Operations Research

Integer Programming

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Scheduling workforce again

- We know that United Airline developed an LP to determine the number of staffs in each of their service locations.
- The same problem is faced by Taco Bell.
  - It has more than 6500 restaurants in the US.
  - It asks how many staffs to have at each restaurant in each shift.
- Taco Bell developed an Integer Program (i.e., an LP with integer variables) to solve its workforce scheduling problem.
  - The number of staffs is typically **small**! Rounding is very inaccurate.
- $13 million are saved per year.
- Read the short story in Section 11.5 and the article on CEIBA.

Integer programming

- We have worked with LP for four weeks.
- In some cases, variables must only take **integer values**.
  - Producing tables and chairs in a big factory: fractional variables.
  - Selecting some books to sell (knapsack): integer variables.
  - United Airline vs. Taco Bell.
  - We will see other reasons to use integer variables.
- The subject of formulating and solving models with integer variables is **Integer Programming** (IP).
  - An IP is typically a linear IP (LIP).
  - If the objective function or any functional constraint is nonlinear, it is a nonlinear IP (NLIP).
  - We will focus on linear IP in this course.

Integer programming

- First, we will introduce one general **algorithm** for solving IPs.
  - It "decomposes" an IP to multiple LPs, solve all the LPs, and compares those outcomes to reach a conclusion.
  - Each LP is solved separately (with the simplex method or other ways).
  - In general, solving a large-scale IP can takes a very long time.
- We then demonstrate how to use **binary variables** to enrich our formulations and model more complicated situations.
- Read Sections 11.1–11.7 in the textbook.

Road map

- **Linear relaxation**.
- Branch and bound.
- Integer programming formulation.

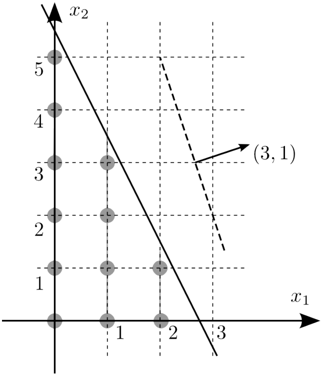
Solving an IP

- Suppose we are given an IP, how may we solve it?
- The simplex method does not work!
  - The feasible region is not "a region".
  - It is **discrete**.
  - There is no way to "move along edges".
- But all we know is how to solve LPs. How about solving a **linear relaxation** first?

Definition 1 (Linear relaxation)

For a given IP, its linear relaxation is the resulting LP after removing all the integer constraints.

$$\begin{array}{rrcrcr} \max & 3x_1 & + & x_2 & & \\ \text{s.t.} & 4x_1 & + & 2x_2 & \leq & 11 \\ & & x_i \in \mathbb{Z}_+ & \forall i = 1, 2. & & \end{array}$$



Linear relaxation

- What is the linear relaxation of

$$\begin{array}{rrcrcr} \max & x_1 & + & x_2 & & \\ \text{s.t.} & x_1 & + & 3x_2 & \leq & 10 \\ & 2x_1 & - & x_2 & \geq & 5 \\ & \multicolumn{5}{l}{x_i \in \mathbb{Z}_+ \quad \forall i = 1, 2?} \end{array}$$

  - $\mathbb{Z}$ is the set of all integers. $\mathbb{Z}_+$ is the set of all nonnegative integers.
- The linear relaxation is

$$\begin{array}{rrcrcr} \max & x_1 & + & x_2 & & \\ \text{s.t.} & x_1 & + & 3x_2 & \leq & 10 \\ & 2x_1 & - & x_2 & \geq & 5 \\ & \multicolumn{5}{l}{x_i \geq 0 \quad \forall i = 1, 2.} \end{array}$$

Linear relaxation

- For the knapsack problem

$$\begin{array}{rccccccccc} \max & 16x_1 & + & 22x_2 & + & 12x_3 & + & 8x_4 & & \\ \text{s.t.} & 5x_1 & + & 7x_2 & + & 4x_3 & + & 3x_4 & \leq & 10 \\ & & x_i \in \{0,1\} & \forall i = 1, ..., 4, & & & & & & \end{array}$$

  the linear relaxation is

$$\begin{array}{rccccccccc} \max & 16x_1 & + & 22x_2 & + & 12x_3 & + & 8x_4 & & \\ \text{s.t.} & 5x_1 & + & 7x_2 & + & 4x_3 & + & 3x_4 & \leq & 10 \\ & & x_i \in [0,1] & \forall i = 1, ..., 4, & & & & & & \end{array}$$

  - $x_i \in [0,1]$ is equivalent to $x_i \geq 0$ and $x_i \leq 1$.

Linear relaxation provides a bound

- For a **minimization** IP, its linear relaxation provides a **lower bound**.

Proposition 1

Let z^* and z' be the objective values associated to optimal solutions of a minimization IP and its linear relaxation, respectively, then $z' \leq z^*$.

Proof. They have the same objective function. However, the linear relaxation's feasible region is (weakly) larger than that of the IP. □

- For a **maximization** IP, linear relaxation provides an **upper bound**.

Linear relaxation may solve the IP

- If we are lucky, the linear relaxation may be infeasible or unbounded.
  - The IP is then infeasible or unbounded.
- If we are lucky, an optimal solution to the linear relaxation may be **feasible** to the original IP. When this happens, the IP is solved:

Proposition 2

Let x' be an optimal solutions to the linear relaxation of an IP. If x' is feasible to the IP, it is optimal to the IP.

Proof. Suppose x' is not optimal to the IP, there must be another feasible solution x'' that is better. However, as x'' is feasible to the IP, it is also feasible to the linear relaxation, which implies that x' cannot be optimal to the linear relaxation. □

- What if we are **unlucky**?

Rounding a fractional solution

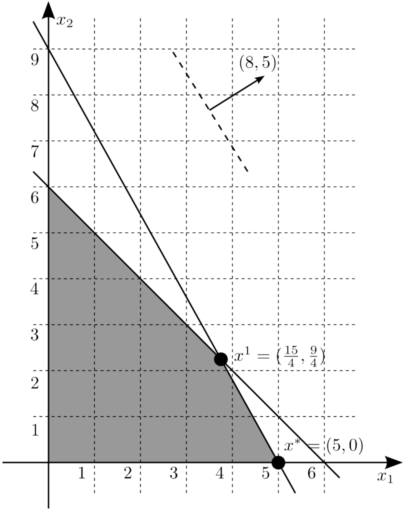
- Suppose we solve a linear relaxation with an LR-optimal solution x'.
  - "LR-optimal" means x' is optimal to the linear relaxation.
- x', however, has at least one variable violating the integer constraint in the original IP.
- We may choose to **round** the variable.
  - Round up or down?
  - Is the resulting solution always feasible?
  - Will the resulting solution be close to an IP-optimal solution x^*?

Rounding a fractional solution

- Consider the following IP

$$\begin{array}{rrcrcr} \max & 8x_1 & + & 5x_2 & & \\ \text{s.t.} & x_1 & + & x_2 & \leq & 6 \\ & 9x_1 & + & 5x_2 & \leq & 45 \\ & & & x_i \in \mathbb{Z}_+ & & \forall i = 1, 2. \end{array}$$

- $x^* = (5, 0)$ is IP-optimal.
- But $x^1 = (\frac{15}{4}, \frac{9}{4})$ is LR-optimal!
  - Rounding up any variable results in infeasible solutions.
  - None of the four grid points around x^1 is optimal.
- We need a way that guarantees to find an optimal solution.

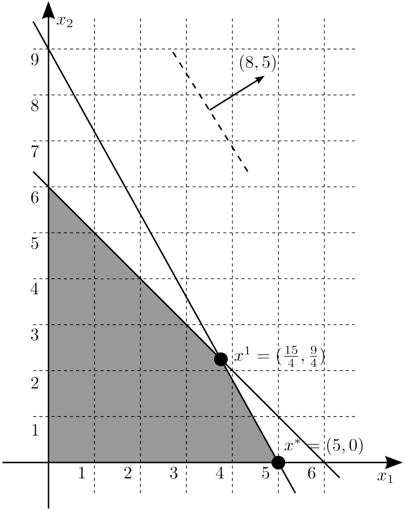


Road map

- Linear relaxation.
- **Branch and bound**.
- Integer programming formulation.

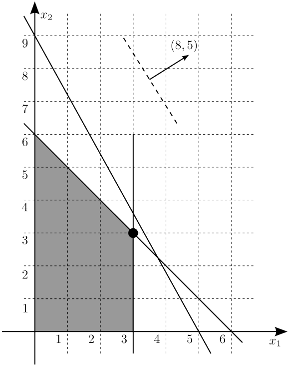
Rounding a fractional solution

- $x^1 = (\frac{15}{4}, \frac{9}{4})$ is LR-optimal.
  - Rounding up or down x_1 (i.e., adding $x_1 = 4$ or $x_1 = 3$ into the program) both **fail** to find the optimal solution.
  - Because we eliminate too many feasible points!
  - Instead of adding equalities, we should add **inequalities**.
- What will happen if we add $x_1 \geq 4$ or $x_1 \leq 3$ into the program?
- We will **branch** this problem into two problems, one with an additional constraint.

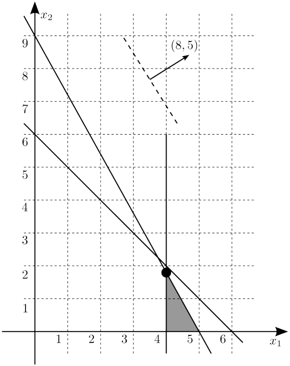


Rounding a fractional solution

If we add $x_1 \leq 3$:



If we add $x_1 \geq 4$:



- The optimal solution to the IP must be contained in one of the above two feasible regions. Why?

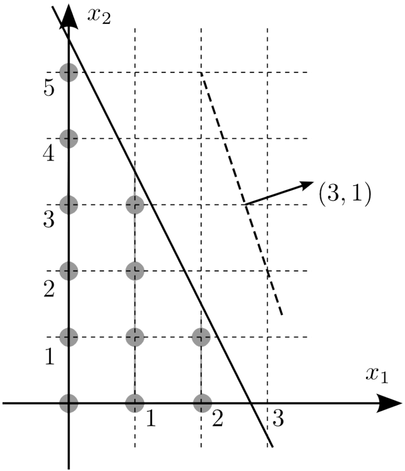
Rounding a fractional solution

- So when we solve the linear relaxation and find any variable violating an integer constraint, we will **branch** this problem into two problems, one with an additional constraint.
- The two new programs are still linear programs.
- Once we solved them:
  - If their LR-optimal solutions are both IP-feasible, compare them and choose the better one.
  - If any of them results in a variable violating the integer constraint, **branch** on that variable **recursively**.
  - Eventually compare all the IP-feasible solutions we obtain.

Example

- Let's illustrate the branch-and-bound algorithm with the following example:

$$
(P_0) \qquad \begin{array}{rrcrcl} \max & 3x_1 & + & x_2 & & \\ \text{s.t.} & 4x_1 & + & 2x_2 & \leq & 11 \\ & & & x_i \in \mathbb{Z}_+ & \forall i = 1, 2. & \end{array}
$$

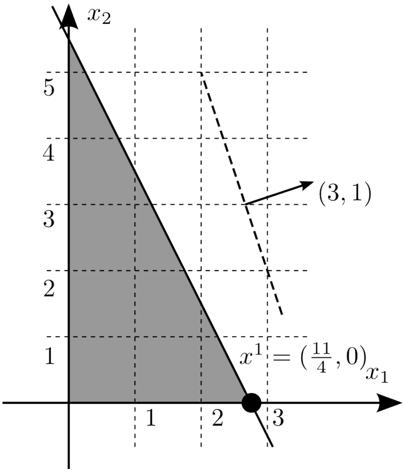


Subproblem 1

- First we solve the linear relaxation:

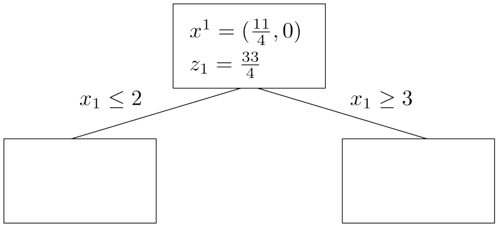
$$(P_1)\quad \begin{array}{rrcrcr} \max & 3x_1 & + & x_2 & & \\ \text{s.t.} & 4x_1 & + & 2x_2 & \leq & 11 \\ & \color{blue}{x_i \geq 0} & & \forall i = 1, 2. & & \end{array}$$

- The optimal solution is $x^1 = (\frac{11}{4}, 0)$.
- So we need to branch on x_1.



Branching tree

- The branch and bound algorithm produces a **branching tree**.
  - Each node represents a subproblem (which is an LP).
  - Each time we branch on a variable, we create two child nodes.

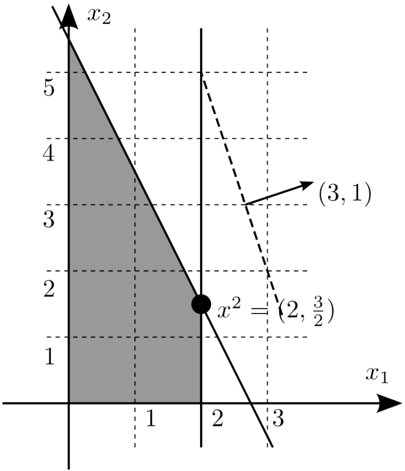


Subproblem 2

- When we add $x_1 \leq 2$:

$$(P_2) \quad \begin{array}{rrcrcr} \max & 3x_1 & + & x_2 & & \\ \text{s.t.} & 4x_1 & + & 2x_2 & \leq & 11 \\ & x_1 & & & \leq & 2 \\ & & x_i \geq 0 & \forall i = 1, 2. & & \end{array}$$

- An (P_2)-optimal solution is $x^2 = (2, \frac{3}{2})$.
  - So later we need to branch on x_2.
- Before that, let's solve (P_3).

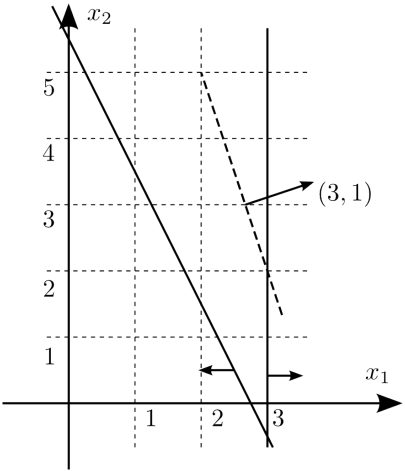


Subproblem 3

- When we add $x_1 \geq 3$:

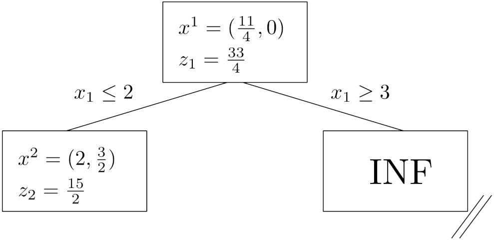
$$
(P_3) \quad \begin{array}{rrcrcr}
\max & 3x_1 & + & x_2 & & \\
\text{s.t.} & 4x_1 & + & 2x_2 & \leq & 11 \\
& \color{blue}{x_1} & & & \color{blue}{\geq} & \color{blue}{3} \\
& \multicolumn{5}{l}{x_i \geq 0 \quad \forall i = 1, 2.}
\end{array}
$$

- The problem is infeasible!
- This node is "dead" and does not produce any candidate solution.



Branching tree

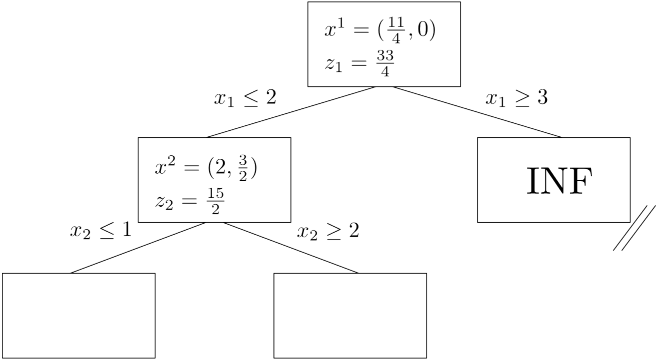
- The current progress can be summarized in the branching tree.



- Note that $z_2 = 7.5 < 8.25 = z_1$.
- In general, when we branch to the next level, the objective value associated with a subproblem-optimal solution will **always** be weakly **lower** (for a maximization problem). Why?

Branching tree

- As $x_2 = \frac{3}{2}$ in x^2, we will branch subproblem 2 on x_2.

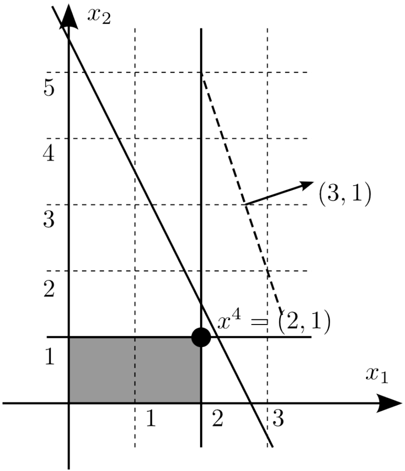


Subproblem 4

- When we add $x_2 \leq 1$:

$$
(P_4) \quad \begin{array}{rrcrcr}
\max & 3x_1 & + & x_2 & & \\
\text{s.t.} & 4x_1 & + & 2x_2 & \leq & 11 \\
& x_1 & & & \leq & 2 \\
& & & \color{blue}{x_2} & \color{blue}{\leq} & \color{blue}{1} \\
& x_i \geq 0 & & \forall i = 1, 2. & &
\end{array}
$$

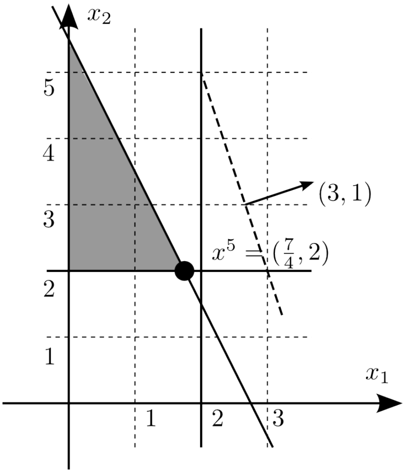
- Note that we add $x_2 \leq 1$ into subproblem 2, so $x_1 \leq 2$ is still there.



Subproblem 5

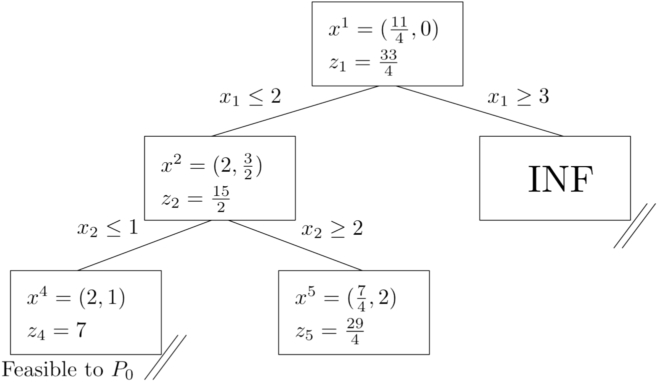
- When we add $x_2 \geq 2$:

$$
(P_5) \quad \begin{array}{rrcrcr}
\max & 3x_1 & + & x_2 & & \\
\text{s.t.} & 4x_1 & + & 2x_2 & \leq & 11 \\
 & x_1 & & & \leq & 2 \\
 & & & {\color{blue}x_2} & {\color{blue}\geq} & {\color{blue}2} \\
 & \multicolumn{5}{l}{x_i \geq 0 \quad \forall i = 1, 2.}
\end{array}
$$



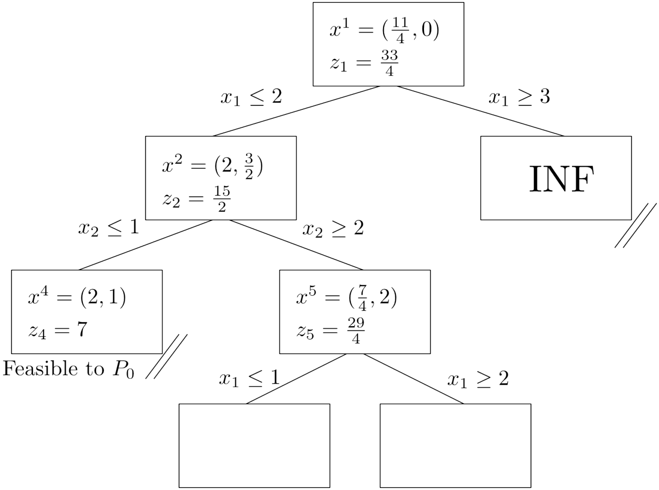
Branching tree

- x^4 satisfies all the integer constraints.
- It is IP-feasible and thus a **candidate solution** to the original IP.
- But branching subproblem 5 may result in a better solution.



Branching tree

- Let's branch subproblem 5 on x_1.

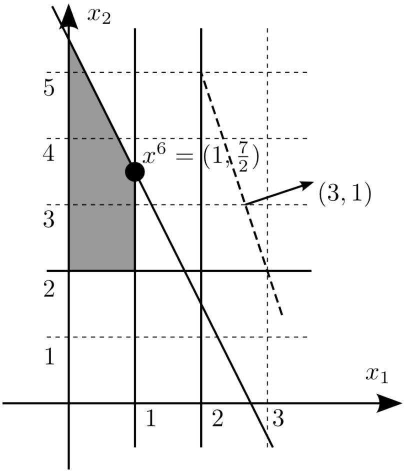


Subproblem 6

- When we add $x_1 \leq 1$:

$$
(P_6) \quad \begin{array}{rrcrcr}
\max & 3x_1 & + & x_2 & & \\
\text{s.t.} & 4x_1 & + & 2x_2 & \leq & 11 \\
 & x_1 & & & \leq & 2 \\
 & & & x_2 & \geq & 2 \\
 & \color{blue}{x_1} & & & \color{blue}{\leq} & \color{blue}{1} \\
 & x_i \geq 0 & & \forall i = 1, 2. & &
\end{array}
$$

- $x^6 = (1, \frac{7}{2})$. We may need to branch on x_2 again. However, let's solve subproblem 7 first.

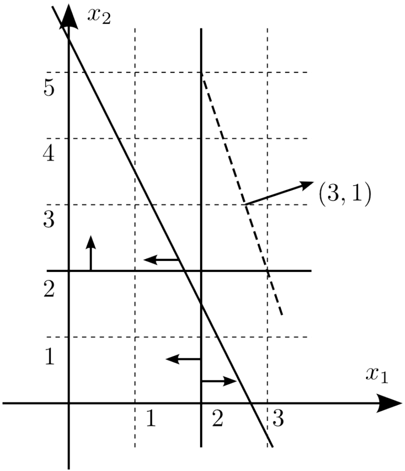


Subproblem 7

- When we add $x_1 \geq 2$:

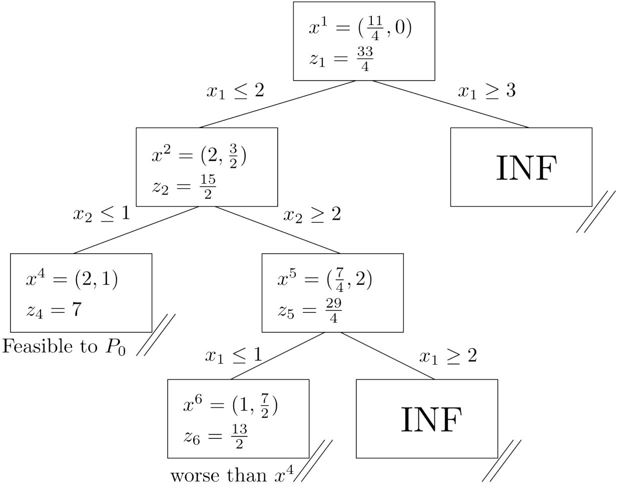
$$
(P_7) \quad \begin{array}{rrcrcr}
\max & 3x_1 & + & x_2 & & \\
\text{s.t.} & 4x_1 & + & 2x_2 & \leq & 11 \\
& x_1 & & & \leq & 2 \\
& & & x_2 & \geq & 2 \\
& \color{blue}{x_1} & & & \color{blue}{\geq} & \color{blue}{2} \\
& x_i \geq 0 & & \forall i = 1, 2. & &
\end{array}
$$

- The problem is infeasible.
- The node is "dead".



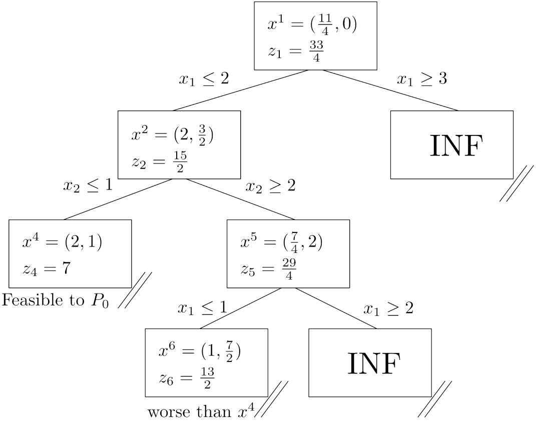
Branching tree

- The only "alive" node is subproblem 6, with x_2 fractional.
- Before we branch subproblem 6, consider the following:



Bounding

- $z_6 = \frac{13}{2}$. If we branch (P_6), all the candidate solutions (if any) under it will be (weakly) **worse** than $\frac{13}{2}$.
- However, $\frac{13}{2} < 7 = z_4$, and x_4 is already a candidate!
- So there is no need to branch (P_6). This is the "**bounding**" situation in the branch-and-bound algorithm.
  - This allows us to solve fewer subproblems.



Summary

- In running the branch-and-bound algorithm, we maintain a tree.
- If a subproblem-optimal solution is IP-feasible, set it to the candidate solution if it is currently the best among all IP-feasible solutions. Stop branching this node.
- If a subproblem is infeasible, stop branching this node.
- If a subproblem-optimal solution is not IP-feasible:
  - If it is better than the current candidate solution, branch.
  - Otherwise, stop branching.

Another example

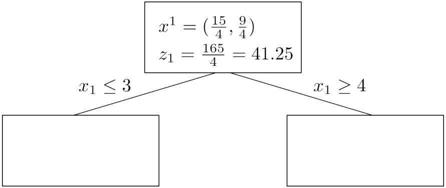
- Now let's go back to our motivating example:

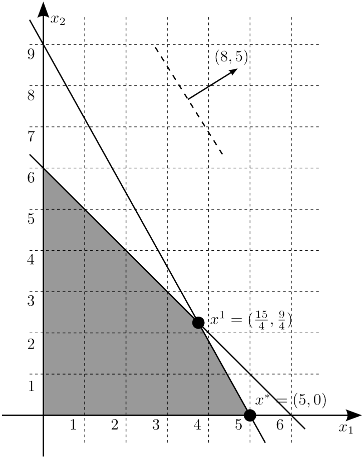
$$
(Q_0) \qquad \begin{array}{rrcrcr}
\max & 8x_1 & + & 5x_2 & & \\
\text{s.t.} & x_1 & + & x_2 & \leq & 6 \\
 & 9x_1 & + & 5x_2 & \leq & 45 \\
 & \multicolumn{5}{c}{x_i \in \mathbb{Z}_+ \quad \forall i = 1, 2.}
\end{array}
$$

- Let's solve it with the branch-and-bound algorithm.

Subproblem 1

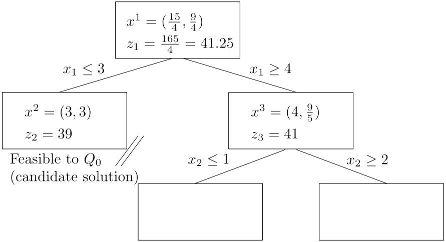
- $x^1 = (\frac{15}{4}, \frac{9}{4})$.
- We may branch on either variable. Let's branch on x_1.

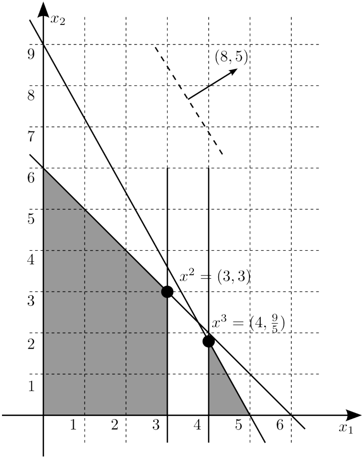




Subproblems 2 and 3

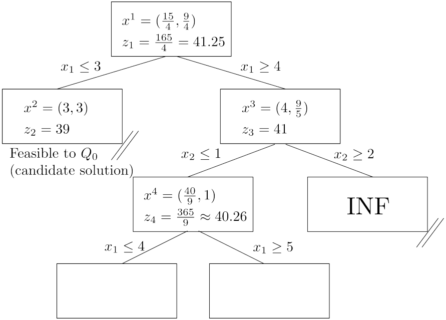
- Subproblem 2 generates a candidate solution.
- $x^3 = (4, \frac{9}{5})$. As $z_3 = 41 > z_2 = 39$, we should branch subproblem 3.

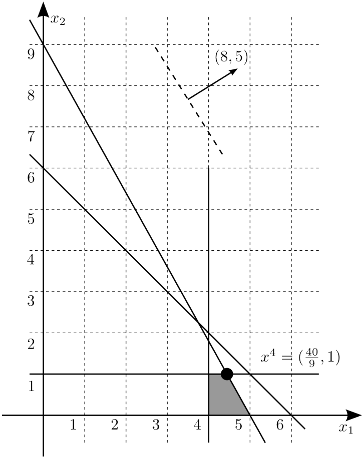




Subproblems 4 and 5

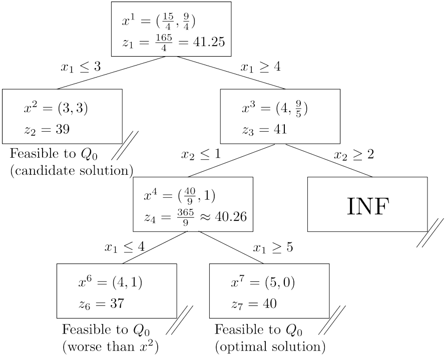
- $x^4 = (\frac{40}{9}, 1)$. As $z_4 = 40.25 > z_2 = 39$, we should branch subproblem 4.
- Subproblem 5 is infeasible.

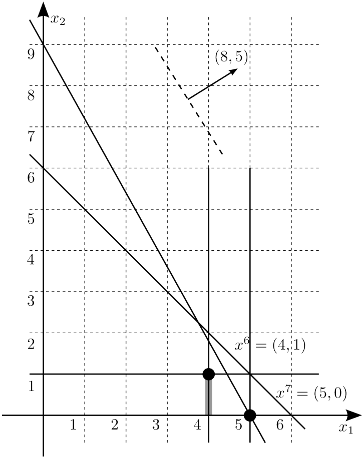




Subproblems 6 and 7

- $x^6 = (4, 1)$ but $z_6 = 37 < 39 = z_2$.
- $x^7 = (5, 0)$ and $z_7 = 40 > 39 = z_2$. As it is also the last node, x^7 is an optimal solution.





Remarks

- To select a node to branch:
  - Among all alive nodes, there are many different ways of selecting a node to branch.
  - One common approach is to branch the node with the highest objective value (for a maximization problem). Why?
  - Another popular approach is "once a node is branched, all its descendants are branched before any nondescendant. Why?
- Choosing a variable to branch on is also a challenging task.
- The branch-and-bound algorithm guarantees to find an optimal solution, if one exists.
- However, it is an **exponential-time** algorithm.
  - Roughly speaking, with n integer variables, the number of subproblems solved is approximately proportional to 2^n.

Road map

- Linear relaxation.
- Branch and bound.
- **Integer programming formulation**.

The knapsack problem

- We start our illustration with the classic **knapsack** problem.
- There are four items to select:

Item	1	2	3	4
Value ($)	16	22	12	8
Weight(kg)	5	7	4	3

- The knapsack capacity is 10 kg.
- We maximize the total value without exceeding the knapsack capacity.
- The complete formulation:

$$\begin{array}{rccccccccc} \max & 16x_1 & + & 22x_2 & + & 12x_3 & + & 8x_4 & & \\ \text{s.t.} & 5x_1 & + & 7x_2 & + & 4x_3 & + & 3x_4 & \leq & 10 \\ & & x_i \in \{0,1\} & \forall i = 1, ..., 4. & & & & & & \end{array}$$

Requirements on selecting variables

- Integer programming allows us to implement some selection rules.
- At least/most some items:
  - Suppose we must select **at least** one item among items 2, 3, and 4:

$$x_2 + x_3 + x_4 \geq 1.$$

  - Suppose we must select **at most** two items among items 1, 3, and 4:

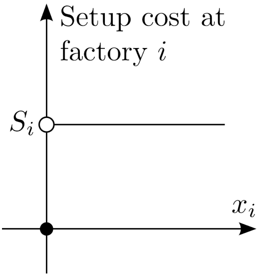
$$x_1 + x_3 + x_4 \leq 2$$

Requirements on selecting variables

- Or:
  - Select item 2 or item 3:
    $$x_2 + x_3 \geq 1.$$
  - Select item 2; otherwise, items 3 and 4 togehter:
    $$2x_2 + x_3 + x_4 \geq 2.$$
- If-else:
  - If item 2 is selected, select item 3:
    $$x_2 \leq x_3.$$
  - If item 1 is selected, do not select items 3 and 4:
    $$2(1 - x_1) \geq x_3 + x_4.$$

Fixed-charge constraints

- Consider the following example:
- n factories, 1 market, 1 product.
  - Capacity of factory i: K_i.
  - Unit production cost at factory i: C_i.
  - Demand: D.
  - We want to satisfy the demand with the minimum cost.
- **Setup cost** at factory i: S_i.
  - One needs to pay the setup cost as long as any **positive** amount of products is produced.



Basic formulation

- Let the decision variables be

$$x_i = \text{production quantity at factory } i,\ i = 1, ..., n,$$

$$y_i = \begin{cases} 1 & \text{if some products are produced at factory } i,\ i = 1, ..., n. \\ 0 & \text{o/w.} \end{cases}$$

- Objective function:

$$\min \quad \sum_{i=1}^{n} C_i x_i + \sum_{i=1}^{n} S_i y_i.$$

- Capacity limitation:

$$x_i \leq K_i \quad \forall i = 1, ..., n.$$

- Demand fulfillment:

$$\sum_{i=1}^{n} x_i \geq D.$$

Setup costs

- How may we know whether we need to pay the setup cost at factory i?
  - If $x_i > 0$, y_i must be 1; if $x_i = 0$, y_i should be 0.
- So the relationship between x_i and y_i should be:

$$x_i \leq K_i y_i \quad \forall i = 1, ..., n.$$

  - If $x_i > 0$, y_i cannot be 0.
  - If $x_i = 0$, y_i can be 0 or 1. Why y_i will always be 0 when $x_i = 0$?
- Finally, binary and nonnegative constraints:

$$x_i \geq 0, y_i \in \{0, 1\} \quad \forall i = 1, ..., n.$$

Fixed-charge constraints

- The constraint $x_i \leq K_i y_i$ is known as a **fixed-charge constraint**.
- In general, a fixed-charge constraint is

$$x \leq My.$$

  - Both x and y are decision variables.
  - $y \in \{0, 1\}$ is determined by x.
  - M must be set to be an **upper bound** of x.
  - When x is binary, $x \leq y$ is sufficient.
- We need to make M an upper bound of x.
  - For example, K_i is an upper bound of x_i in the factory example. Why?
  - What if there is no capacity limitation?

At least/most some constraints

- Using a similar technique, we may **flexibly** select constraints.
- Suppose satisfying one of the two constraints

$$g_1(x) \leq b_1 \quad \text{and} \quad g_2(x) \leq b_2$$

  is enough. How to formulate this situation?
- Let's define a binary variable

$$z = \begin{cases} 0 & \text{if } g_1(x) \leq b_1 \text{ is satisfied,} \\ 1 & \text{if } g_2(x) \leq b_2 \text{ is satisfied.} \end{cases}$$

- With M_i being an upper bound of each LHS, the following two constraints implement what we need:

$$\begin{aligned} g_1(x) - b_1 &\leq M_1 z \\ g_2(x) - b_2 &\leq M_2(1 - z). \end{aligned}$$

At least/most some constraints

- Suppose at least two of the three constraints

$$g_i(x) \leq b_i, \quad i = 1, 2, 3,$$

must be satisfied. How to play the same trick?

- Let

$$z_i = \begin{cases} 1 & \text{if } g_i(x) \leq b_i \text{ is satisfied,} \\ 0 & \text{if } g_i(x) \leq b_i \text{ may be unsatisfied.} \end{cases}$$

- With M_i being an upper bound of each LHS, the following constraints are what we need:

$$\begin{aligned} & g_i(x) - b_i \leq M_i(1 - z_i) \quad \forall i = 1, ..., 3. \\ & z_1 + z_2 + z_3 \geq 2. \end{aligned}$$

If-else constraints

- In some cases, if $g_1(x) > b_1$ is satisfied, then $g_2(x) \leq b_2$ must also be satisfied.
- How to model this situation?
  - First, note that "if A then B" $\Leftrightarrow$ "(not A) or B".
  - So what we really want to do is $g_1(x) \leq b_1$ or $g_2(x) \leq b_2$.
  - So simply select at least one of $g_1(x) \leq b_1$ and $g_2(x) \leq b_2$!

Route selection

WASTE MANAGEMENT

- Waste Management Inc. operates an recycling network with 293 landfill sites, 16 waste-to-energy plants, 72 gas-to-energy facilities, 146 recycling plants, 346 transfer stations, and 435 collection depots.
  - 20000 routes must be go through by its vehicles in each day.
- How to determine a route?
  - Construct a network with nodes and edges.
  - Give each edge a **binary** variable: 1 if included and 0 otherwise.
  - Constraints are required to make sure that selected edges are really forming a route.
- A huge IP is constructed to save the company $498 million in operational expenses over a 5-year period.
- Read the short story in Section 11.7 and the article on CEIBA.