Foundation Of Operations Research
Theory

Christian Rossi

Academic Year 2023-2024

Abstract

Operations Research is the branch of applied mathematics dealing with quantitative methods to analyze and solve complex real-world decision-making problems.

The course covers some fundamental concepts and methods of Operations Research pertaining to graph optimization, linear programming and integer linear programming.

The emphasis is on optimization models and efficient algorithms with a wide range of important applications in engineering and management.

Contents

Chapter 1

Introduction

1.1 Definition

Definition

Operations Research is the branch of mathematics in which mathematical models and quantitative methods are used to analyze complex decision-making problems and find near-optimal solutions.

It is an interdisciplinary field at the interface of applied mathematics, computer science, economics and industrial engineering.

1.2 Decision-making problems

Definition

The *decision-making problems* are problems in which we must choose a feasible solution among many alternatives based on one or several criteria.

The more complex decision-making problems are tackled via a mathematical modelling approach. Those problems can be classified in the following categories:

1. Assignment problem: given m jobs and m machines, suppose that each job can be executed by any machine and that t_{ij} is the execution time of job J_i one machine M_j. We want to decide which job assign to each machine to minimize the total execution time. Each job must be assigned to exactly one machine, and each machine to exactly one job. The number of feasible solution is equal to $m!$.

2. Network design: we want to decide how to connect n cities via a collection of possible links to minimize the total link cost. Given a graph

$G = (N, E)$ with a node $i \in N$ for each city and an edge $\{i, j\} \in E$ of cost $c_i j$, select a subset of edges of minimum total cost, guaranteeing that all pairs of nodes are connected. The number of feasible solution is equal to $2^{|E|}$.

3. Shortest path: given a direct graph that represents a road network with distances (traveling times) for each arc, determine the shortest path between two points (nodes).

4. Personnel scheduling: determine the week schedule for the hospital personnel, to minimize the number of people involved while meeting the daily requirements.

5. Service management: determine how many desks to open at a given time of the day so that the average customer waiting time does not exceed a certain value.

6. Multi-criteria problem: decide which laptop to buy considering the price, the weight and the performance.

7. Maximum clique (community detection in social networks): determine the complete sub-graph of a graph, with the maximum number of vertices.

1.3 History

In the World War II, teams of scientists were asked to do research on the most efficient way to conduct the operations. In the decades after the war, the techniques became public and began to be applied more widely to problems in business, industry and society. During the industrial boom, the substantial increase in the size of the companies and organizations gave rise to more complex decision-making problems thanks to fast progress in Operations Research and in numerical analysis methodologies and advent and diffusion of computers.

1.4 Operations Research workflow

The main steps in studying an Operations Research problem are:

1. Define the problem.
2. Build the model.
3. Select or develop an appropriate algorithm.
4. Implement it or use an existing program.

After all this process we need to analyze the results with feedbacks.

The model obtained with this process is a simplified representation of a real-world problem. To define it we must identify the fundamental elements of the problem and the main relationships among them.

Example : A company produces three types of electronic devices: D_1, D_2, D_3, going through three main phases of the production process: assembly, refinement and quality control. The time required for each phase and product is:

	D_1	D_2	D_3
Assembly	80	70	120
Refinement	70	90	20
Quality control	40	30	20

The available resources within the planning horizon in minutes are:

Assembly	**Refinement**	**Quality control**
30 000	25 000	18 000

The unary product for each product in:

D_1	D_2	D_3
1600	1000	2000

The main assumption is that the company can sell whatever it produces.

The mathematical model that describes the problem given before is the following:

- Decision variables: x_j is the number of devices D_j produced for $j = 1, 2, 3$.
- Objective function: we need to maximize the earning, so we have:

$$\max [1.6x_1 + 1x_2 + 2x_3]$$

- Constraints: they are on the production limit of each phase, that are:

$$80x_1 + 70x_2 + 120x_3 \leq 30000$$

$$70x_1 + 90x_2 + 20x_3 \leq 25000$$

$$40x_1 + 30x_2 + 20x_3 \leq 18000$$

- Variable type: the variables must be non-negative values, so we have $x_1, x_2, x_3 \geq 0$.

Example: An insurance company must decide which investments to select out of a given set of possible assets.

Investments	**Area**	**Capital** (c_j)	**Return** (r_j)
A (automotive)	Germany	150000	11%
B (automotive)	Italy	150000	9%
C (ICT)	USA	60000	13%
D (ICT)	Italy	100000	10%
E (real estate)	Italy	125000	8%
F (real estate)	France	100000	7%
G (treasury bonds)	Italy	50000	3%
H (treasury bonds)	UK	80000	5%

The available capital is 600000 euro. It is required to take at most five different investments. It is also required to take at maximum three investments in Italy and maximum three abroad.

The mathematical model that describes the problem given before is the following:

- Decision variables: boolean value to communicate if the investment is selected or not: $x_j = 1$ if the j-th investment is selected and $x_j = 0$ otherwise, for $j = 0, \ldots, 8$.
- Objective function: we need to maximize the expected return, so we have:

$$\max \left[\sum_{j=1}^{8} c_j r_j x_j \right]$$

- Constraints: there is a constraint on the capital that insurance

$$\sum_{j=1}^{8} c_j x_j \leq 800$$

There is a constraint also on the max number of general investment and on the region they are coming from formalized asked

$$\sum_{j=1}^{8} x_j \leq 5$$

$$x_2 + x_4 + x_5 + x_7 \leq 3$$

$$x_1 + x_3 + x_6 + x_8 \leq 3$$

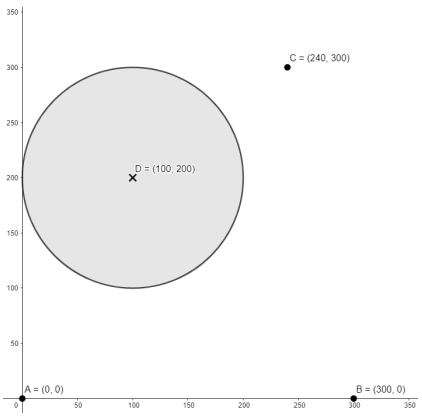
- Variable type: the variables are binary integer defined as $x_j \in \{0, 1\}$ $1 \leq j \leq 8$.

The variant requires that if any of the ICT investment is selected, then at least one of the treasury bond must be select. This requires one new constraint that is:

$$\frac{x_3 + x_4}{2} \leq x_7 + x_8$$

It is divided by two because if both ICT are selected at least one treasury bound must be selected and not two.

Example : Consider three oil pits, located in positions $A = (0, 0)$, $B = (300, 0)$, and $C = (240, 300)$, from which oil is extracted.



Connect them to a refinery with pipelines whose cost is proportional to the square of their length. The refinery must be at least 100 km away from point $D = (100, 200)$, but the oil pipelines can cross the corresponding forbidden zone. Give a mathematical model to decide where to locate the refinery to minimize the total pipeline cost.

- Decision variables: the coordinates of the refinery x_1, x_2.

- Objective function: we need to minimize the cost, so we have:

$$\begin{aligned}\min z = &\left[(x_1 - 0)^2 + (x_2 - 0)^2\right] \\ &+ \left[(x_1 - 300)^2 + (x_2 - 0)^2\right] \\ &+ \left[(x_1 - 240)^2 + (x_2 - 300)^2\right]\end{aligned}$$

- Constraints: there is a constraint on the location that is

$$\sqrt{(x_1 - 100)^2 + (x_2 - 100)^2} \geq 100$$

- Variable type: $x_1, x_2 \in \mathbb{R}$

1.5 Mathematical programming problem

	Decisions	**Objective**	**Uncertainty**
Mathematical programming	single	one	-
Multi-objective programming	single	multiple	-
Stochastic programming	-	-	✓
Game theory	multiple	-	-

We will analyze the mathematical programming problems. The optimization usually requires to minimizing or maximizing a given function. Note that maximizing $f(x)$ is the same problem of minimizing $-f(x)$. The problems are characterized by:

- Decision variables $x \in \mathbb{R}^n$: numerical variables that identify a solution.
- Feasible region $X \subseteq \mathbb{R}^n$.
- Objective function $f : X \to \mathbb{R}$: expresses in quantitative terms the value of each feasible solution.

Solving a mathematical programming problem consists in finding a feasible solution which is globally optimum. It may happen that: the problem is infeasible, unbounded, has a single optimal solution or has numerous optimal solutions. When the problem is very hard we must settle for a feasible solution that is a local optimum. An optimization problem can have many local optima. Mathematical programming can be classified in:

1. Linear Programming.
2. Integer Linear Programming.

3. Nonlinear Programming.

Multi-objective programming can be taken into account in different ways. Suppose we wish to minimize $f_1(x)$ and maximize $f_2(x)$, we can:

1. Turn it into a single objective problem by expressing the two objectives in terms of the same unit:

$$\min \lambda_1 f_1(x) - \lambda_2 f_2(x)$$

for appropriate scalars λ_1 and λ_2.

2. Optimize the primary objective function and turn the other objective into a constraint:

$$\max_{x \in X} f_2(x) \quad f_1(x) \leq \epsilon$$

for an appropriate constant ϵ.

Chapter 2

Algorithms

2.1 Complexity

Definition

An *algorithm* for a problem is a sequence of instructions that allows to solve any of its instances.

The execution time of an algorithm depends on the instance and on the computer. We want to evaluate the complexity of the algorithm as a function of the size of the instance independently of the hardware. Thus, we consider the number of elementary operations and assume all have the same cost. Since it is usually hard to determine the exact number of elementary operations, so we consider the asymptotic number of elementary operations in the worst case. We look for a function $f(n)$ which is asymptotically an upper bound on number of elementary operations needed to solve any instance of size at most n.

Definition

A function f is *ordered* of g, written $f(n) = O(g(n))$, if $\exists c > 0$ such that $f(n) \leq cg(n)$, for n sufficiently large.

Two classes of algorithms are distinguished according to their worst-case complexity:

- Polynomial: $O(n^d)$ for a given constant d.
- Exponential: $O(2^n)$.

Algorithms with a high order polynomial complexity are not efficient in practice.

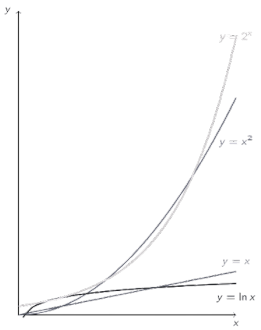


Figure 2.1: Plot of various algorithm's complexity

2.2 Definitions

Definition

An algorithm is *exact* if it provides an optimal solution for every instance.

Otherwise, it is *heuristic*.

Definition

A *greedy algorithm* constructs a feasible solution iteratively by making at each step a locally optimal choice, without reconsidering previous choices.

Chapter 3

Network optimization models

3.1 Introduction

Many decision-making problems can be formulated in terms of graphs.

Definition

A *graph* is a pair $G = (N, E)$, with a set of nodes N and a set of edges or arcs $E \subseteq N \times N$ connecting them pairwise.

An edge connecting the nodes i and j is represented by $\{i, j\}$ or (i, j) if the graph is *undirected* or *directed* respectively.

Example : A road network which connects n cities can be modelled by a graph where a city corresponds to a node, and a connection corresponds to an edge.

The graph on the left is undirected and defined as:

- $N = \{1, 2, 3, 4, 5\}$
- $E = \{\{1, 2\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\}$

The graph on the right is directed and defined as:

- $N = \{1, 2, 3, 4, 5\}$
- $E' = \{(1, 2), (1, 4), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5)\}$

Definition

Two nodes are *adjacent* if they are connected by an edge.

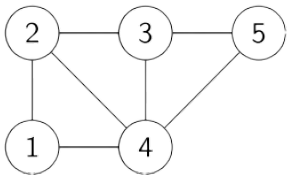
An edge e is *incident* in a node v if v is an endpoint of e.

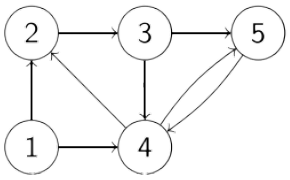
In undirected graphs, the *degree* of a node is the number of incident edges in a given node.

In directed graph, the *in-degree* (*out-degree*) of a node is the number of arcs that ave it as successor (predecessor).

Example: In the undirected graph, we have that nodes 1 and 2 are adjacent and 1 and 3 are not. The edge $\{1, 2\}$ is incident in nodes 1 and 2. Node 1 has a degree 2, and node 4 has a degree of 4.

In the directed graph, the node 1 has an in-degree equal to 0, and an out-degree equal to 2.





Definition

A *directed path from* $i \in N$ *to* $j \in N$ is a sequence of arcs $p = \langle\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}\rangle$ connecting nodes v_1 and v_k.

Nodes u and v are *connected* if there is a path connecting them. A graph (N, E) is connected if u, v are connected for any $u, v \in N$.

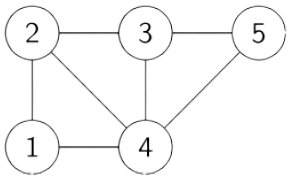
A graph is *strongly connected* if u, v are connected by a directed path for any $u, v \in N$.

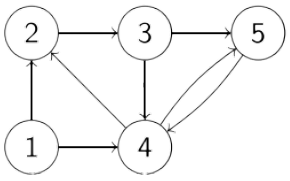
A *cycle* or circuit is a path with $v_1 = v_k$.

Example: The undirected graph has a path $\langle\{2, 3\}, \{3, 4\}, \{4, 5\}\rangle$ from node 2 to node 5. So we say those nodes are connected.

The directed graph has a directed path $\langle(3, 5), (5, 4), (4, 2), (2, 3), (3, 4)\rangle$ from node 3 to node 4. So we say those nodes are not strongly connected.

In the undirected graph $\langle\{2, 3\}, \{3, 5\}, \{5, 4\}, \{4, 2\}\rangle$ is a cycle. In the directed graph $\langle(2, 3), (3, 4), (4, 2)\rangle$ is a circuit.



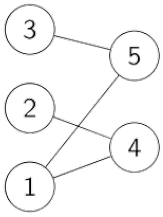


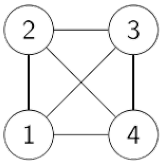
Definition

A graph is *bipartite* if there is a partition $N = N_1 \cup N_2$ with $N_1 \cap N_1 = \varnothing$ such that no edge connects nodes in the same subset.

A graph is *complete* if $E = \{\{v_i, v_j\} | v_i, v_j \in N \wedge i \leq j\}$

Example: The graphic on the left is bipartite because we can find two subsets of nodes such that $N = N_1 \cup N_2$ with $N_1 \cap N_1 = \varnothing$ that are: $N_1 = \{1, 2, 3\}$ and $N_2 = \{4, 5\}$. The graph on the right is a complete graph because all the nodes are connected with each other.





Definition

Given a directed graph $G = (N, A)$ and $S \subset NM$, the *outgoing cut* induced by S is:

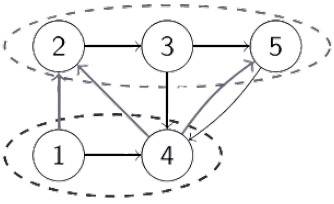
$$\delta^+(S) = \{(u, v) \in A | u \in S \wedge v \in N - S\}$$

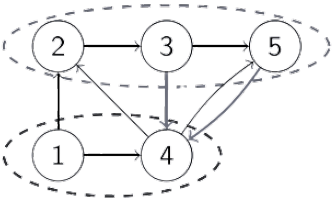
Given a directed graph $G = (N, A)$ and $S \subset NM$, the *incoming cut* induced by S is:

$$\delta^-(S) = \{(u, v) \in A | v \in S \wedge u \in N - S\}$$

Example: In the following graph we can note that:

- $\delta^+(\{1, 4\}) = \{(1, 2), (4, 2), (4, 5)\}$
- $\delta^-(\{1, 4\}) = \{(3, 4), (5, 4)\}$





An undirected graph with n nodes has at most $m = n(n-1)$ arcs. A directed graph with n nodes has at most $m = \dfrac{n(n-1)}{2}$ arcs.

Definition

Given m, the number of arcs or edges, and n, the number of nodes of the graph, we have that a graph is called *dense* if:

$$m \approx n^2$$

Given m, the number of arcs or edges, and n, the number of nodes of the graph, we have that a graph is called *sparse* if:

$$m \ll n^2$$

The best way to represent a dense graph is by using an $n \times n$ adjacency matrix, that is defined in the following way:

$$\begin{cases} a_{ij} = 1 & \text{if } (i,j) \in A \\ a_{ij} = 0 & \text{otherwise} \end{cases}$$

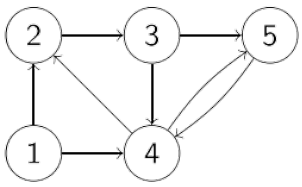
The best way to represent a sparse graph is by using lists of successors for each node.

Example: The adjacency matrix for the following graph is:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

And the list of successor is:

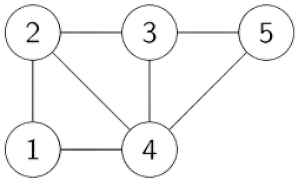
$$S(1) = \{2,4\} \quad S(2) = \{3\} \quad S(3) = \{4,5\} \quad S(4) = \{2,5\} \quad S(5) = \{4\}$$



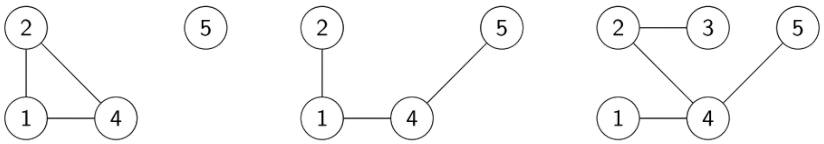
Definition

$G' = (N', E')$ is a *sub-graph* of $G = (N, E)$ if $N' \subseteq N$ and $E' \subseteq E$.
A *tree* $G_T = (N', T)$ *of* G is a connected and acyclic sub-graph of G.
$G_T = (N', T)$ is a *spanning tree of* G if it contains all nodes in G.
The *leaves* of a tree are the nodes of degree one.

Example : Given the following graph:



We can obtain: a sub-graph, a tree, and a spanning tree:



The trees have the following properties:

1. Every tree with n nodes has $n - 1$ edges.
2. Any pair of nodes in a tree is connected via a unique path.
3. By adding a new edge to a tree, we create a unique cycle.
4. Let $G_T = (N, T)$ be a spanning tree of $G = (N, E)$. Consider an edge $e \notin T$ and the unique cycle C of $T \cup \{e\}$. For each edge $f \in C - \{e\}$, the sub-graph $T \cup \{e\} - \{f\}$ is also a spanning tree of G.
5. Let F be a partial tree contained in an optimal tree of G. Consider $e\{u, v\} \in \delta(S)$ of minimum cost, then there exists a minimum cost spanning tree of G containing e.
6. If $c_j \geq 0$ for all $(i, j) \in A$, there is at least one shortest path which is simple.

Proof Property one: We will demonstrate this property with a proof by induction. For the base case we have that the claim holds for $n = 1$. For the inductive step we have to show that, if this is true for trees with n nodes, then it is also true for those with $n+1$ nodes. Let T_1 be a tree with $n+1$ nodes and recall that any tree with $n \geq 2$ nodes has at least two leaves. By deleting one of the leaf and its incident edge we obtain a tree T_2 with n nodes. By induction hypothesis, T_2 has $n-1$ edges. Therefore, the tree T_1 has $n-1+1=n$ edges. ■

Proof Property five: By contradiction, assume $T^* \subseteq E$ is a minimum cost spanning tree with $F \subseteq T^*$ and $e \notin T^*$. Adding edge e to T^* creates the cycle C. Let $f \in \delta(S) \cap C$:

- If $c_e = c_f$, then $T^* \cup \{e\} - \{f\}$ is also optimal since it has same cost of T^*.
- If $c_e < c_f$, then $c\,(T^* \cup \{e\} - \{f\}) < C(T^*)$, hence T^* is not optimal. ■

3.2 Graph reachability problem

Given the directed graph $G = (N, A)$ and a node s, determine all the nodes that are reachable from s.

Algorithm 1 Graph reachability problem

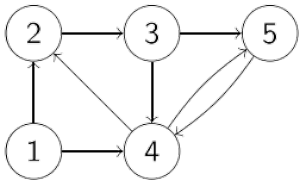
```
1: Q ← {s}
2: M ← {∅}
3: while Q ≠ 0 do
4:     u ← node in Q
5:     Q ← Q − {u}
6:     M ← M ∪ {u}
7:     for (u, v) ∈ δ+(u) do
8:         if v ∉ M and v ∉ Q then
9:             Q ← Q ∪ {v}
10:        end if
11:    end for
12: end while
```

The worst case complexity of the previous algorithm is $O(n^2)$.

Example: Given the following graph and $s = 2$ the algorithm makes the following steps:

1. $Q = \{2\}$ $M = \varnothing$
2. $Q = \{3\}$ $M = \{2\}$
3. $Q = \{4, 5\}$ $M = \{2, 3\}$
4. $Q = \{5\}$ $M = \{2, 3, 4\}$
5. $Q = \varnothing$ $M = \{2, 3, 4, 5\}$

So the nodes $\{2, 3, 4, 5\}$ are reachable from node two.



3.3 Minimum spanning tree problem

Given an undirected graph $G = (N, E)$ and a cost function, find a spanning tree $G_T = (N, T)$ of minimum total cost:

$$\min_{T \in X} \sum_{e \in T} c_e$$

where X is the set of all spanning trees of G.

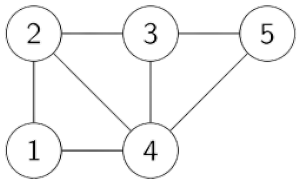
Theorem (*Cayley*)

A complete graph with n nodes ($n \geq 1$) has $n^{(n-2)}$ spanning trees.

To find the spanning tree with the minimum total cost we can use an algorithm that iteratively builds the spanning tree. The algorithm for the minimum spanning tree problem is the following:

1. Select any node arbitrarily, and connect it to the nearest distinct node.
2. Identify the unconnected node that is closest to a connected node, and then connect these two nodes. Repeat this step until all nodes have been connected.

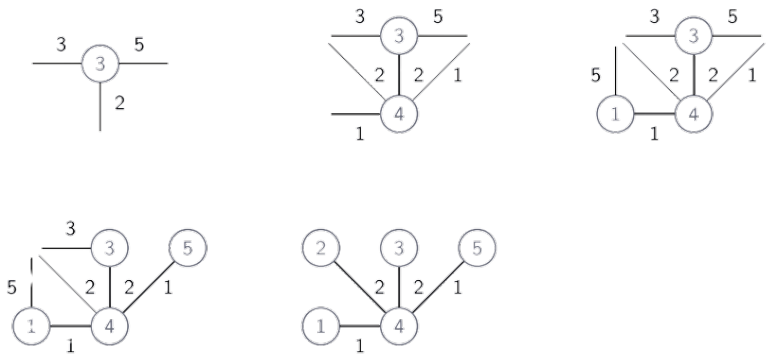
Example : Apply the Prim's algorithm to the following graph:



We select the node 3 as starting node, and so we have $S = \{3\}\, T = \{\varnothing\}$, then:

- The edge with minimum cost is the one that connects the nodes 3 and 4. Now we have: $S = \{3, 4\}$ and $T = \{\{3, 4\}\}$.
- The edge with minimum cost is the one that connects the nodes 1 and 4. Now we have: $S = \{1, 3, 4\}$ and $T = \{\{3, 4\}, \{1, 4\}\}$.
- The edge with minimum cost is the one that connects the nodes 4 and 5. Now we have: $S = \{1, 3, 4, 5\}$ and $T = \{\{3, 4\}, \{1, 4\}, \{4, 5\}\}$.
- The edge with minimum cost is the one that connects the nodes 4 and 5. Now we have: $S = N$ and $T = \{\{3, 4\}, \{1, 4\}, \{4, 5\}, \{2, 4\}\}$.

The total cost in this case is equal to $c(T) = 6$. Graphically, we have the following graphs:



Given a connected graph $G = (N, E)$ with edge cost the algorithm outputs $T \subseteq N$ of edges of G such that $G_T = (N, T)$ is a minimum cost spanning tree of G.

Algorithm 2 Prim's algorithm for the minimum cost spanning tree problem

1: $S \leftarrow \{u\}$
2: $T \leftarrow \{\varnothing\}$
3: **while** $|T| < n - 1$ **do**
4: $\quad \{u, v\} \leftarrow$ edge in$\delta(S)$of minimum cost
5: $\quad S \leftarrow S \cup \{v\}$
6: $\quad T \leftarrow T \cup \{u, v\}$
7: **end while**

where $u \in S$ and $v \in N - S$. The worst-case complexity is $O(n^2)$.

Proposition : Prim's algorithm is exact.

The exactness does not depend on the choice of the first node nor on the selected edge of minimum cost in $\delta(S)$.

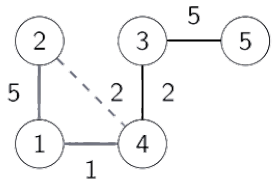
Proposition : Prim's algorithm is greedy.

At each step a minimum cost edge is selected among those in the cut $\delta(S)$ induced by the current set of nodes S.

Definition

Given a spanning tree T, an edge $e \notin T$ is *cost decreasing* if when e is added to T it creates a cycle C with $C \subseteq T \cup \{e\}$ and $\exists f \in C - \{e\}$ such that $c_e < c_f$.

Example : Given the following graph



$c_{24} = 2 < 5 = c_{12}$

Because $c(T \cup \{e\} - \{f\}) = c(T) + c_e - c_f$, if e is cost decreasing, then

$$c(T \cup \{e\} - \{f\}) < c(T)$$

Theorem (*Tree optimaility condition*)

A tree T is of minimum total cost if and only if no cost decreasing edge exists.

Proof of direct implication : If a cost-decreasing edge exists, then T is not of minimum total cost. ■

Proof of inverse implication : If no cost-decreasing edge exists, then T is of minimum total cost. Let T^* be a minimum cost spanning tree found by Prim's algorithm. It can be verified that, by exchanging one edge at a time, T^* can be iteratively transformed into T without modifying the total cost. Thus, T is also optimal. ■

The optimality condition allows us to verify whether a spanning tree T is optimal: it is sufficient to check that each $e \in E - T$ is not a cost-decreasing edge.

3.4 Graph shortest path problem

Given a directed graph $G = (N, A)$ with a cost $c_j \in R$ for each arc $(i, j) \in A$, and two nodes s and t, determine a minimum cost path from s to t.

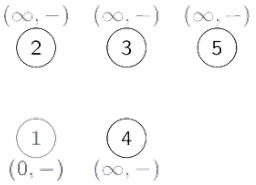
Each value c_j represents the cost of arc $(i, j) \in A$. Node s is the origin, or source, and t is the destination, or sink.

Definition

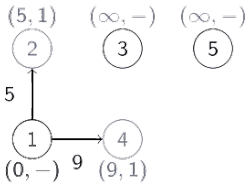
A path is *simple* if no node is visited more than once.

The input of Dijkstra's algorithm is a Graph $G = (N, A)$ with non-negative arc costs and $s \in N$. This algorithm will give us the shortest paths from s to all other nodes of G. The idea behind it is to consider the nodes in increasing order of cost of the shortest path from s to any one of the other nodes. To each node $j \in N$, we assign a label L_j which corresponds to the cost of a minimum cost path from s to j and a label $pred_j$ that is the predecessor of j in the shortest path from s to j. Note that this algorithm is greedy with respect to path from s to j.

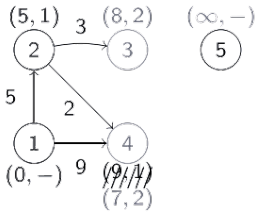
Example : Given a graph, and selecting 1 as the initial node, we set the following labels.



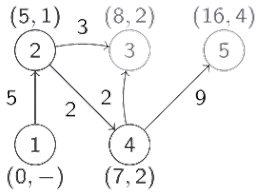
Next, we check all the arcs going from the starting node to other nodes, and we update the nodes label (the node one will always have no predecessor and null cost).



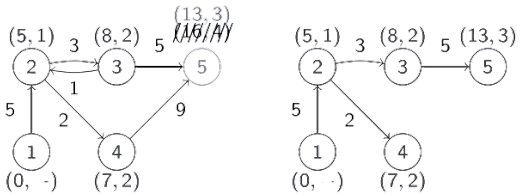
Now, we move to the node 2 and check all the reachable nodes. In this case we found the shortest path from the initial node to 4, so we update the label of 4.



Now, we move to the node 4 that is the closest node to s, and we check all the arcs going in other nodes.



We do the same for the remaining nodes, again in increasing order of cost from the start.



It is now possible to retrieve backward the shortest path to any node of the graph using the predecessor label. For instance, we have that the shortest path from s to 5 has a cost of 13 and the path is $(1, 2, 3, 5)$.

Algorithm 3 Dijkstra's algorithm for the graph shortest path problem

1: $S \leftarrow \varnothing$
2: $X \leftarrow \{s\}$
3: **for** $u \in N$ **do**
4: $\quad L_u \leftarrow +\infty$
5: **end for**
6: $L_s \leftarrow 0$
7: **while** $|S| \neq n$ **do**
8: $\quad u \leftarrow \text{argmin}\{L_i | i \in X\}$
9: $\quad X \leftarrow X - \{u\}$
10: $\quad S \leftarrow S \cup \{u\}$
11: $\quad$ **for** $(u, v) \in \delta^+(u)$ such that $L_v > L_u + c_{uv}$ **do**
12: $\quad\quad L_v \leftarrow L_u + c_{uv}$
13: $\quad\quad pred_v \leftarrow u$
14: $\quad\quad X \leftarrow X \cup \{v\}$
15: $\quad$ **end for**
16: **end while**

The worst case complexity of this algorithm is $O(n^3)$.

Proposition : Dijkstra's algorithm is exact.

Proof : At the k-th step: $S = \{s, i_2, \ldots, i_k\}$ and

$$L_j = \begin{cases} \text{cost of a minimum path from } s \text{ to } j,\ j \in S \\ \text{cost of a minimum path with all intermediate nodes in } S,\ j \notin S \end{cases}$$

By induction on the number k of steps:

- Base case: it is easy to see that the statement holds for $k = 1$, since

$$S = \{s\},\ L_s = 0,\ L_j = +\infty,\ \forall j \neq S$$

- Inductive step: we must prove that, if the statement holds at the k-th step, it must also hold for the $(k+1)$-th step.

In the $(k+1)$-th step let $u \notin S$be the node that is inserted in S and $\varnothing$ the path from s to u such that:

$$L_v + c_{uv} \leq L_i + C_{ij},\ \forall (i, j) \in \delta^+(S)$$

Let us verify that every path π from s to u has $c(\pi) \geq c(\varnothing)$. There exist $i \in S$ and $j \notin S$ such that:

$$\pi = \pi_1 \cup \{(i, j)\} \cup \pi_2$$

Where (i, j) is the first arc in $\pi \cap \delta^+(S)$. Moreover:

$$c(\pi) = c(\pi_1) + c_{ij} + c(\pi_2) \geq L_i + c_{ij}$$

Because $c_{ij} \geq 0$, thus, $c(\pi_2) \geq 0$, and by the induction assumption, $c(\pi_1) \geq L_i$. Finally, by the choice of (v, u) we have:

$$L_i + c_{ij} \geq L_v + c_{vu} = c(\varnothing)$$

■

We can note that:

- A set of shortest paths from s to all the nodes j can be retrieved via the vector of predecessors.
- The union of a set of shortest paths from node s to all the other nodes of G is a shortest path trees rooted at s. Such shortest path trees have nothing to do with minimum cost spanning trees.
- Dijkstra's algorithm does not work when there are arcs with negative cost.