CIVIL-557

Decision Aid Methodologies In Transportation

Lecture 6: Integer programming: Branch and Bound

Virginie Lurkin, Nikola Obrenović

Transport and Mobility Laboratory TRANSP-OR
École Polytechnique Fédérale de Lausanne EPFL

TRANSP-OR

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

On the previous lecture…

- Container Storage Inside a Container Terminal
  - Two-phase simplex method
  - Duality theory

Linear programming

What if the situation is different…

Suppose 2 new containers are expected to arrive for storage in the next planning period of a terminal. Suppose there are only 2 blocks in the terminal, each with 20 storage spaces. For the moment, there are ~~6~~ **10** containers in block 1 and ~~12~~ **9** containers in block 2.

$\boldsymbol{N = 2}$
$\boldsymbol{B = 2}$
$\boldsymbol{A = 20}$
$\boldsymbol{a_1 = 10}$
$\boldsymbol{a_2 = 9}$
$\boldsymbol{F = \frac{10+9+2}{2 \times 20} = 0.525}$

$$\begin{aligned} &\textit{Minimize} && u_1^+ + u_1^- + u_2^+ + u_2^- \\ &\textit{subject to} \\ & && x_1 - u_1^+ + u_1^- = 0.5 \\ & && x_2 - u_2^+ + u_2^- = 1.5 \\ & && x_1 + x_2 = 2 \\ & && x_1, x_2, u_1^+, u_1^-, u_2^+, u_2^- \geq 0 \end{aligned}$$

- Optimal solution: $(\mathbf{0.5}, \mathbf{1.5}, 0, 0, 0, 0)$
- x_1 and x_2 represent the number of containers to be assigned, respectively, to block 1 and 2
- x_1 and x_2, in reality, should be integer

Linear (mixed) integer programming

Linear integer programming

Pure integer programming (IP):

$$\min_{x \in \mathbb{Z}^n} c^T x$$

subject to $Ax \leq b$

$x \geq 0,$

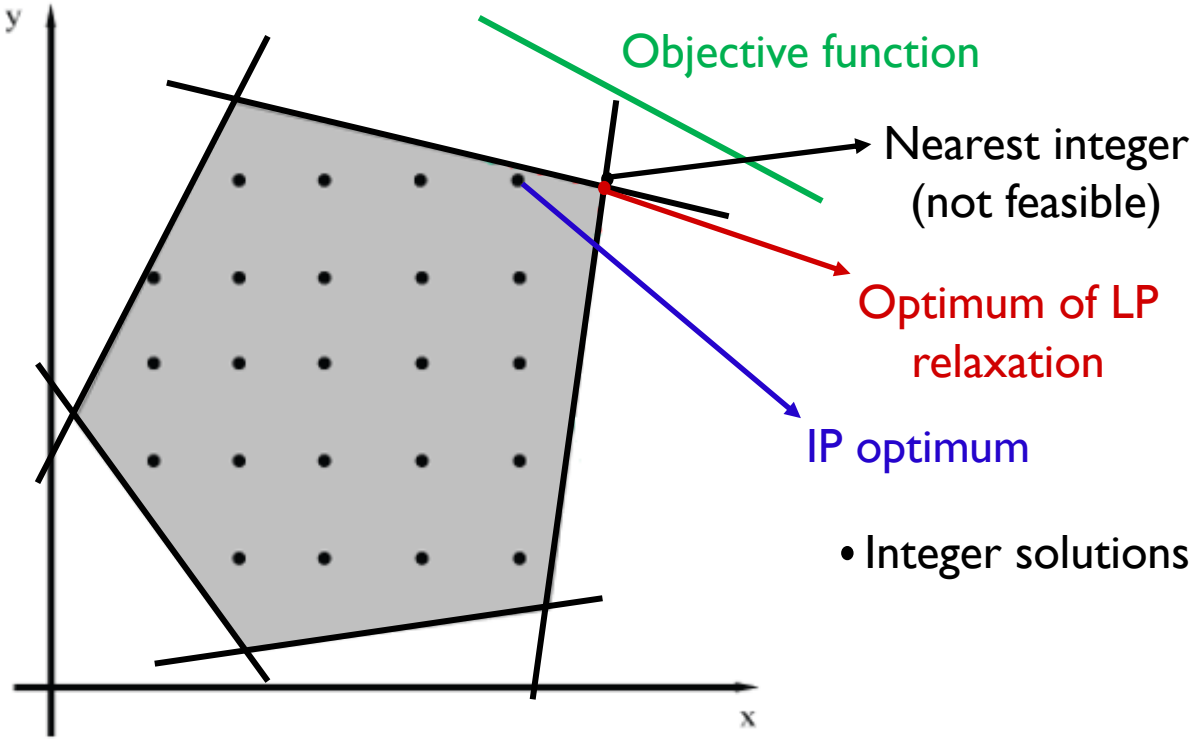
where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$

- A **mixed-integer programming (MIP)** problem is one where only some of the decision variables are constrained to be integer values

Linear (mixed) integer programming

How to solve integer programming problem?

- Intuitive idea: remove the integer constraints from the IP formulation (LP relaxation)
  - Solve the LP relaxation and round to the nearest integer



Not the best idea!

Linear (mixed) integer programming

How to solve integer programming problem?

- There is no optimality conditions for integer programming problem
- Explicit enumeration is normally impossible due to the exponentially increasing number of potential solutions

- Exact methods can be used to find the **optimal solution:**
  - Two main methods: Branch and Bound (B&B) and Cutting planes
  - Computing a true optimal solution might be very costly

- **Heuristic** methods can be used to find a **good solution**:
  - Does not guarantee optimality

Branch and Bound algorithm

Branch and Bound algorithm

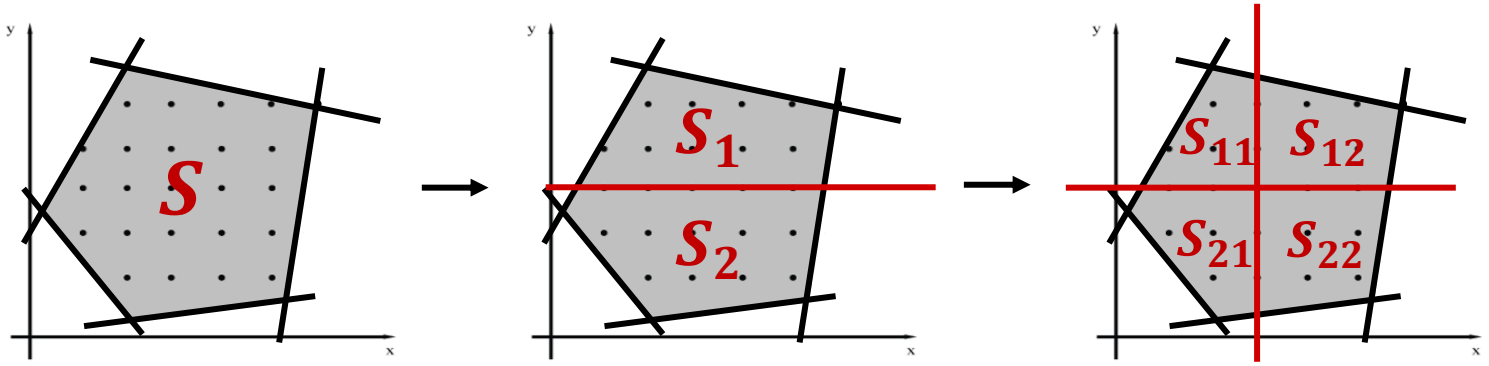
General idea

- Implicit enumeration of the feasible set, by **ruling out large sets of feasible points**
- How ?
  - By **partitioning** the feasible set into smaller subsets
  - By **using bounds** to identify subsets that are guaranteed not to contain an optimal solution

Branch and Bound algorithm

Partitioning:

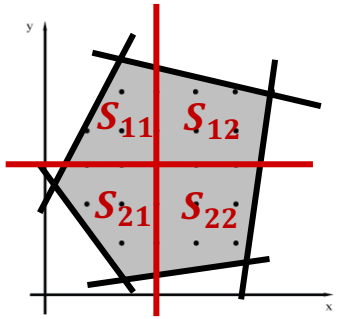
- **Partitioning** the feasible set into smaller subsets
- **Partitioning** the original problem into smaller subproblems



Branch and Bound algorithm

Theorem 1:

- P_k is an optimization problem (**minimization**)
- S is the feasible set of the optimization problem P
- S is partitioned into K subsets: $S_1 \cup \ldots \cup S_m \cup S_{m+1} \cup \ldots \cup S_K$



For each $k = 1,2, \ldots, K$, let $let\ x_k^*$ be an optimal solution of the optimization problem P_k(minimization). Let i be such that:

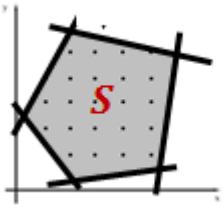
$$f(x_i^*) \leq f(x_k^*), \qquad k = 1, \ldots, K$$

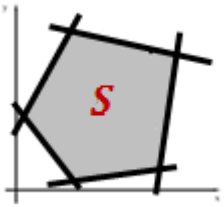
Then x_i^* is an optimal solution of the optimization problem P

Branch and Bound algorithm

Using bounds:

- Any feasible solution provides an upper bound on the optimal solution (minimization)
- A feasible solution is a solution that satisfies all the constraints
- For each feasible subproblem, we will calculate a lower bound (*best* possible value)
- Lower bounds are obtained by solving relaxations of the original subproblems
- Calculate lower bounds is much easier (simplex algorithm can be used)

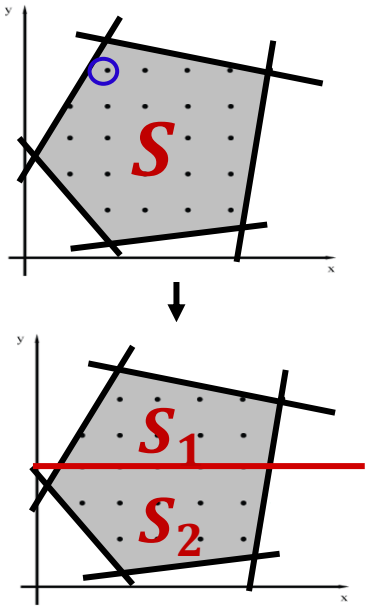


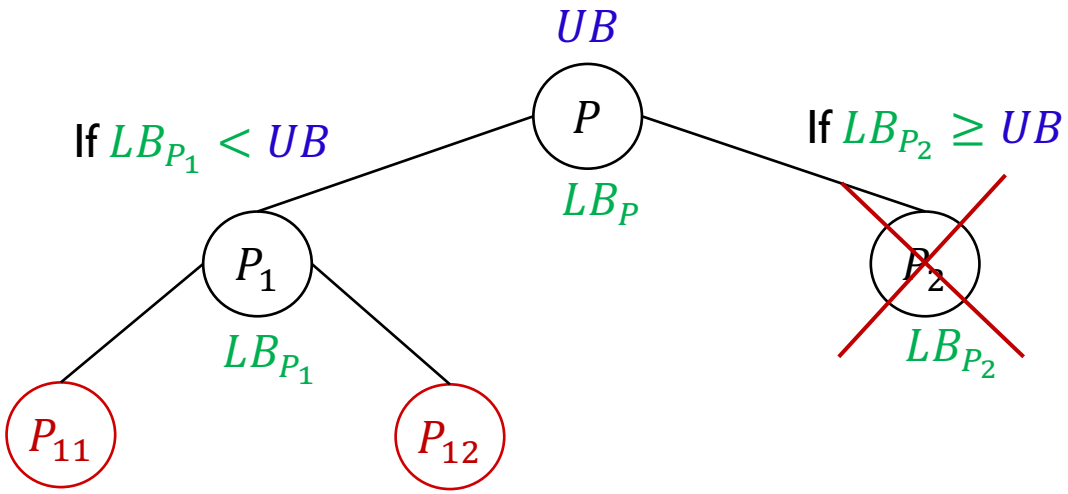


Branch and Bound algorithm

Using bounds:

- Some subproblems are excluded based on the value of the lower bound





- Branching refers to exploring the subtree of an active node
- Bounding refers to estimating a lower bound at an active node

Branch and Bound algorithm

Theorem 2:

For each $k = 1,2, \dots, m$, let $let\ x_k^*$ be an optimal solution of P_k.

For each $k = m+1, \dots, K$, let $l(P_k)$ be a lower bound on P_k:

$$l(P_k) \leq f(x_k)\ \ \forall k \in S_k$$

Let i be such that:

$$f(x_i^*) \leq f(x_k^*),\ \ k = 1, \dots, m$$

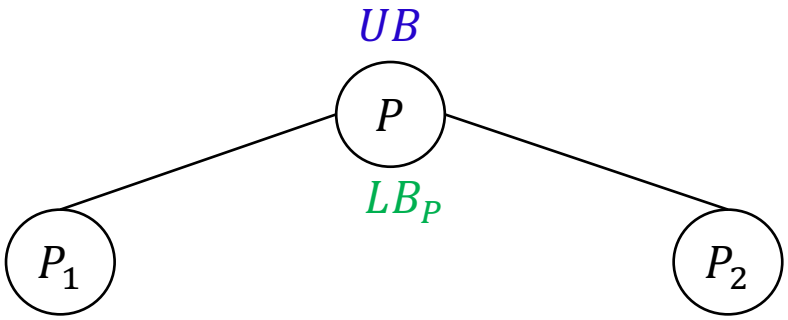
and

$$f(x_i^*) \leq l(P_k), k = m+1, \dots, K$$

Then x_i^* is an optimal solution of the optimization problem P

Branch and Bound algorithm

Branching:



How to branch?

- Select a variable x_j that has a fractional value:

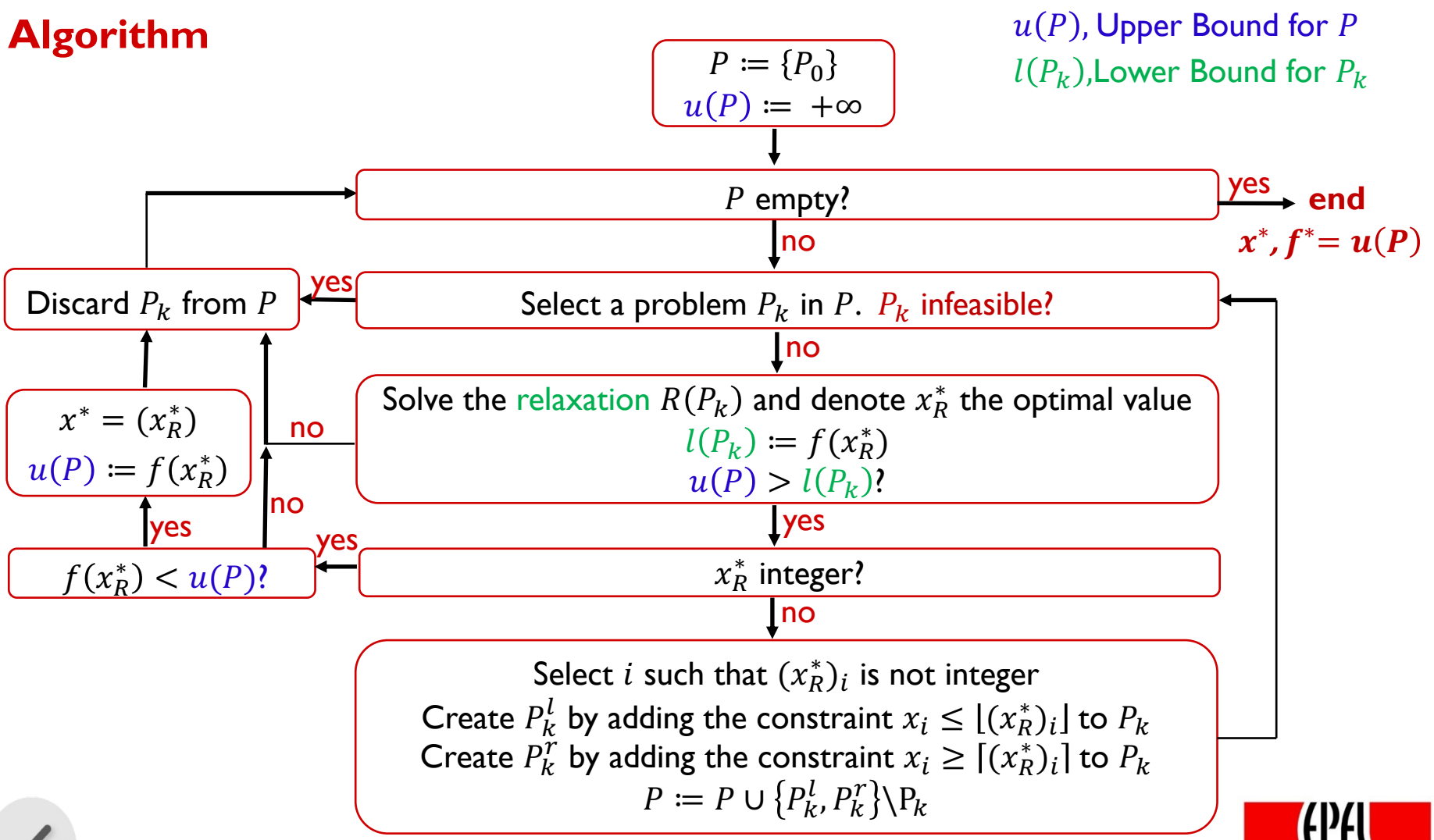
  $k < x_j < k + 1$, with k integer

- Create two new LP-subproblems:

  1. Previous LP-subproblem + constraint $x_j \leq k$ $\quad (k = \lfloor x_j \rfloor)$
  2. Previous LP-subproblem + constraint $x_j \geq k + 1$ $\quad (k + 1 = \lceil x_j \rceil)$

Branch and Bound algorithm

Algorithm

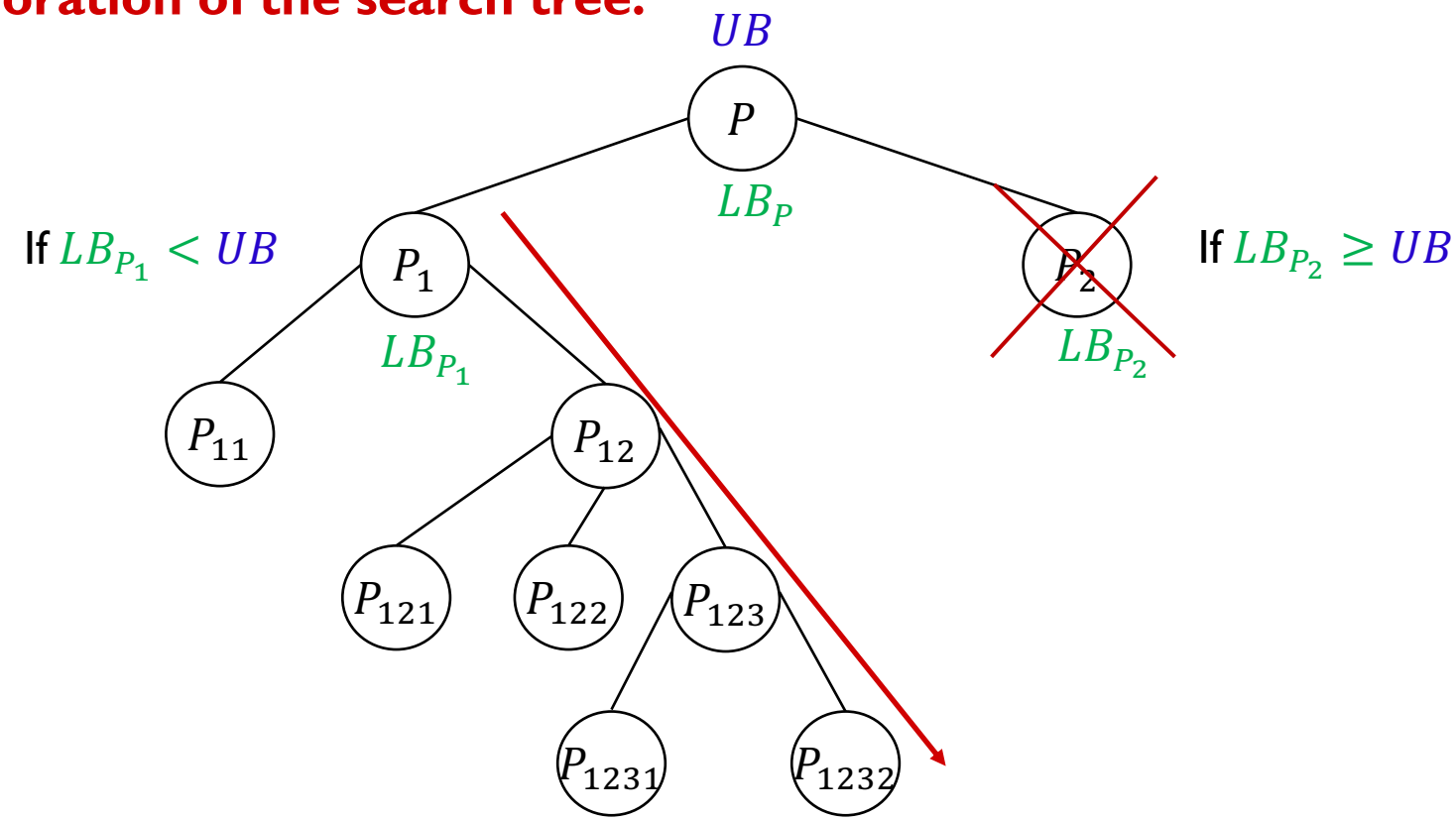


TRANSP-OR

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Branch and Bound algorithm

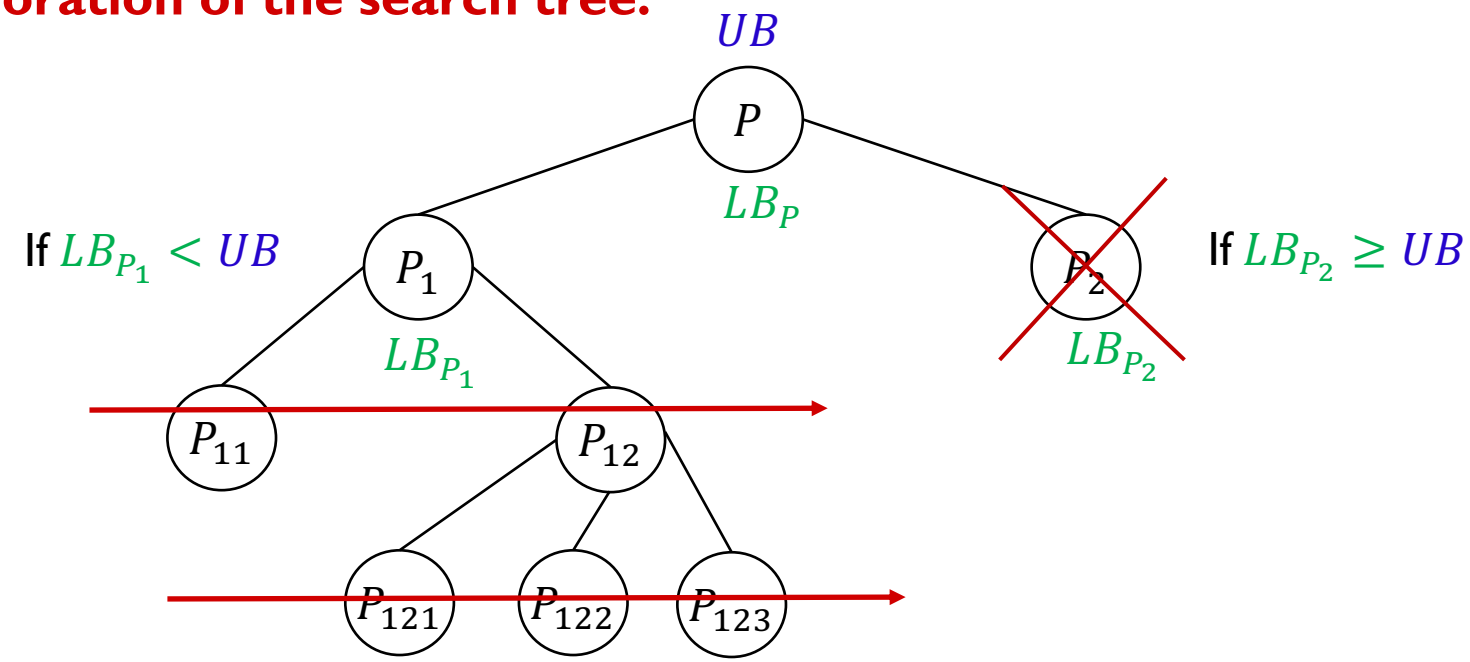
Exploration of the search tree:



- Depth-first search (DFS): explores as far as possible along each branch before backtracking

Branch and Bound algorithm

Exploration of the search tree:



- Breadth-first search (BFS): explores nodes level by level
- Depending on the problem at hand, either DFS or BFS could be advantageous

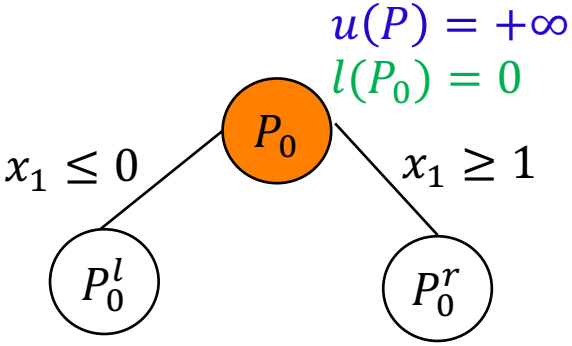
Branch and Bound algorithm

Back to our case:

$R(P_0)$

$$
\begin{aligned}
&\textit{Minimize} && u_1^+ + u_1^- + u_2^+ + u_2^- \\
&\textit{subject to} && \\
& && x_1 - u_1^+ + u_1^- = 0.5 \\
& && x_2 - u_2^+ + u_2^- = 1.5 \\
& && x_1 + x_2 = 2 \\
& && x_1, x_2, u_1^+, u_1^-, u_2^+, u_2^- \geq 0
\end{aligned}
$$

- $x_R^* = (0.5, 1.5, 0, 0, 0, 0)$ and $f(x_R^*) = 0$
- $l(P_0) = 0$
- $u(P) > l(P_0)$
- x_1, x_2 are not integers
- Select x_1 for branching



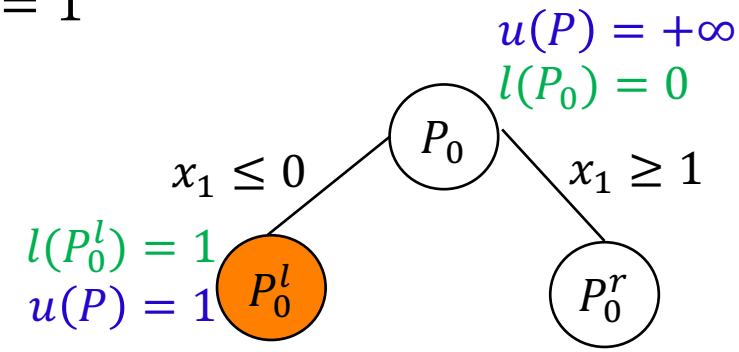
Branch and Bound algorithm

Back to our case:

$R(P_0^l)$

$$
\begin{aligned}
&\textit{Minimize} && u_1^+ + u_1^- + u_2^+ + u_2^- \\
&\textit{subject to} && \\
& && x_1 - u_1^+ + u_1^- = 0.5 \\
& && x_2 - u_2^+ + u_2^- = 1.5 \\
& && x_1 + x_2 = 2 \\
& && x_1 \leq 0 \\
& && x_1, x_2, u_1^+, u_1^-, u_2^+, u_2^- \geq 0
\end{aligned}
$$

- $x_R^* = (0, 2, 0, 0.5, 0.5, 0)$ and $f(x_R^*) = 1$
- $l(P_0^l) = 1$
- $u(P) > l(P_0)$
- x_1, x_2 are integers → $u(P) = 1$
- Discard P_0^l



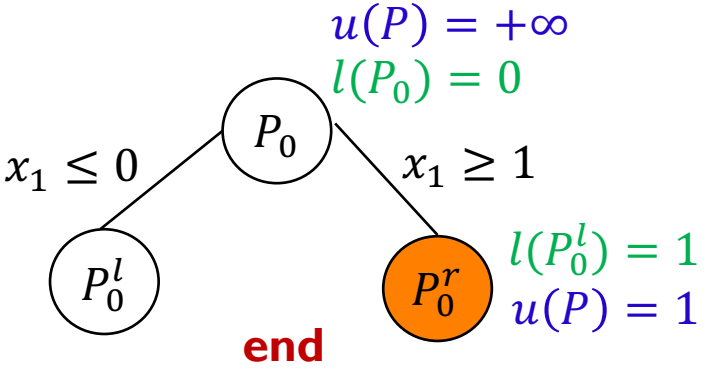
Branch and Bound algorithm

Back to our case:

$R(P_0^r)$

$$
\begin{aligned}
&\textit{Minimize} && u_1^+ + u_1^- + u_2^+ + u_2^- \\
&\textit{subject to} && \\
& && x_1 - u_1^+ + u_1^- = 0.5 \\
& && x_2 - u_2^+ + u_2^- = 1.5 \\
& && x_1 + x_2 = 2 \\
& && x_1 \geq 1 \\
& && x_1, x_2, u_1^+, u_1^-, u_2^+, u_2^- \geq 0
\end{aligned}
$$

- $x_R^* = (1, 1, 0.5, 0, 0, 0.5)$ and $f(x_R^*) = 1$
- $l(P_0^r) = 1$
- $u(P) > l(P_0)$
- x_1, x_2 are integers
- Discard P_0^r



end

$$f(x_R^*) = 1$$

$$x_R^* = (0, 2, 0, 0.5, 0.5, 0) \text{ or } x_R^* = (1, 1, 0.5, 0, 0, 0.5)$$

Some integer programming tricks

Integer programming tricks

Example 1

Suppose that a manager must decide which of n warehouses to use for meeting the demands of m customers for a good. The decisions to be made are which warehouses to operate and how much to ship from any warehouse to any customer

Fixed costs

- There is a fixed operating cost associated to each warehouse
- Decisions must be made about tradeoffs between transportation costs and costs for operating distribution centers
- Very common issue in modeling distribution systems

Integer programming tricks

Fixed costs

- We introduce an indicator variable

$$y_i = \begin{cases} 1 & \text{if warehouse } i \text{ is opened} \\ 0 & \text{if warehouse } i \text{ is not opened} \end{cases}$$

- Other decisions variables:

x_{ij} = amount to be sent from warehouse i to customer j

- Data:

f_i = fixed operating cost for warehouse i

p_i = per-unit operating cost at warehouse i

c_{ij} = transportation cost for shipping from warehouse i to customer j

Integer programming tricks

Fixed costs

- Model:

$$\min \quad \sum_{i=1}^{m}\sum_{j=1}^{n}(p_i+c_{ij})x_{ij} + \sum_{i=1}^{m} f_i y_i$$

$$\text{s.t.} \quad \sum_{i=1}^{m} x_{ij} = d_j\,, \forall j = 1,2,\dots,n$$

$$\sum_{j=1}^{n} x_{ij} \leq \boldsymbol{M} y_i, \forall i = 1,2,\dots,m \qquad \boldsymbol{M} \text{ is a big number}$$

$$x_{ij} \geq 0, \forall i = 1,2,\dots,m, \forall j = 1,2,\dots,n$$

$$y_i \in \{0,1\}, \forall i = 1,2,\dots,m$$

Integer programming tricks

Example 2

Suppose that a company has to choose among two modes of operation for the manufacturing process of its products on a single machine

Either… or …

- A mode of operation might better suit to some products
- Each mode has its own performance (number of units produced by minute)
- Very common issue in modeling manufacturing process

Integer programming tricks

Either… or …

- We introduce an indicator variable

$$y_j = \begin{cases} 1 & \text{if process 1 is chosen for product } j \\ 0 & \text{if process 2 is not chosen for product } j \end{cases}$$

- Other decisions variables:

x_j = number of units of product j produced by the company

- Data:

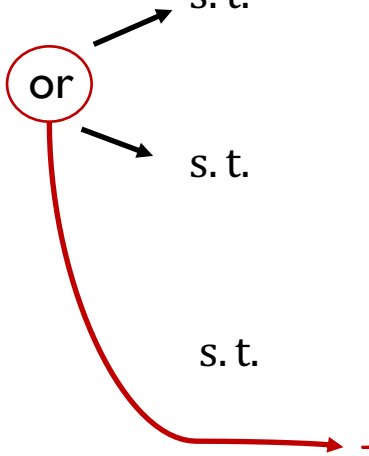
a_{ij} = per-unit time needed (in minutes) to produce product j using process i

b_i = maximum amount of time the machine can be used if process i is used

Integer programming tricks

Either… or …

- Constraints:



s. t. $\sum_{j=1}^{n} a_{1j} x_j \leq b_1$ if process 1 is chosen for product j, i.e. if $y_j = 1$

s. t. $\sum_{j=1}^{n} a_{2j} x_j \leq b_2$ if process 2 is chosen for product j, i.e. if $y_j = 0$

s. t. $\sum_{j=1}^{n} a_{1j} x_j y_j \leq b_1$

$\sum_{j=1}^{n} a_{2j} x_j (1 - y_j) \leq b_2$

$x_j \geq 0, \forall j = 1,2, \ldots, n$

$y_j \in \{0,1\}$

Integer programming tricks

If… then …

- If process 1 is applied to product 1, it has to be applied to product 2

$$\text{if } y_1 = 1, \text{then } y_2 = 1$$

- Constraints:

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{1j} x_j y_j \leq b_1$$

$$\sum_{j=1}^{n} a_{2j} x_j (1 - y_j) \leq b_2$$

$$x_j \geq 0, \forall j = 1,2, \dots, n$$

$$y_j \in \{0,1\}$$

$$y_2 \geq y_1$$

Integer programming tricks

Elimination of products of variables

- Data:

  c_{ij} = per-unit cost to produce product j using process i

- Objective function:

$$\min \quad \sum_{j=1}^{n} (c_{1j} x_j y_j + c_{2j} x_j (1 - y_j))$$

$$\sum_{j=1}^{n} ((c_{1j} - c_{2j}) x_j y_j + c_{2j} x_j)$$

nonlinear

Integer programming tricks

Elimination of products of variables

- In general, a product of two variables can be replaced by one variable,
- A number of new constraints is imposed on the new variable

continuous variables

binary variables

$$\min \sum_{j=1}^{n} ((c_{1j} - c_{2j}) x_j y_j + c_{2j} x_j)$$

Product of a binary variable and a continuous variable

Upper bound on x_j $\quad 0 \le x_j \le u$

New variable z_j $\quad z_j = x_j \times y_j$

$$\begin{cases} z_j \le u y_j \\ z_j \le x_j \\ z_j \ge x_j - u(1 - y_j) \\ z_j \ge 0 \end{cases}$$

Integer programming tricks

Product of two binary variables

binary variables $x_j y_y$ binary variables

New variable z_j $\quad z_j = x_j \times y_j$

$$\begin{cases} z_j \leq x_j \\ z_j \leq y_j \\ z_j \geq x_j + y_j - 1 \\ z_j \in \{0,1\} \end{cases}$$

Product of two continuous variables

continuous variables $x_j y_y$ continuous variables

New variables z_j and w_j $\quad z_j^2 - w_j^2 = x_j \times y_j$

Quadratic function

$$\begin{cases} z_j = \frac{1}{2}(x_j + y_j) \\ w_j = \frac{1}{2}(x_j - y_j) \end{cases}$$

Heuristics

Heuristics

Algorithms and complexity measures

- An algorithm is a finite set of instructions that when executed on an input can produce an output after finitely many steps.
- The input encodes an instance of the problem that the algorithm is supposed to solve, and the output encodes a solution for the given problem instance.
- The computation of an algorithm on an input is the sequence of steps it performs.
- Algorithms do not always terminate; they may perform an infinite computation for some, or any, input.

Heuristics

Algorithms and complexity measures

- There are problems that can be easily and precisely defined in mathematical terms, yet that cannot be solved by any algorithm (untrackable / uncomputable problems).

- In practice, we cannot expect to be able to solve real world problem instances of arbitrary size to optimality.

- Depending on the size of an instance or depending on the available CPU time we will often have to be satisfied with computing approximate solutions.

Heuristics

Algorithms and complexity measures

- The term heuristic is used for algorithms which find solutions among all possible ones, but they do not guarantee that the best will be found.

- A good heuristic algorithm should fulfil the following properties:
  - A solution can be obtained with reasonable computational effort.
  - The solution should be near optimal (with high probability).
  - The likelihood for obtaining a bad solution (far from optimal) should be low.

How can we evaluate the performance of an heuristic algorithm ?

Heuristics

A concrete example for our container storage problem

1. Rearrange blocks so that a_i increases with i
2. Determine x_i in increasing order of i using:

$$x_1 = \min\{N, \max\{0, A\times\mathrm{F} - a_1\}\}$$

$$x_i = \min\{\max\{0, A\times\mathrm{F} - a_i\}, N - \sum_{r=1}^{i-1} x_r\} \qquad \forall i \geq 2$$

Linear (mixed) integer programming

IP and MIP have extremely wide applications in practice

Knapsack problem

Which containers should be loaded in the vessel?

Knapsack problem

- Data

  w_i: weight if container i

  p_i: price of container i

  C: capacity of the vessel

- Decision variables

$$x_i = \begin{cases} 1 & \text{If container } i \text{ is selected} \\ 0 & \text{Otherwise} \end{cases}$$

- Objective function

$$\textbf{max} \sum_i p_i x_i$$

- Constraints

$$\sum_i w_i x_i \leq C$$

Knapsack problems

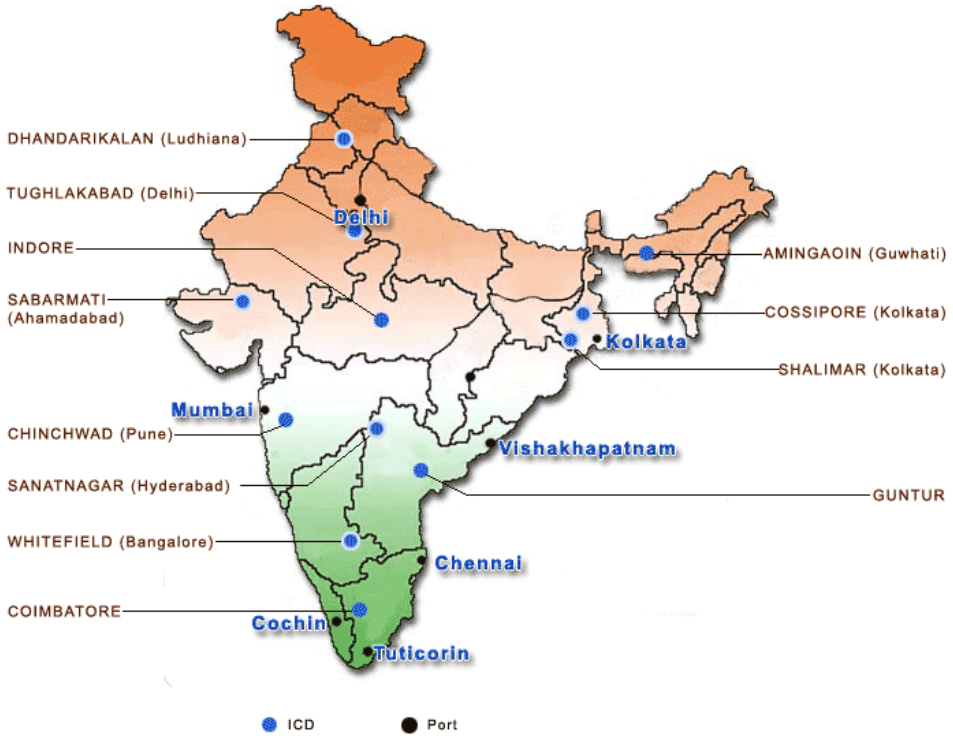
Which boxes should be loaded in the container in order to maximize the revenue?

Related problems deal with the loading of containers in trains or aircrafts

Facility location problem

Where to locate inland container depots?

- **Inland container depots**: storage area for shipping containers situated at inland points away from sea ports



TRANSP-OR

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Facility location problem

- Data

  I: set of m alternative facility locations

  J: set of n customer zones

  c_{ij}: per-unit cost for supplying customer zone j by using the facility at i

  f_i: fixed cost of establishing a facility location i

  b_i: capacity of facility location i

  d_j: total demand for customer zone j

- Decision variables

$$y_i = \begin{cases} 1 & \text{If a facility is established at location } i \\ 0 & \text{Otherwise} \end{cases}$$

x_{ij} = Units of customer zone j's demand satisfied by facility at location i

Facility location problem

- Objective function

$$\textbf{max} \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{m} f_i y_i$$

- Constraints

$$\sum_{i=1}^{m} x_{ij} = d_j, \qquad \forall j = 1, \dots, n$$

$$\sum_{j=1}^{n} x_{ij} \leq b_i y_i, \qquad \forall i = 1, \dots, m$$

$$x_{ij} \geq 0, \qquad \forall i = 1,2, \dots, m; \forall j = 1,2, \dots, n$$

$$y_i \in \{0,1\}, \qquad \forall i = 1,2, \dots, m$$

Assignment problem

Berth allocation problem

- The berth allocation problem consists of assigning incoming ships to berthing position

Assignment problem

- Data

  B: set of m berths

  V: set of n vessels

  h_{ij}: handling time of vessel i at berth j

- Decision variables

$$y_{ij} = \begin{cases} 1 & \text{If a vessel } j \text{ is allocated to berth } i \\ 0 & \text{Otherwise} \end{cases}$$

- Objective function

$$\textbf{min} \sum_{i=1}^{m} \sum_{j=1}^{n} h_{ij} y_{ij}$$

Assignment problem

- Constraints

$$\sum_{i=1}^{m} y_{ij} \leq 1, \qquad \forall j = 1, \ldots, n$$

$$\sum_{j=1}^{n} y_{ij} = 1, \qquad \forall i = 1, \ldots, m$$

$$y_{ij} \in \{0,1\}, \qquad \forall i = 1,2, \ldots, m$$

Summary

- Making Branch and Bound work well in practice requires lots of good ideas
- Heuristics are often used to find good initial solutions
- An idea for speeding up Branch and Bound is to add valid inequalities (Gomory cuts)

« Divide ut imperes »

Main references

- Murty, K. G. (2015). Intelligent Modeling Essential to Get Good Results: Container Storage Inside a Container Terminal. In *Case Studies in Operations Research* (pp. 1-15). Springer New York.
- Bierlaire, M. (2015). *Optimization: principles and algorithms*. EPFL Press, Lausanne, Switzerland.