Combinatorial Optimization and the B&B Method

Optimization Methods in Management Science
Master in Management
HEC Lausanne

Dr. Rodrigue Oeuvray

Fall 2019 Semester

Combinatorial Optimization

A combinatorial optimization problem is characterized by:

- a finite set of feasible solutions Ω,
- an objective function $f : \Omega \to \mathbb{R}$ assigning a value to each feasible solution

It consists in determining the solution $\boldsymbol{x}^* \in \Omega$ minimizing or maximizing f

$$\boldsymbol{x}^* = \text{argmin/max} \{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \Omega\} = \underset{\boldsymbol{x} \in \Omega}{\text{argmin/max}} \, f(\boldsymbol{x})$$

A Finite Set

- A set is **finite** if it has a finite number of elements
- Informally, a finite set is a set which one could in principle count and finish counting
- Examples:
  - $\{1, 2, 3, 4, 5\}$ is a finite set with 5 elements
  - $\mathbb{R}$ and $\mathbb{Z}$ aren't finite sets

Examples of Combinatorial Problems

- **Minimal weight spanning tree**: it is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, with no cycle and with the minimum possible total edge weight

- **Graph coloring**: in its simplest form, it is a way of coloring with the minimum number of different colors the vertices of a graph such that no two adjacent vertices share the same color

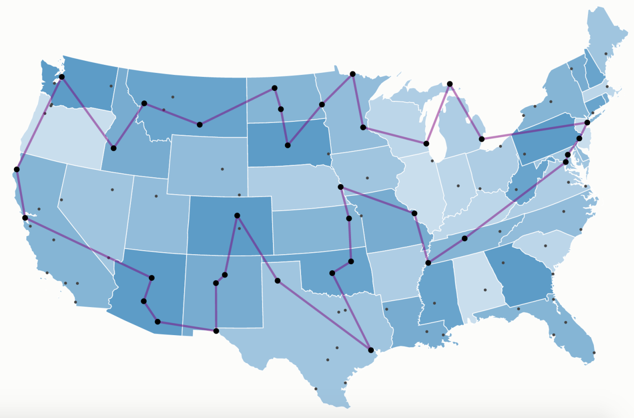
- **Shortest path problem**: it is the problem of finding a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimized

The Travelling Salesman Problem (TSP)

Problem (TSP)

The travelling salesman problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

TSP



The Knapsack Problem

Problem

Given a set of items $i = 1, \ldots, n$ each with a **volume** a_i and a **utility** c_i, determine the number of each item to include in a collection so that the **capacity** (total volume) is less than or equal to a given limit b and the **total utility** is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items

Let's define for each item a **boolean** decision variable:

$$x_i = \begin{cases} 1 & \text{if we keep item } i \\ 0 & \text{otherwise} \end{cases} \qquad i = 1, \ldots, n$$

The Knapsack Problem (Cont'd)

To determine an optimal selection of the items, we have to solve an **integer** linear program:

$$\begin{array}{llll} \text{Max} & z = & \displaystyle\sum_{i=1}^{n} c_i x_i & \\ \text{s.t.} & & \displaystyle\sum_{i=1}^{n} a_i x_i & \leq \quad b \end{array}$$

$$x_i \in \{0,1\}, c_i, a_i, b > 0 \quad \forall\, i$$

The Continuous Knapsack Problem

To determine an optimal selection of the items, we have to solve a linear program:

$$
\begin{aligned}
\text{Max} \quad z = \quad & \sum_{i=1}^{n} c_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_i x_i \quad \leq \quad b \\
& 0 \leq x_i \leq 1, c_i, a_i, b > 0 \quad \forall\, i
\end{aligned}
$$

The Continuous Knapsack Problem (Cont'd)

- It is very easy to determine the solution of the **continuous** problem !
- We classify the items in the decreasing order of the ratios $\rho_i = c_i / a_i$
- Let k be the index defined by $\sum_{i=1}^{k-1} a_i \leq b$ and $\sum_{i=1}^{k} a_i > b$. Then the **optimal** solution is simply given by

$$x_i = \begin{cases} 1 & i < k \\ \frac{b - \sum_{i=1}^{k-1} a_i}{a_k} & i = k \\ 0 & i > k \end{cases}$$

- Concretely, there is **at most** one variable which is **fractional** in the optimal solution

The Continuous Knapsack Problem : Example

- We consider the following example :

$$\begin{array}{lrcrcrcrcrcl} \text{Max} & z = & 7x_1 & + & 8x_2 & + & 11x_3 & + & 4x_4 & & \\ \text{s.t.} & & 4x_1 & + & 5x_2 & + & 7x_3 & + & 3x_4 & \leq & 15 \\ & & & & & & \multicolumn{5}{l}{0 \leq x_i \leq 1 \quad i = 1, 2, 3, 4} \end{array}$$

- We classify the items in the decreasing order of the ratios $\rho_i = c_i / a_i$:

Item i	1	2	3	4
Utility c_i	7	8	11	4
Volume a_i	4	5	7	3
$\rho_i = c_i / a_i$	1.75	1.6	1.57	1.33

Capacity $b = 15$

Example (Cont'd)

- x_1 has the **highest** normalized utility, then x_2, x_3, and x_4
- We take x_1, the remaining capacity is 11. Then x_2 with a remaining capacity of 6. We can only take a fraction of x_3: 6/7. The remaining capacity is null
- We conclude that the **optimal** solution is: $x_1 = 1, x_2 = 1, x_3 = 6/7$, $x_4 = 0$ for a total utility of $\bar{z}^* = 24 + 3/7$
- **Data**:

Item i	1	2	3	4
Utility c_i	7	8	11	4
Volume a_i	4	5	7	3
$\rho_i = c_i/a_i$	1.75	1.6	1.57	1.33

Capacity $b = 15$

Problem Instance

Definition

A problem refers to the abstract question to be solved. In contrast, an **instance** of this problem is a rather concrete utterance, which can serve as the **input** for a decision problem. Stated another way, the instance is a **particular input** to the problem, and the solution is the output corresponding to the given input

Problem-Solving Methods

- **Universal algorithm** for the resolution of combinatorial optimization problem: **enumerate** all the feasible solutions $\boldsymbol{x} \in \Omega$ and keep the best of them !
- But the cardinality of Ω can be very huge ! Imagine that you can select or not any of 100 items. They are $2^{100} \approx 10^{30}$ different combinations !
- In practice, the **enumeration** method only works for small size problems
- We generally classify problem-solving methods in three categories: heuristics, approximations, and exact methods

Heuristic Methods

- A **heuristic** method is an algorithm providing not necessary an optimal solution but a **satisfactory** solution in a **reasonable** time

- Meta-heuristics (non-exhaustive list): genetic algorithms, simulated annealing, tabu search, variable neighborhood search, greedy randomized adaptive search procedures, ...

Approximation Algorithms

- An approximation algorithm is a way of dealing with NP-completeness for optimization problems
- There is no guarantee that we will get the best solution
- The goal of an approximation algorithm is to come **as close as possible** to the optimal value in a **reasonable** amount of time (**at most polynomial**)

Approximation Algorithms (Cont'd)

- The **quality** of the solution can be measured in absolute or relative term. Let $\phi_H(I)$ be the value provided by the heuristic for a **specific** instance I and $\phi^*(I)$ an **optimal** solution, then H has
  - an absolute performance of α **if** $\left|\dfrac{\phi_H(I)}{\phi^*(I)}\right| \leq \alpha$ for **every** instance I of the problem
  - a relative performance of β **if** $\left|\dfrac{\phi_H(I) - \phi^*(I)}{\phi^*(I)}\right| \leq \beta$ for **every** instance I of the problem
- A heuristic with an absolute performance of α is an α-**approximation**

Exact Problem-Solving Methods

- Exact methods provide an **optimal** solution, but they may be extremely time-consuming when solving real-world problems
- Non-exhaustive list of **exact** methods:
  - enumerative,
  - branch and bound,
  - dynamic programming,
  - ...

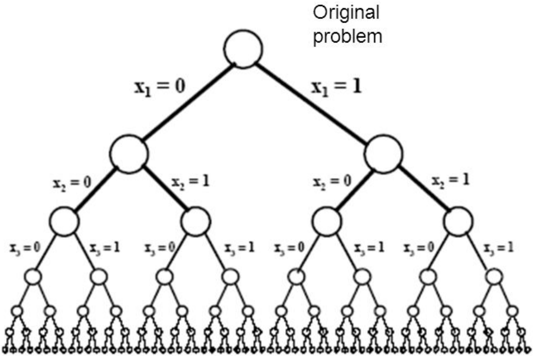
Introduction to The Branch and Bound (B&B) Method

- Enumeration tree
- Implicit enumeration
- Relaxation to find a bound
- The generic B&B algorithm
- Application to the knapsack problem
- Final remarks

Enumeration Tree

- Let's assume that we have the following variables: $x_1, \cdots, x_n$ with $x_i \in \{0, 1\}$
- A recursive way of enumerating all the feasible solutions is to set x_1 to 0 and to recursively enumerate all the solutions for the reduced-size problem. Then set x_1 to 1, and do the same
- This procedure yields to an **enumeration tree**

An Enumeration Tree



A vertex in an enumeration tree is called a node. At the **initial** node, **none** of the variables are **fixed**. For the **leaves**, this is the opposite. All the variables are fixed

Implicit Enumeration

- We cannot afford to enumerate **all** combinations !
- We must try to enumerate the overwhelming part of all combinations **implicitly** !
- The trick is to focus only on the parts of the enumeration tree that can lead to an optimal solution !
- If a part of the enumeration tree cannot provide any better solution than the current one, then we simply **discard** it

Relaxation

- We can achieve an **upper** bound on an optimization problem like the knapsack problem by computing an **optimal** solution **over a larger set** of feasible solutions
- We can allow more solutions by getting rid of some constraints - hopefully in such a way that the new problem is **easier** to solve
- This approach is called a **relaxation**
- The milder the effect of a relaxation on the objective value, the better our estimate !

Linear Relaxation

- The most commonly used relaxation consists in **dropping** the constraint that **variables are integer**
- In the knapsack problem for instance, we replace $x_i \in \{0, 1\}$ by $0 \leq x_i \leq 1$
- Then, optimizing the **relaxed** problem calls for solving a linear program and we know efficient techniques to solve it (example: the simplex algorithm) !

The Branch and Bound (B&B) Method

- A **branch-and-bound** algorithm consists of an **implicit** enumeration of the feasible solutions
- The set of feasible solutions Ω is thought of as forming a **rooted** tree (the enumeration tree) with the **full set at the root**
- The algorithm explores **branches** of this tree, which represent **subsets** Ω_i of the **solution set**

The Branch and Bound (B&B) Method (Cont'd)

- Before enumerating the feasible solutions of a branch, the branch is **checked** against a bound on the optimal solution, and is **discarded** if it cannot produce a better solution than the best one found so far
- This bound is obtained with a **relaxation** of some constraints of the problem

The Generic B&B Algorithm

Assumptions: we assume that we want to **maximize** an objective function f and one requires a **bounding** function g, that computes **upper** bounds of f on the nodes of the enumeration tree

Algorithm:

(1) Using a **heuristic**, find a solution $\boldsymbol{x_h}$ to the problem. Store its value, $Val = f(\boldsymbol{x_h})$. If no heuristic is available, set $Val = -\infty$. Val will denote the **best** solution value found so far, and will be used as a lower bound on candidate solutions

The Generic B&B Algorithm (Cont'd)

(2) Initialize a set S with the initial node (the root). S is the set of **active** nodes

(3) **Loop** until S is empty:

- (3.1) Take a node $\boldsymbol{N}$ off S
- (3.2) If $\boldsymbol{N}$ represents a single candidate solution $\boldsymbol{x}$ (meaning that all the variables are fixed) and $f(\boldsymbol{x}) > Val$, then $\boldsymbol{x}$ is the best solution so far. Record it and set $Val = f(\boldsymbol{x})$
- (3.3) Else, **branch** on $\boldsymbol{N}$ to produce new nodes $\boldsymbol{N}_i$. For each of these:
  - (3.3.1) If $g(\boldsymbol{N}_i) < Val$, do nothing; since the upper bound on this node is smaller than Val, it will never lead to an optimal solution, and can be **discarded**
  - (3.3.2) Else, store $\boldsymbol{N}_i$ in S

Application to the Knapsack Problem (KP)

Let's consider the knapsack problem:

$$
\begin{array}{llcccccccc}
\text{Max} & z = & 7x_1 & + & 8x_2 & + & 11x_3 & + & 4x_4 & & \\
\text{s.t.} & & 4x_1 & + & 5x_2 & + & 7x_3 & + & 3x_4 & \leq & 15 \\
& & & & & & & & x_i \in \{0,1\} & & i = 1,2,3,4
\end{array}
$$

corresponding to the following input data:

Item i	1	2	3	4
Utility c_i	7	8	11	4
Volume a_i	4	5	7	3
$\rho_i = c_i/a_i$	1.75	1.6	1.57	1.33

Capacity $b = 15$

The row ρ_i corresponds to the normalized utility (utility divided by the volume)

KP: Relaxed Problem

- The relaxation consists in replacing $x_i \in \{0, 1\}$ by $0 \leq x_i \leq 1$
- The **relaxed** problem is given by:

$$\begin{array}{llcccccccccc} \text{Max} & z = & 7x_1 & + & 8x_2 & + & 11x_3 & + & 4x_4 & & \\ \text{s.t.} & & 4x_1 & + & 5x_2 & + & 7x_3 & + & 3x_4 & \leq & 15 \\ & & & & & & \multicolumn{5}{l}{0 \leq x_i \leq 1 \quad i = 1, 2, 3, 4} \end{array}$$

- It is very easy to determine a solution to it !

KP: Relaxed Problem (Cont'd)

- x_1 has the **highest** normalized utility, then x_2, x_3, and x_4
- We take x_1, the remaining capacity is 11. Then x_2 with a remaining capacity of 6. We can only take a fraction of x_3: 6/7. The remaining capacity is null
- We conclude that the optimal solution of the linear relaxation is: $x_1 = 1, x_2 = 1$, $x_3 = 6/7, x_4 = 0$ for a total utility of $\bar{z}_0 = 24 + 3/7$
- $\bar{z}_0$ is an **upper bound** for the original problem
- **We have the guarantee than none of the feasible solution will have a total utility larger than $\bar{z}_0$**
- **Data**:

Item i	1	2	3	4
Utility c_i	7	8	11	4
Volume a_i	4	5	7	3
$\rho_i = c_i/a_i$	1.75	1.6	1.57	1.33

Capacity $b = 15$

Steps in the Resolution of the Knapsack Problem - 1

- During the "**bound**" phase, we solve the linear relaxation, i.e, the linear program obtained by replacing the constraints $x_i \in \{0, 1\} \; \forall \, i$ by $0 \leq x_i \leq 1 \; \forall \, i$ for some variables that have not yet been **fixed**
- This relaxation problem is particularly **easy to solve** and there is **at most** one variable which can be **fractional** in this solution
- This bound is used to decide if it is worth or not continuing branching the active node under consideration. If this **upper** bound is **smaller** than the **current best** solution, then this **active** node is simply **discarded** and we consider another active node
- This just means that we **cannot improve** the current solution by **branching** this node

Steps in the Resolution of the Knapsack Problem - 2

- If the current node is not discarded:
  - if the solution of the relaxation problem is **not integer**, then we create **two sub-problems** by branching x_j, the first one with $x_j = 1$ and the second one with $x_j = 0$, where x_j is the **fractional** variable in the linear relaxation
  - if the solution of the relaxation problem is **integer**:
    - then this is not only a feasible solution of the problem (all variables are integer) but this is also the best one so far !
    - Then we update the current best solution value with the new one and we move to another active node
- We continue the process until there is no active node any more

KP - Resolution (1)

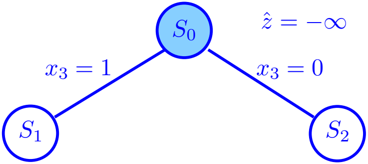
Initialization: The set S of active nodes is initialized with S_0 which represents the initial problem. The **current best solution value** $\hat{z}$ is set to $-\infty$ (this is a *max* problem)

Iteration 1: we compute an upper bound $\bar{z}_0$ for S_0 (relaxation: $0 \leq x_i \leq 1, i = 1, \cdots, 4$). The solution of the relaxed problem is:

$$x_1 = 1, \quad x_2 = 1, \quad x_3 = 6/7, \quad x_4 = 0, \qquad \bar{z}_0 = 24 + 3/7$$

It means that we won't find any feasible solution (if any) with a value larger than 24 +3/7. We **branch** S_0 in two sub-problems, one with $x_3 = 1$ and the other with $x_3 = 0$

KP - Resolution (2)

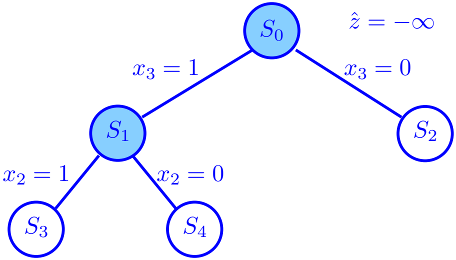


Iteration 2: The two active nodes are S_1 and S_2. We determine an upper bound $\bar{z}_1$ for S_1 ($x_3 = 1$, relaxation: $0 \leq x_i \leq 1, i = 1, 2, 4$). The solution of the relaxed problem is:

$$x_3 = 1, \quad x_1 = 1, \quad x_2 = 4/5, \quad x_4 = 0, \qquad \bar{z}_1 = 24.4$$

This solution is not integer. As $\bar{z}_1 > \hat{z}$, it means that it is worth continuing the branching. We create two new sub-problems with $x_2 = 1$ and $x_2 = 0$ respectively

KP - Resolution (3)

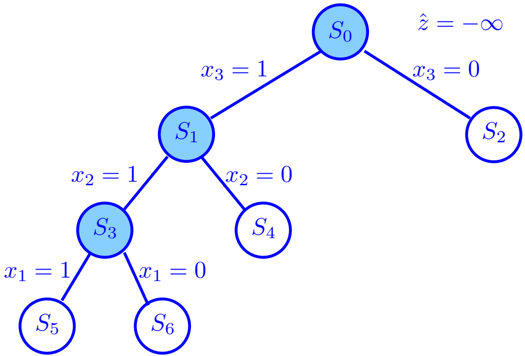


Iteration 3: Let's continue with the active node S_3 ($x_3 = 1, x_2 = 1$, relaxation: $0 \leq x_i \leq 1, i = 1, 4$). The solution of the relaxed problem is:

$$x_2 = 1, \quad x_3 = 1, \quad x_1 = 3/4, \quad x_4 = 0, \qquad \bar{z}_3 = 24.25$$

$\bar{z}_3 > \hat{z}$, we split S_3 by branching on x_1

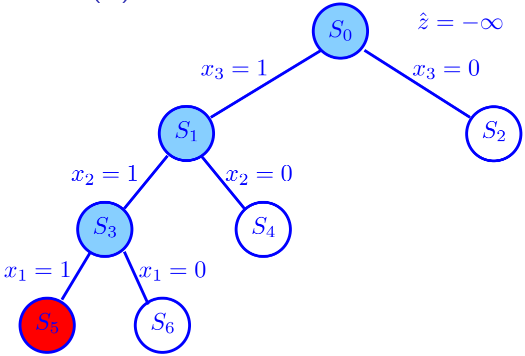
KP - Resolution (4)



Iteration 4: Let's continue with the active node S_5. We have that $x_1 = x_2 = x_3 = 1$ but $a_1 + a_2 + a_3 = 4 + 5 + 7 = 16 > 15 = b$. This solution is not feasible and the set Ω_5[1] is empty

[1] We remind that Ω_i is the subset of feasible solutions associated with node i

KP - Resolution (5)

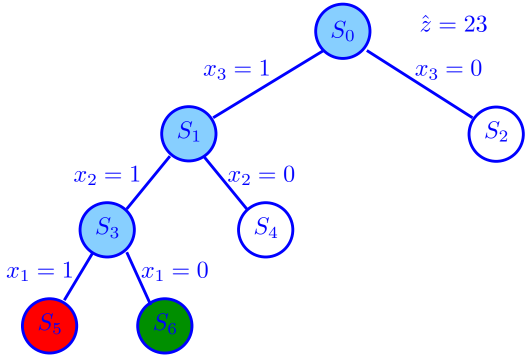


Iteration 5: We continue with the active node S_6 ($x_3 = 1, x_2 = 1, x_1 = 0$, relaxation: $0 \leq x_4 \leq 1$). The solution of the relaxed problem is:

$$x_1 = 0, \quad x_2 = 1, \quad x_3 = 1, \quad x_4 = 1, \qquad \bar{z}_6 = 23$$

This is an **integer** feasible solution. We update $\hat{z} = 23$ and we **record** this solution

KP - Resolution (6)

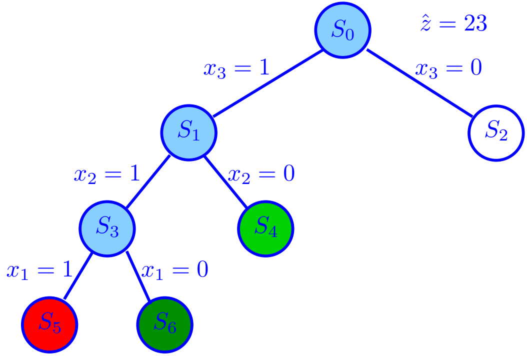


Iteration 6: We continue with S_4. The solution of the relaxed problem is:

$$x_2 = 0, \quad x_3 = 1, \quad x_1 = 1, \quad x_4 = 1, \qquad \bar{z}_4 = 22$$

The solution is integer but $\bar{z}_4 < \hat{z}$. This node is **discarded**

KP - Resolution (7)

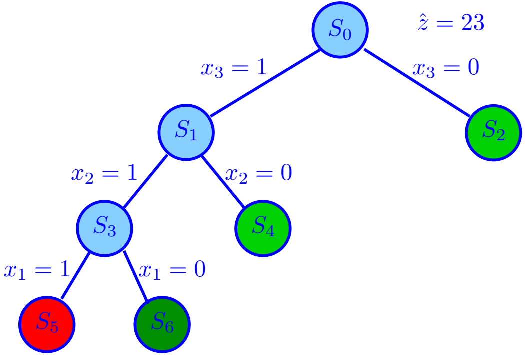


Iteration 7: We continue with S_2. The solution of the relaxed problem is:

$$x_3 = 0, \quad x_1 = 1, \quad x_2 = 1, \quad x_4 = 1, \qquad \bar{z}_2 = 19$$

The solution is integer but $\bar{z}_2 < \hat{z}$. This node is **discarded**

KP - Resolution (8)



There is **no active** node any more. The process is **completed**. The **optimal** solution corresponds to the solution associated with node S_6. We need to choose items 2, 3, and 4 for a maximal utility of 23

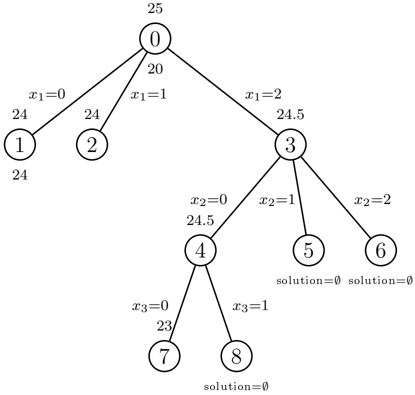
Bounds at a Node

We consider a **maximization** problem

- The solution of the relaxation at node i gives an **upper** bound $\bar{z}_i$ for the **partial** problem that we consider
- If the solution of the relaxed problem at node i is **integer**, then its value is also a **lower** bound $\underline{z}_i$ of the **initial** problem (the optimal solution value should be larger than $\underline{z}_i$)

Enumeration Tree with Some Node Bounds

- We consider a *max* problem
- Node 0: upper bound 25. We branch on x_1
- Node 1: the relaxation problem gives an integer feasible solution for the partial problem
- The current best solution value is updated to 24
- Node 2: discarded
- Node 3: upper bound of 24.5. We branch on x_2
- Node 4: upper bound 24.5. We branch on x_3
- Node 7: discarded
- Node 8: empty solution
- Node 5: empty solution
- Node 6: empty solution
- **Output**: the **optimal** solution is given by node 1



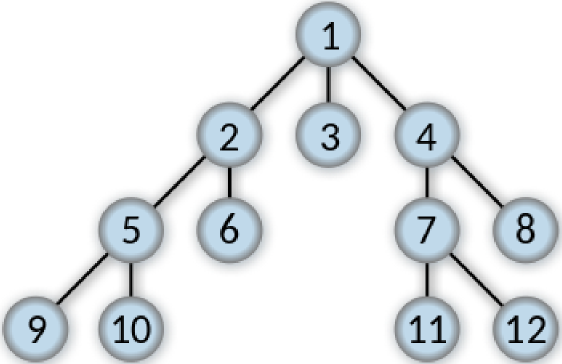
Liberties in B&B

- In a general Branch-and-Bound scheme, we have some **liberty**:
  - Which active node shall we look at next ?
  - Which variable should we branch on ?
- We would like to dive into the search tree in order to find a feasible solution **quickly**
- When diving, the question which node to pick next comes down to: which of the two son nodes shall we follow first?

Liberties in B&B (Cont'd)

- They are **many** different strategies that can be implemented in a B&B algorithm (typically: breadth-first search, depth-first algorithm, best-first algorithm)
- This directly impacts the **performance** of the algorithm in terms of computation time !

Breadth-First Search



Depth-First Search