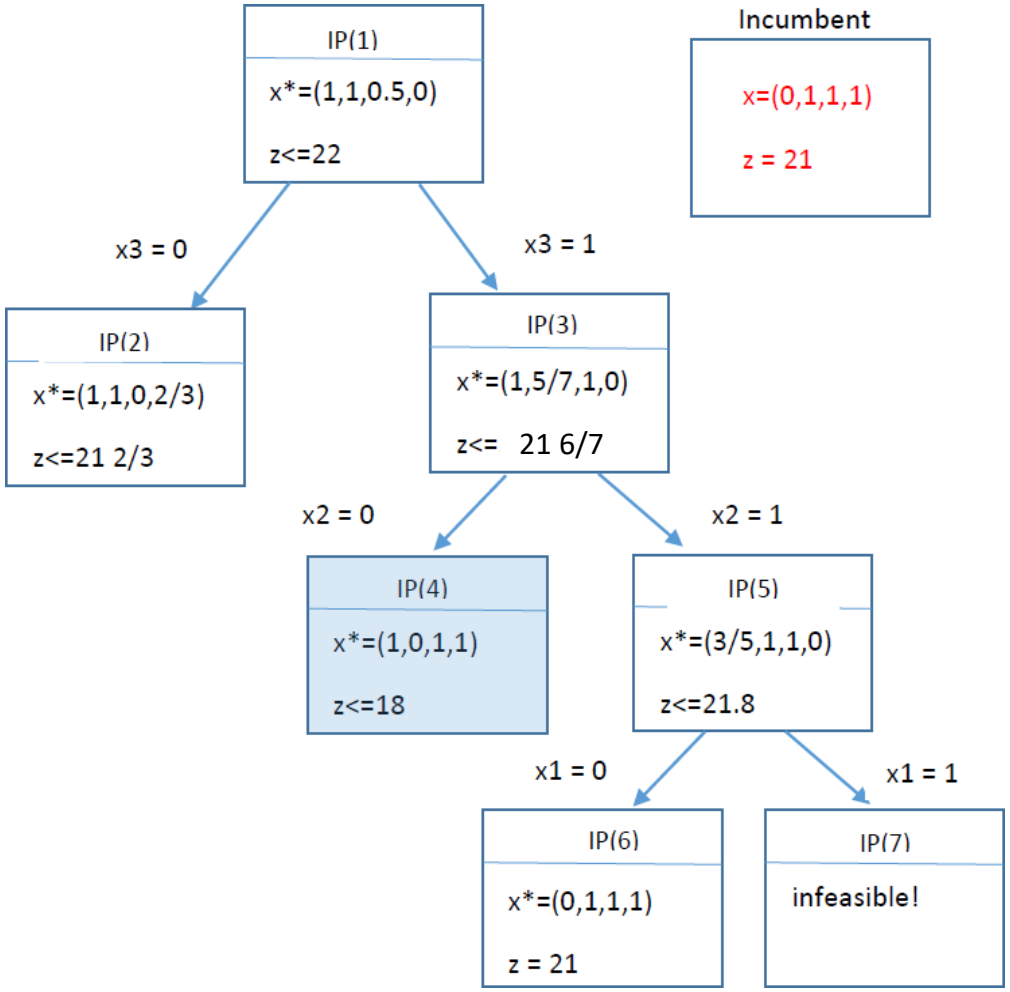
IP(4): Feasible integer solution with value 18.

⇒ *No further branching on subproblem 4 is needed.*

⇒ *Fathoming case 1*: The optimal solution for its LP relaxation is integer.

⇒ If this solution is better than the incumbent, it becomes the new incumbent.

Enumeration Tree

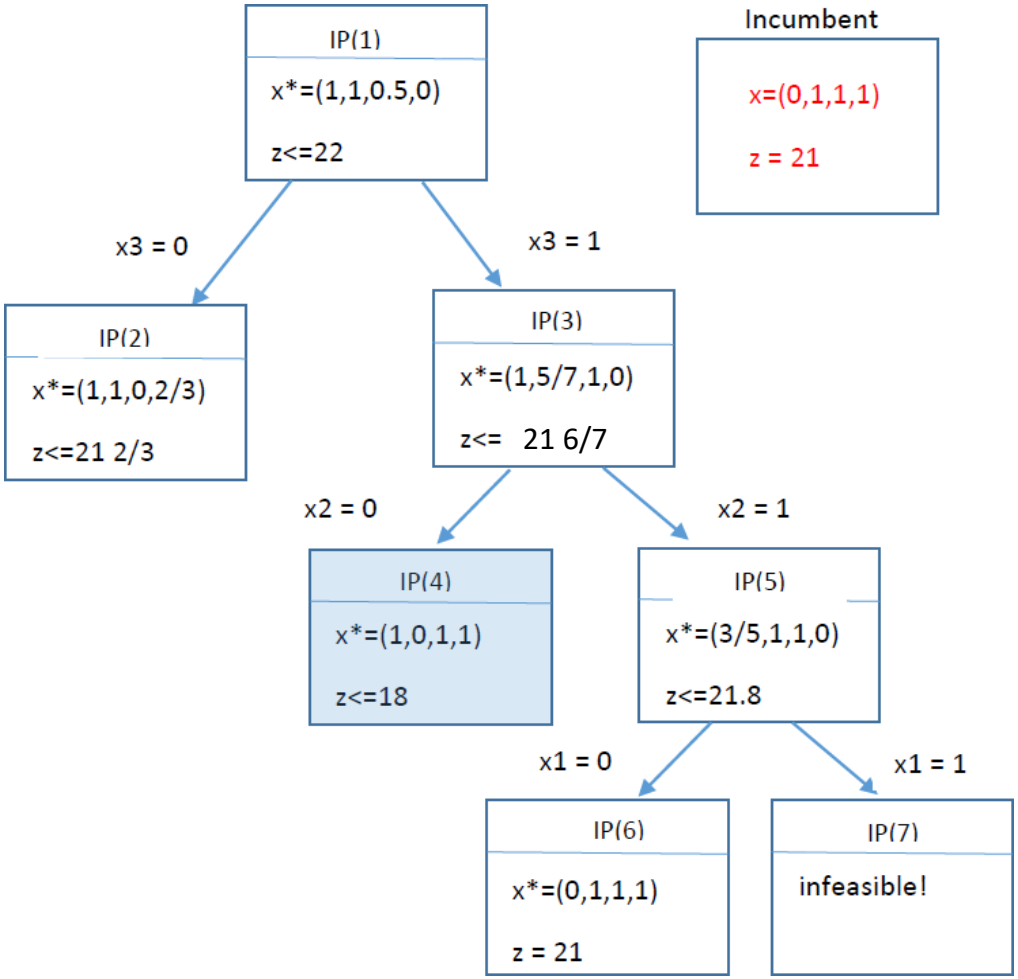


IP(6): Feasible integer solution with value 21 (>18).

⇒ No further branching on subproblem 6 is needed. (Fathoming case 1).

⇒ Update the incumbent and its value.

Enumeration Tree

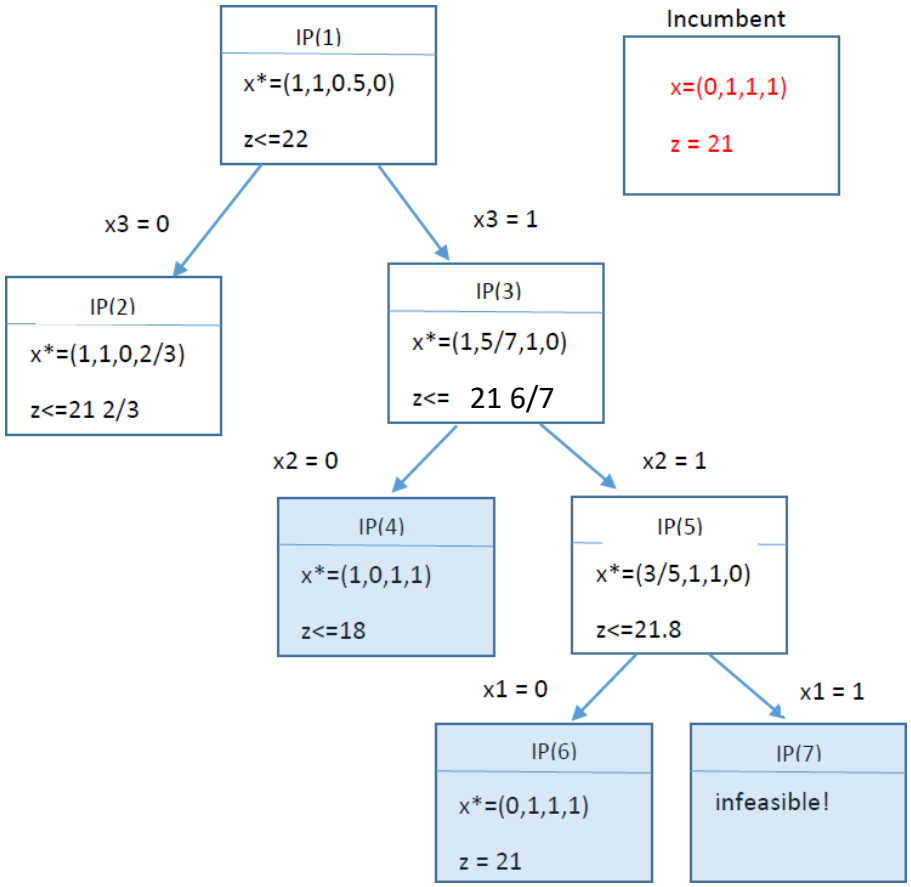


IP(7): Infeasible solution

⇒ No further branching on subproblem 7 is needed.

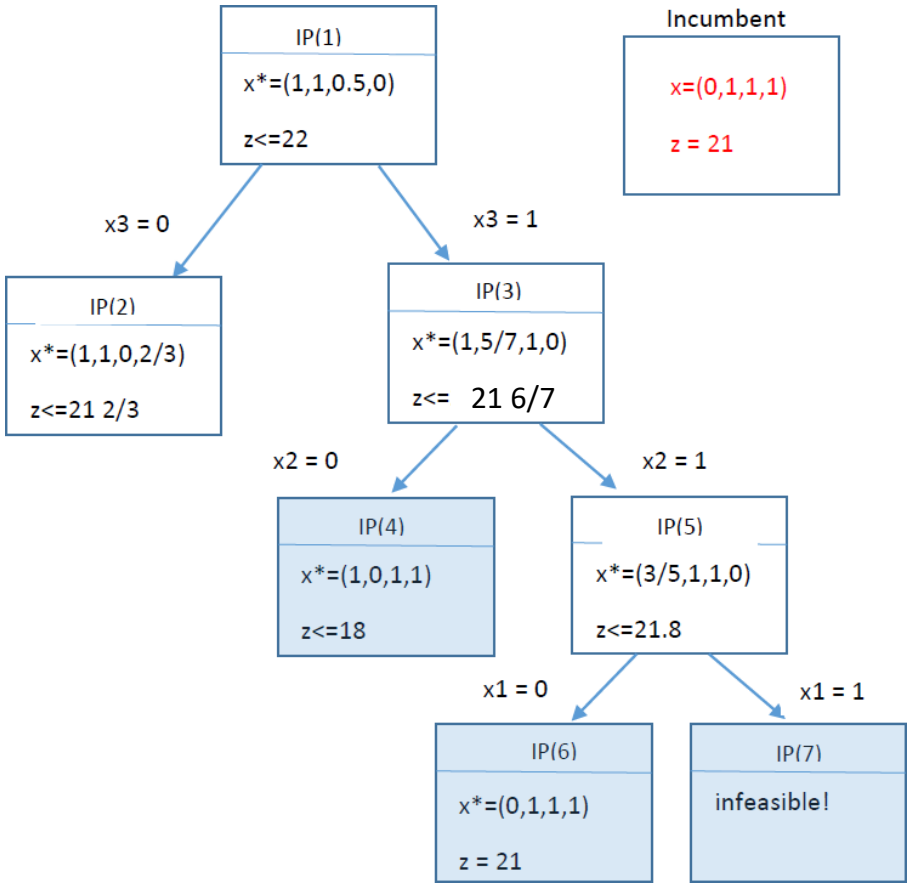
⇒ *Fathoming case 2*: The LP relaxation has no feasible solutions.

Enumeration Tree



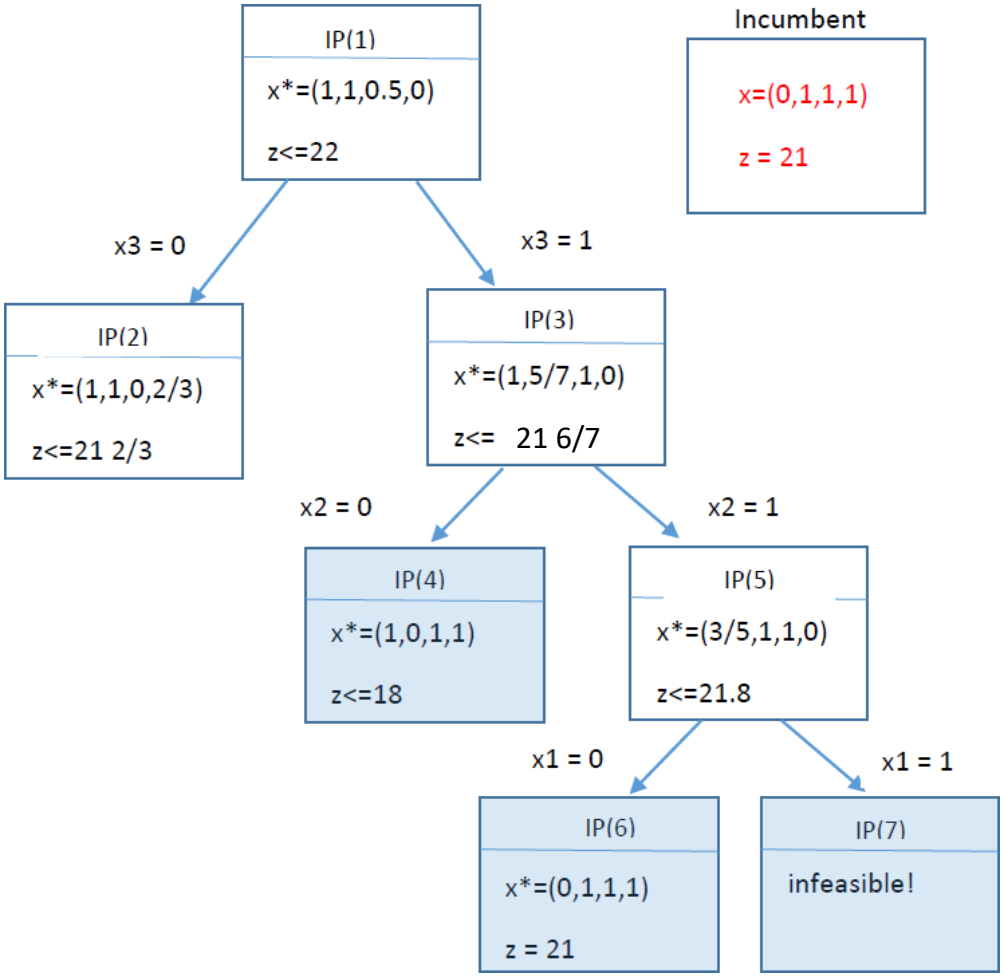
Only one active subproblem left is subproblem 2 with its bound = $21\frac{2}{3}$.

Enumeration Tree



Only one active subproblem left is subproblem 2 with its bound = $21\frac{2}{3}$. Since the optimal objective value of the original problem is integer, if the best value solution for a node is at most $21\frac{2}{3}$, then we know the best bound is *at most 21*.

Enumeration Tree

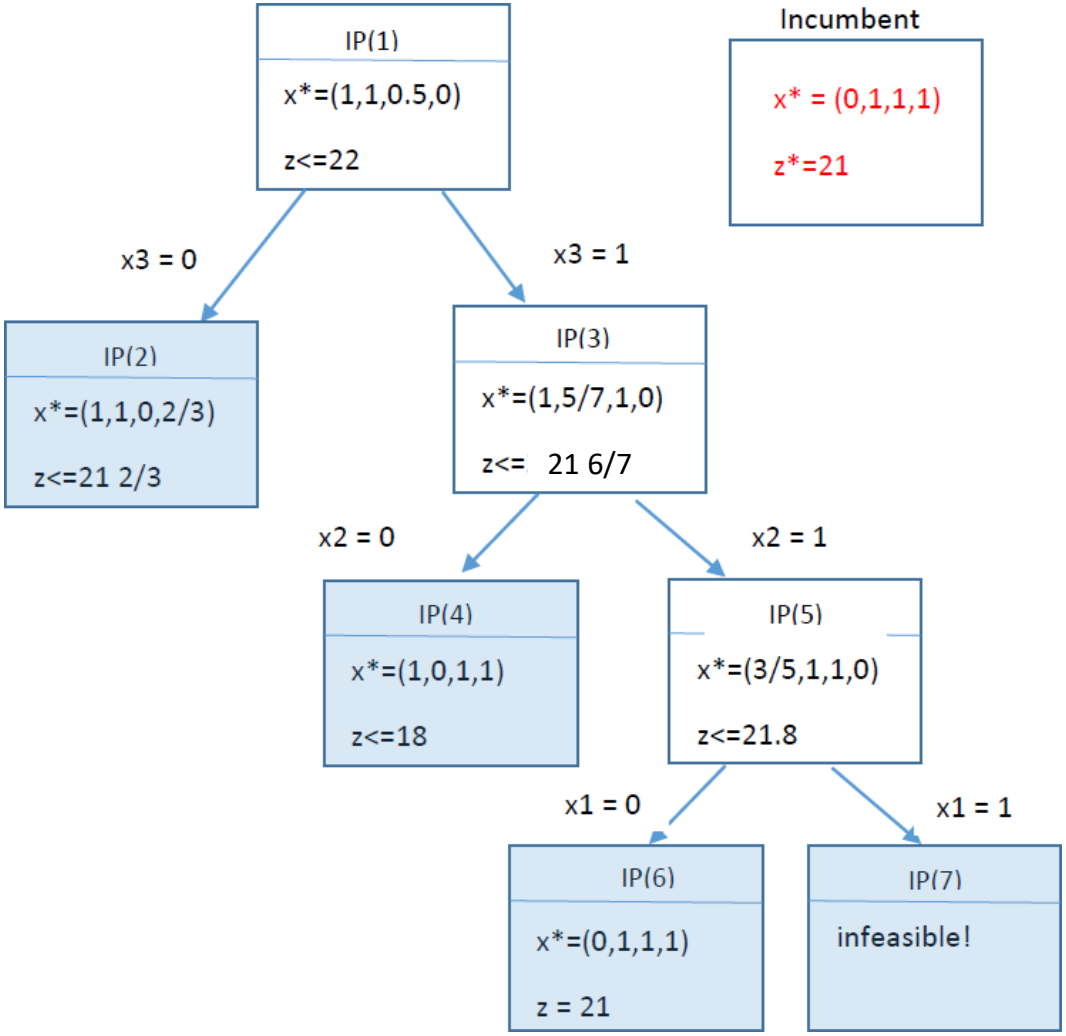


Only one active subproblem left is subproblem 2 with its bound = 21.

⇒ We fathom this subproblem.

⇒ *Fathoming case 3*: Subproblem's bound ≤ z.

Enumeration Tree



There are no longer any active subproblems, so the optimal solution value is 21.

Branch-and-Bound Algorithm

- **Branch-and-bound strategy:**
  - Solve the linear relaxation of the problem.
    - If the solution is integer, then we are done.
    - Otherwise, create two new subproblems by branching on a fractional variable.
  - A node (subproblem) is not active when any of the following occurs:
    - The node is being branched on;
    - The solution is integral;
    - The subproblem is infeasible;
    - You can fathom the subproblem by a bounding argument.
  - Choose an active node and branch on a fractional variable. Repeat until there are no active subproblems.

A branch-and-bound algorithm for mixed integer programming

$$\text{Max} \quad z = \sum_{j=1}^{n} c_j x_j$$

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

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

$$x_j \text{ is integer, for } j = 1, \ldots, I \quad (I \le n)$$

Modifications:

- branching variables: only variables considered are the integer-restricted variables that have a non-integer value in the optimal solution for the LP relaxation of the current subproblem.
- values assigned to the branching variable for creating the new smaller subproblems:

$$x_j \le \lfloor x_j^* \rfloor \text{ and } x_j \ge \lceil x_j^* \rceil$$

Example

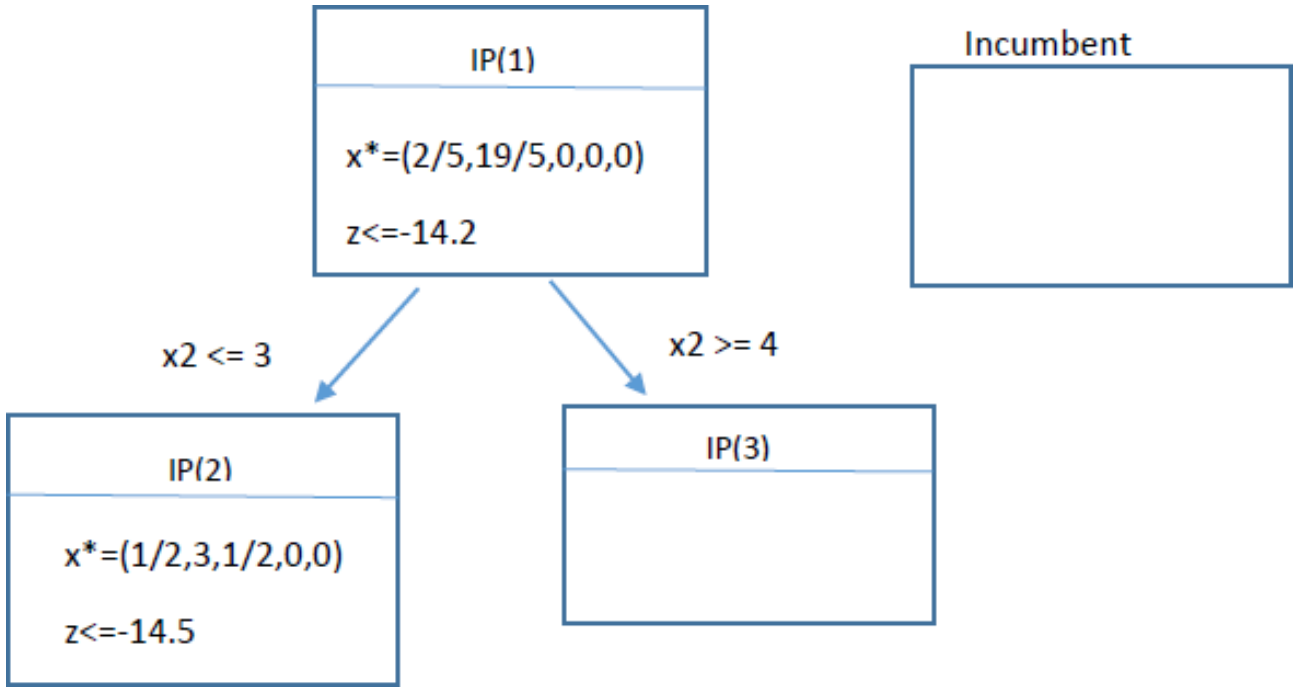
$$\max \quad z = -7x_1 - 3x_2 - 4x_3$$

$$\text{s.t.} \quad \begin{cases} x_1 + 2x_2 + 3x_3 - x_4 = 8, \\ 3x_1 + x_2 + x_3 - x_5 = 5, \\ x_1, \quad x_2, \quad \cdots, \quad x_5 \geq 0 \end{cases}$$ and integer.

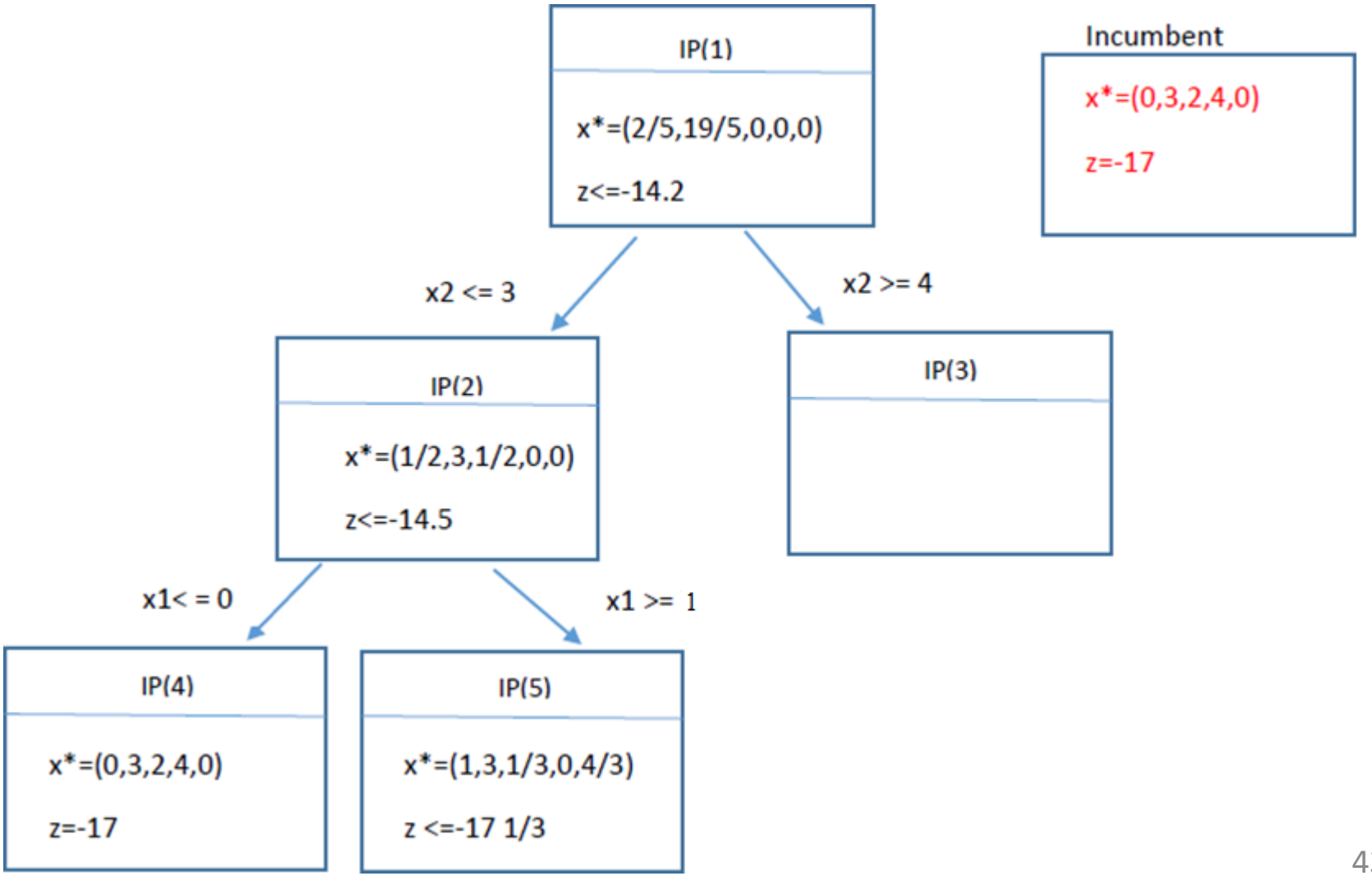
IP(1)
x*=(2/5,19/5,0,0,0)
z<=-71/5

Incumbent

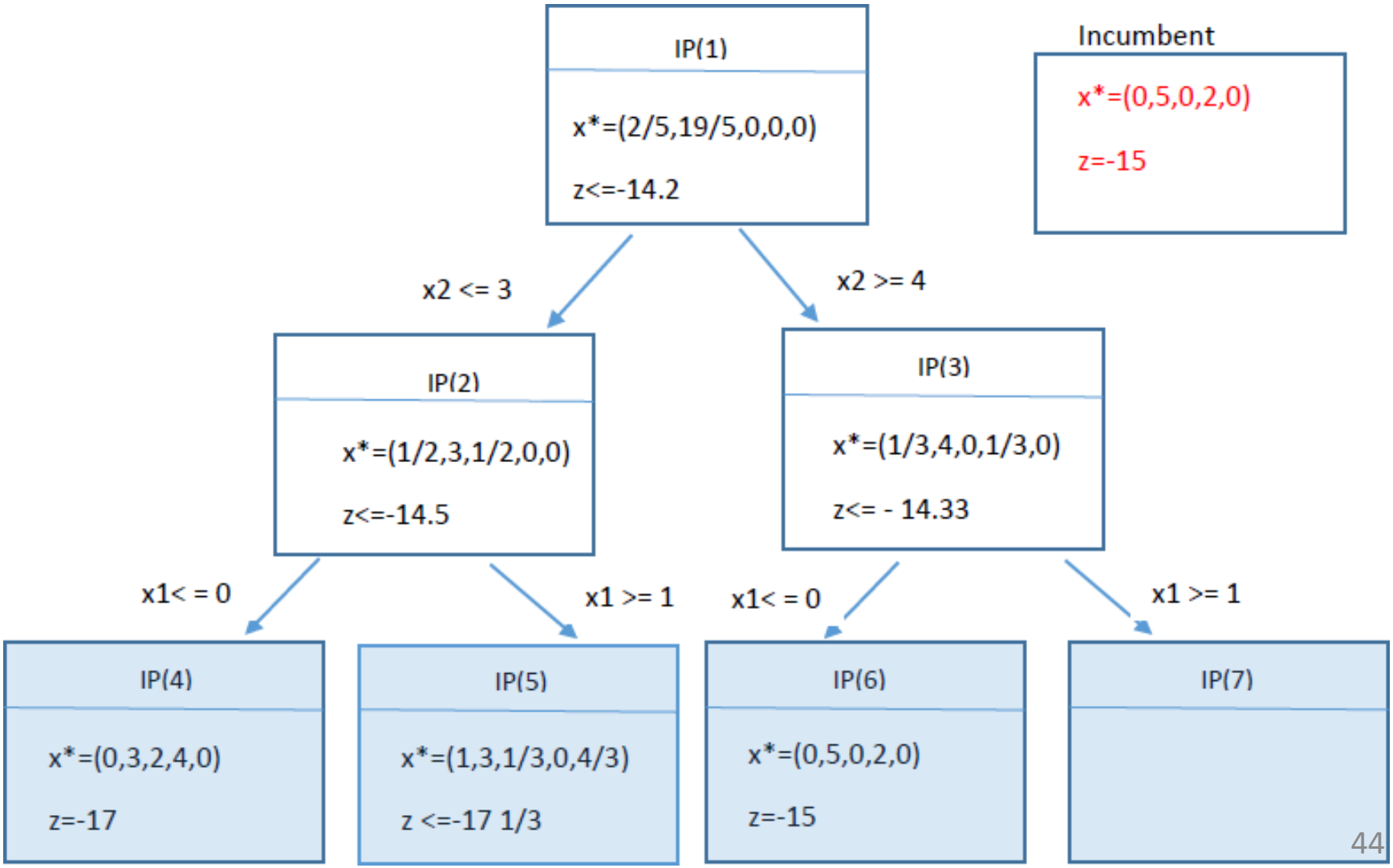
Example



Example


IP(1)
x*=(2/5,19/5,0,0,0)
z<=-14.2
Incumbent
x*=(0,3,2,4,0)
z=-17
x2 <= 3
x2 >= 4
IP(2)
x*=(1/2,3,1/2,0,0)
z<=-14.5
IP(3)
x1< = 0
x1 >= 1
IP(4)
x*=(0,3,2,4,0)
z=-17
IP(5)
x*=(1,3,1/3,0,4/3)
z <=-17 1/3

Example


IP(1)
x*=(2/5,19/5,0,0,0)
z<=-14.2
Incumbent
x*=(0,5,0,2,0)
z=-15
x2 <= 3
x2 >= 4
IP(2)
x*=(1/2,3,1/2,0,0)
z<=-14.5
IP(3)
x*=(1/3,4,0,1/3,0)
z<= - 14.33
x1< = 0
x1 >= 1
x1< = 0
x1 >= 1
IP(4)
x*=(0,3,2,4,0)
z=-17
IP(5)
x*=(1,3,1/3,0,4/3)
z <=-17 1/3
IP(6)
x*=(0,5,0,2,0)
z=-15
IP(7)

Lessons Learned

- Branch-and-bound can speed up the search.

  e.g., Only 7 nodes (LPs) out of 16 were evaluated.

- Branch-and-bound relies on eliminating subtrees, either because the IP at the node was solved, or else because the IP solution cannot possibly be optimum.
- Complete enumerations not possible (because the running time) if there are more than 100 variables. (Even 50 variables would take too long.)
- In practice, there are a lot of ways to make Branch-and-bound even faster.

How to Branch?

- We want to divide the current problem into two or more subproblems that are easier than the original. A commonly used branching method:

  $x_i \leq \lfloor x_i^* \rfloor, x_i \geq \lceil x_i^* \rceil$ where x_i^* is a fractional variable.

- Which variable to branch?

  A commonly used branching rule: Branch the most fractional variable.

- We would like to choose the branching that minimizes the sum of the solution times of all the created subproblems.
- How do we know how long it will take to solve each subproblem?

  Answer: We don't.

  Idea: Try to predict the difficulty of a subproblem.

- A good branching rule: The value of the linear programming relaxation changes a lot!

Which Node to Select?

- An important choice in branch and bound is the strategy for selecting the next subproblem to be processed.
- Goals:
  - Minimizing overall solution time.
  - Finding a good feasible solution quickly.
- Some commonly used search strategies:
  - Best First
  - Depth First
  - Hybrid Strategies

The Best First Approach

- One way to minimize overall solution time is to try to minimize the size of the search tree. We can achieve this by choosing the subproblem with the *best bound* (lowest lower bound if we are minimizing).
- Drawbacks of Best First
  - Doesn’t necessarily find feasible solutions quickly since feasible solution are "more likely" to be found deep in the tree.
  - Node setup costs are high. The linear program being solved may change quite a bit from one node evaluation to the next.
  - Memory usage is high. It can require a lot of memory to store the candidate list, since the tree can grow "broad".

The Depth First Approach

- The depth first approach is to always choose the deepest node to process next, then back up and go the other way.
- This avoids most of the problems with best first: The number of candidate nodes is minimized (saving memory). The node set-up costs are minimized.
- LPs change very little from one iteration to the next. Feasible solutions are usually found quickly.
- Drawback: If the initial lower bound is not very good, then we may end up processing lots of non-critical nodes.
- Hybrid Strategies: Go depth-first until you find a feasible solution, then do best first search.