Foundations of Operations Research - Notes - v0.3.1

260236

October 2024

Preface

Every theory section in these notes has been taken from the sources:

- Course slides. [2]

About:

GitHub repository

These notes are an unofficial resource and shouldn't replace the course material or any other book on foundations of operations research. It is not made for commercial purposes. I've made the following notes to help me improve my knowledge and maybe it can be helpful for everyone.

As I have highlighted, a student should choose the teacher's material or a book on the topic. These notes can only be a helpful material.

Contents

Algorithms

1 Introduction

Definition 1

Operations Research (OR), often shortened to the initialism `OR`, is the branch of mathematics in which **mathematical models** and **quantitative methods** (e.g. optimization, game theory, simulation) are **used to analyze complex decision-making problems** and **find (near-)optimal solutions**.

The overall and primary *goal* is to *help make better decisions*.

OR can be seen as an interdisciplinary field at the intersection of applied mathematics, computer science, economics, and industrial engineering.

Operations research is often concerned with **determining the extreme values of some real-world objective**: the *maximum* (of profit, performance, or yield) or *minimum* (of loss, risk, or cost). Originating in military efforts before World War II, its techniques have grown to concern problems in a variety of industries. [3]

Its origins date back to World War II: teams of scientists were asked to research the most efficient way to conduct operations (e.g., to optimize the allocation of scarce resources).

In the decades after the war, the techniques became public and were applied more widely to problems in business, industry, and society.

During the industrial boom, the substantial increase in the size of firms and organizations led to more complex decision problems.

There are favorable circumstances: rapid progress in OR and in numerical analysis methods, and the advent and spread of computers (available computing power and widespread software).

1.1 Decision-making problems

Decision-making problems are analyzed using mathematical models and quantitative methods.

Definition 2

Decision-making problems are problems in which we must **choose** a (feasible) **solution among a large number of alternatives based on one or several criteria**.

In other words, they are complex decision-making problems that are **addressed through a mathematical modeling approach** (mathematical models, algorithms, and computer implementations).

Some practical **examples** include assignment problem, network design, shortest paths, personnel scheduling, service management, multicriteria problem, and maximum clique.

1.1.1 Assignment problem

A mathematical definition of an **assignment problem** is as follows. 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 on machine M_j.

	M_1	M_2	M_3
J_1	2	6	3
J_2	8	4	9
J_3	5	7	8

Table 1: Example of an assignment problem table.

The **main goal** is to **decide which job to assign to each machine in order to minimize the total execution time**. Also, (constraints) each job must be assigned to exactly one machine, and each machine must be assigned to exactly one job.

The **number** of feasible **solutions** is the permutations, then the **factorial of** m: $m!$.

1.1.2 Network design

The **network design problem** is characterized as **how to connect** n **cities** (offices) **via a collection of possible links** so as (*main goal*) **to minimize the total link cost**.

Using mathematical terms, 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_{ij}, select a subset of edges of minimum total cost, guaranteeing that all pairs of nodes are connected.

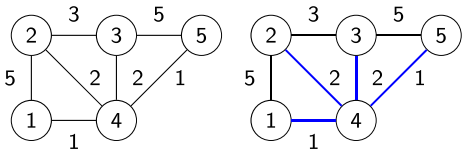


Figure 1: Examples of network design graphs.

The **number** of alternative **solutions** is at most 2^m, where $m = |E|$.

1.1.3 Shortest path

The **shortest path problem** is similar to network design. Given a direct path that represents a road network with distances (traveling times) for each arc, determine the shortest (fastest) path between two points (nodes).

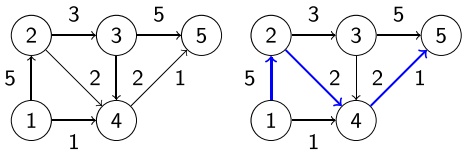


Figure 2: Examples of shortest path graphs.

1.1.4 Other problems

Other decision-making problems are:

- **Personnel scheduling problem**: determine the week schedule for the hospital personnel, so as to minimize the number of people involved (physicians, nurses, ...) while meeting the daily requirements.
- **Service management problem**: determine how many counters/desks to open at a given time of the day so that the average customer waiting time does not exceed a certain value (guarantee a given service quality).
- **Multicriteria problem**: decide which laptop to buy considering the price, the weight and the performance.
- **Maximum clique problem**: determine the complete subgraph of a graph, with maximum number of vertices.

1.2 Scheme of an OR study

The most important and common **steps** in operational research are:

1. **Problem**. Define the problem;
2. **Model**. Build the model;
3. **Algorithm**. Select or develop an appropriate algorithm;
4. **Implementation**. Implementing or using an efficient computer program;
5. **Results**. Analyze the results.

Definition 3

A mathematical **model** is a **simplified representation of a real-world problem**.

To define a mathematical model, it is necessary to identify the fundamental elements of the problem and the main relationships between them. But **how can we decide** the *number of decision makers*, the *number of objectives* and the *level of uncertainty in the parameters*? It depends on the environment. If we have:

- One decision maker, one object, then we will use **mathematical programming** (section 1.3, page 12).
- One decision maker, multiple objectives, then we will use **multi-objective programming** (section 1.4, page 15).
- Uncertainty greater than zero, then we will use **stochastic programming**.
- Multiple decision makers, then we will use **game theory**.

Example 1: production planning

A company produces 3 types of electronic devices: D_1, D_2, D_3; going through 3 main phases of the production process: assembly, refinement and quality control.
Time (in minutes) required for each phase and product:

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

Available resources within the planning horizon (depend on the workforce) in minutes:

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

Unitary profit (in KEuro):

D_1	D_2	D_3
1.6	1	2

Assumption: the company can sell whatever it produces.
Give a mathematical model for determining a production plan which maximizes the total profit.

- **Decision variables**, x_j is equal to the number of devices D_j produced, for $j = 1, 2, 3$.
- **Objective function**: max $z = 1.6x_1 + 1x_2 + 2x_3$.
- **Constraints**: the production capacity limit for each phase:

$$\begin{aligned} 80x_1 + 70x_2 + 120x_3 &\leq 30'000 \quad \text{(assembly)} \\ 70x_1 + 90x_2 + 20x_3 &\leq 25'000 \quad \text{(refinement)} \\ 40x_1 + 30x_2 + 20x_3 &\leq 18'000 \quad \text{(quality control)} \end{aligned}$$

- **Non-negative variables**: $x_1, x_2, x_3 \geq 0$ may be fractional (real) values.

Example 2: portfolio selection problem

An insurance company must decide which investments to select out of a given set of possible assets (stocks, bonds, options, gold certificates, real estate, ...).

Investments	area	capital (c_j K€)	expected return (r_j)
A	Germany	150	11%
B	Italy	150	9%
C	U.S.A.	60	13%
D	Italy	100	10%
E	Italy	125	8%
F	France	100	7%
G	Italy	50	3%
H	UK	80	5%

Legend:

- A and B: automotive
- C and D: ICT
- E and F: real estate
- G: short term treasury bounds
- H: long term treasury bounds

The available capital is: 600 KEuro.
At most 5 investments to avoid excessive fragmentation.
Geographic diversification to limit risk: ≤ 3 investments in Italy and ≤ 3 abroad.
Give a mathematical model for deciding which investments to select so as to maximize the expected return while satisfying the constraints.

- **Decision variables**, x_j is equal to 1 if j-th investment is selected and $x_j = 0$ otherwise, for $j = 1, \ldots, 8$.
- **Objective function**: $\max\ z = \sum_{j=1}^{8} c_j\, r_j\, x_j$.
- **Constraints**:

$$\begin{aligned}
\sum_{j=1}^{8} c_j\, x_j &\leq 600 && \text{(capital)}\\
\sum_{j=1}^{8} x_j &\leq 5 && \text{(max 5 investments)}\\
x_2 + x_4 + x_5 + x_7 &\leq 3 && \text{(max 3 in Italy)}\\
x_1 + x_3 + x_6 + x_8 &\leq 3 && \text{(max 3 abroad)}
\end{aligned}$$

- **Binary (integer) variables**: $x_j \in \{0, 1\}$ and $1 \leq j \leq 8$.

Possible variant. In order to limit the risk, if any of the ICT investment is selected then at least one of the treasury bond must be selected.

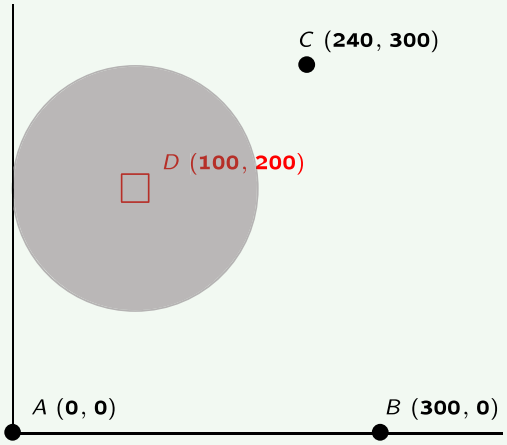
- **Objective function**: $\max z = \sum_{j=1}^{8} c_j\, r_j\, x_j$.
- **Constraints**:

$$\begin{aligned}
\sum_{j=1}^{8} c_j\, x_j &\leq 600 && \text{(capital)}\\
\sum_{j=1}^{8} x_j &\leq 5 && \text{(max 5 investments)}\\
x_2 + x_4 + x_5 + x_7 &\leq 3 && \text{(max 3 in Italy)}\\
x_1 + x_3 + x_6 + x_8 &\leq 3 && \text{(max 3 abroad)}\\
\frac{x_3 + x_4}{2} &\leq x_7 + x_8 && \text{(investment in treasury bonds)}
\end{aligned}$$

- **Binary (integer) variables**: $x_j \in \{0, 1\}$ and $1 \leq j \leq 8$.

Example 3: facility location

Consider 3 oil pits, located in positions A, B and C, 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 so as to minimize the total pipeline cost.

- **Decision variables**, x_1, x_2 cartesian coordinates of the refinery.
- **Objective function**:

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

- **Constraints**:

$$\sqrt{(x_1 - 100)^2 + (x_2 - 200)^2} \geq 100$$

- **Variables**: $x_1, x_2 \in \mathbb{R}$.

1.3 Mathematical programming/optimization

Mathematical Optimization or **Mathematical Programming** is the **selection of a best element**, with regard to some criteria, **from some set of available alternatives**.

In the more general approach, an optimization problem consists of **maximizing** or **minimizing a real function** by systematically choosing input values from within an allowed set and computing the value of the function. The generalization of optimization theory and techniques to other formulations constitutes a large area of applied mathematics.

Okay, how is it defined mathematically?

Mathematical Operation problems belong to the category of decision-making problems. They are characterized by a **single decision maker**, a **single objective**, and **reliable parameter estimates**. In mathematical language, we can say:

$$\text{opt } f(\mathbf{x}) \quad \text{with} \quad \mathbf{x} \in X \quad \text{and} \quad \text{opt } = \begin{Bmatrix} \min \\ \max \end{Bmatrix}$$

Where:

- $\mathbf{x} \in \mathbb{R}^n$ **decision variables**. They are numerical variables whose values identify a solution of the problem.
- $X \subseteq \mathbb{R}^n$ **feasible region**. Distinguishes between feasible and infeasible solutions (via constraints):

$$X = \left\{ \mathbf{x} \in \mathbb{R}^n \ : \ g_i(\mathbf{x}) \begin{Bmatrix} = \\ \leq \\ \geq \end{Bmatrix} 0, i = 1, \ldots, m \right\}$$

- $f : X \to \mathbb{R}$ **objective function**.Expresses in quantitative terms the value or cost of each feasible solution.

Note an interesting observation:

$$\max \{ f(\mathbf{x}) : \ \mathbf{x} \in X \} = -\min \{ -f(\mathbf{x}) : \ \mathbf{x} \in X \}$$

More specifically, how can we solve these problems?

It depends on how hard the given problem is to solve.

- The problem has an **easy/medium level** of complexity. It makes sense to use the **Global Optima** technique. It consists in finding a feasible solution that is **globally optimum**, then a vector $\mathbf{x}^* \in X$ such that:

$$\begin{aligned} f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in X \quad \text{if opt } = \min \\ f(\mathbf{x}^*) \geq f(\mathbf{x}) \quad \forall \mathbf{x} \in X \quad \text{if opt } = \max \end{aligned}$$

  Unfortunately, this method is not perfect and it may happen that the given problem occurs:

- Is **infeasible**, so the feasible region is empty: $X = \emptyset$.
- Is **unbounded**: $\forall c \in \mathbb{R}$, $\exists \mathbf{x}_c \in X$ such that $f(\mathbf{x}_c) \leq c$ or $f(\mathbf{x}_c) \geq c$.
- Has a **single optimal solution**.
- Has a **large number** (even an infinite number) **of optimal solutions** (with the same optimal value!).

- The problem has a **difficult**/**hard level** of complexity. Then the **Local Optima** is the best choice. It consists in finding a feasible solution that is **local optimum** (main different against global optima technique), then a vector $\widehat{\mathbf{x}} \in X$ such that:

$$f(\widehat{\mathbf{x}}) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \text{ with } \mathbf{x} \in X \text{ and } ||\mathbf{x} - \widehat{\mathbf{x}}|| \leq \varepsilon \quad \text{if opt } = \min$$
$$f(\widehat{\mathbf{x}}) \geq f(\mathbf{x}) \quad \forall \mathbf{x} \text{ with } \mathbf{x} \in X \text{ and } ||\mathbf{x} - \widehat{\mathbf{x}}|| \leq \varepsilon \quad \text{if opt } = \max$$

  For an appropriate value $\varepsilon > 0$.

  In this case, it may happen that the given **problem has many local optima**.

Categories

A Mathematical Programming can be **categorized** depending on the feasible region:

- **Linear Programming (LP)**. The function f is linear:

$$X = \left\{ \mathbf{x} \in \mathbb{R}^n \ : \ g_i(\mathbf{x}) \begin{Bmatrix} = \\ \leq \\ \geq \end{Bmatrix} 0, \ i = 1, \ldots, m \right\} \text{ with } g_i \text{ linear } \forall i$$

  An **example** is the *production planning*.

- **Integer Linear Programming (ILP)**. The function f is linear:

$$X = \left\{ \mathbf{x} \in \mathbb{R}^n \ : \ g_i(\mathbf{x}) \begin{Bmatrix} = \\ \leq \\ \geq \end{Bmatrix} 0, \ i = 1, \ldots, m \right\} \cap \mathbb{Z}^n \text{ with } g_i \text{ linear } \forall i$$

  An **example** is the *portfolio selection* (finance). As we can see, the ILP technique is identical to LP with additional integrality constraints on the variables.

- **Nonlinear Programming (NLP)**. The function f is convex/regular or non convex/regular:

$$X = \left\{ \mathbf{x} \in \mathbb{R}^n \ : \ g_i(\mathbf{x}) \begin{Bmatrix} = \\ \leq \\ \geq \end{Bmatrix} 0, \ i = 1, \ldots, m \right\}$$

  With g_i convex/regular or not convex/regular $\forall i$.

  An **example** is the *facility location* (with g_i convex).

History of Mathematical Programming

It is correct to report the history of mathematical programming:

1826/27 Joseph Fourier presents a method to solve systems of linear inequalities (Fourier-Motzkin) and discusses some LPs with 2-3 variables.

1939 Leonid Kantorovitch lays the bases of LP (Nobel prize 1975).

1947 George Dantzig proposes independently LP and invents the simplex algorithm.

1958 Ralph Gomory proposes a cutting plane method for ILP problems.

1.4 Multi-objective programming

How is it born?

Even though some real-word problems can be reduced to a matter of a single objective very often it is hard to define all the aspects in terms of a single objective. Defining multiple objectives often gives a better idea of the task.

Multi-objective programming (also known as **Multi-objective optimization** or **Pareto optimization**) is an **area of multiple-criteria decision making** that is concerned with mathematical optimization problems involving **more than one objective function to be optimized simultaneously**. Multi-objective is a type of vector optimization that has been applied in many fields of science, including engineering, economics and logistics where optimal decisions need to be taken in the presence of trade-offs between two or more conflicting objectives.

Minimizing cost while maximizing comfort while buying a car, and maximizing performance whilst minimizing fuel consumption and emission of pollutants of a vehicle are **examples** of multi-objective optimization problems involving two and three objectives, respectively. [1]

Suppose to minimize $f_1(\mathbf{x})$ and maximize $f_2(\mathbf{x})$ (e.g. laptop: f_1 is cost and $f_2(\mathbf{x})$ is performance):

1. Turn it into a **single objective problem** by expressing the two objectives in terms of the same unit (e.g. monetary unit):

   $$\min \lambda_1 f_1(\mathbf{x}) - \lambda_2 f_2(\mathbf{x})$$

   for appropriate scalars λ_1 and λ_2.

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

   $$\max_{x \in \tilde{X}} f_2(\mathbf{x}) \quad \text{where} \quad \tilde{X} = \{\mathbf{x} \in X \ : \ f_1(\mathbf{x}) \leq \varepsilon\}$$

   for appropriate constant ε.

This is a simple introduction, the more detailed explanation will be explained in the following pages.

1.5 Mathematical Programming or Simulation?

Mathematical Programming and Simulation involves **building a model** and **designing an algorithm**. And the main differences are:

Mathematical Programming	**Simulation**
Problem can be "well" formalized.	Problem is difficult to formalize.
Algorithm yields a(n optimal) solution.	Algorithm simulates the evolution of the real system and allows to evaluate the performance of alternative solutions.
The results are "certain"	The results need to be interpreted.
Example: assignment.	Example: service counters.

Table 2: Major differences between Mathematical Programming and Simulation.

2 Graph and network optimization

2.1 Graphs

2.1.1 Definitions and characteristics

In the following section we will explain some basic concepts about the graphs. The terms used are important, and we will use the definition box to highlight these words.

Definition 1: graph

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

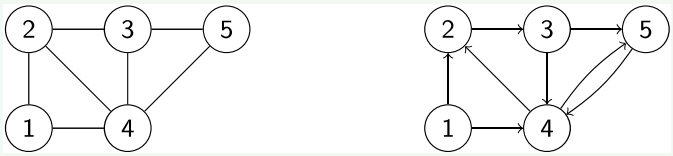
Definition 2: edge

An **edge** connecting the nodes i and j is represented by

- Graph undirected: $\{i, j\}$
- Graph directed: (i, j)

Example 1

For **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.



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

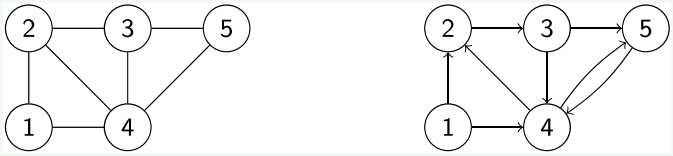
Some graph properties are:

- 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 other words, in a graph G, two edges are incident **if they share a common vertex**. For example, edge $E_1 = (v_1, v_2)$ and edge (v_1, v_3) are incident as they share the same vertex v_1.
- The degree concept depends on the type of graph:
  - Undirected graph: the **degree** of a node is the **number of incident edges**.
  - Directed graph: the **in-degree** and **out-degree** of a node is the **number of arcs that have it as succesor** and **predecessor**.

Example 2: adjacent, incident, degree, in-degree and out-degree

Given the graphs:



- Undirected graph:
  - Nodes 1 and 2 are **adjacent** (unlike nodes 1 and 3).
  - Edge $\{1, 2\}$ is **incident** in nodes 1 and 2.
  - Node 1 has **degree** 2, node 4 has **degree** 4.
- Directed graph: node 1 has **in-degree** 0, and **out-degree** 2.

Other useful features include:

- A **(directed) path from** $i \in N$ **to** $j \in N$ is a sequence of (arcs) edges:

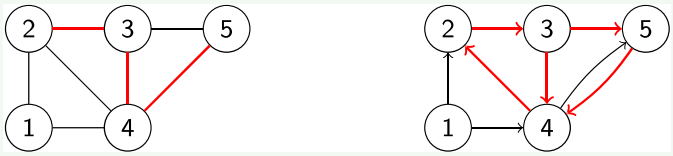
$$p = \langle \{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\} \rangle$$

  Connecting nodes v_1 and v_k, with $\{v_i, v_{i+1}\} \in E$, for $i = 0, \ldots, k-1$.
- A generic **node** u and v are **connected** if there is a path connecting them.
- A **graph** (N, E) is **connected** if two generic nodes u, v are connected, $\forall u, v \in N$. Recall that in generic graph notation, the variable N represents a set of nodes or vertices and E represents a set of edges or arcs connecting them in pairs.

- A **graph** (N, E) is **strongly connected** if two generic nodes u, v are connected by a directed path, $\forall u, v \in N$ (for any node in the set of nodes or vertices of the graph).

Example 3: directed path, connected nodes, connected graph, strongly connected

Given the graphs:



- Undirected graph:
  - $\langle \{2,3\}, \{3,4\}, \{4,5\} \rangle$ is a **path** from node 2 to node 5.
  - **Nodes** 2 and 5 are **connected**.
  - It is a **connected graph**.
- Directed graph:
  - $\langle \{3,5\}, \{5,4\}, \{4,2\}, \{2,3\}, \{3,4\} \rangle$ is a **directed path** from node 3 to node 4.
  - It is not a **strongly connected graph** because the node 1 cannot be the destination of none path. In other words, doesn't exist a directed path from node u to node 1 (where u is a generic node, $\forall u \in N \setminus \{1\}$).

Finally, there are other interesting properties and notations about graphs and edges:

- A **cycle (circuit)** is a directed path with $v_1 = v_k$ (source and destination are the same).
- A **graph** is **bipartite** if there is a partition $N = N_1 \cup N_2$ $(N_1 \cap N_2 = \emptyset)$ such that no edge connects nodes in the same subset.
- A **graph** is **complete** if $E = \{\{v_j, v_j\} \ : \ v_i, v_j \in N \ \wedge \ i \leq j\}$.
- Given a directed graph $G = (N, A)$ and $S \subset N$, the **outgoing cut** induced by S is:

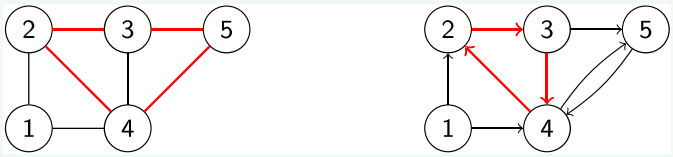
$$\delta^+(S) = \{(u, v) \in A \ : \ u \in S \wedge : v \in N \subseteq S\}$$

  The **incoming cut** induced by S is:

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

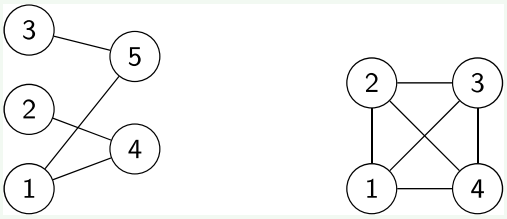
Example 4: cycle/circuit in graph, bipartite graph, complete graph, out/incoming cut

An example of cycle in graph:



- Undirected graph: $\langle \{2,3\}, \{3,5\}, \{5,4\}, \{4,2\} \rangle$ is a cycle.
- Directed graph: $\langle (2,3), (3,4), (4,2) \rangle$ is a circuit.

An example of bipartite/complete graph:

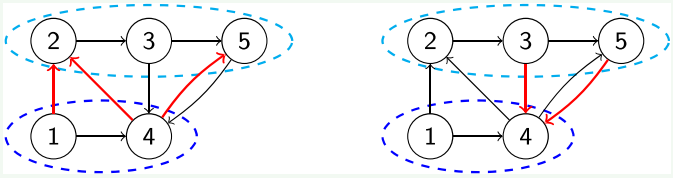


- To the left a **bipartite graph**, because:

$$N_1 = \{1,2,3\} \qquad N_2 = \{4,5\}$$

- And to the right a **complete graph**.

Finally, an example of out/incoming cut:



- Left graph:

$$\begin{aligned} \delta^+(\{1,4\}) &= (\{1,2\}, \{4,2\}, \{4,5\}) \\ S &= \{1,4\} \\ N \setminus S &= \{2,3,5\} \end{aligned}$$

- Right graph:

$$\begin{aligned} \delta^-(\{1,4\}) &= (\{3,4\}, \{5,4\}) \\ S &= \{1,4\} \\ N \setminus S &= \{2,3,5\} \end{aligned}$$

2.1.2 Graphical representation

Such a matrix can easily be represented as a graph. This guarantees that it can be stored efficiently in a computer. But to understand how to do this in general, it's important to understand some other important properties:

- For any graph G with n nodes, the **number of edges** satisfies:
  - $m \leq \frac{n\,(n-1)}{2}$ if G is undirected.
  - $m \leq n\,(n-1)$ if G is directed.
- A graph is **dense** if $m \approx n^2$ and **sparse** if $m \ll n^2$. Where m is the number of arcs and n the number of nodes.
- For dense directed graphs, exist an **adjacency matrix** $A_{n\times n}$:

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

To build the adjacency matrix it is necessary to create a **list of successors** for each node. In other words, for **each node** we need to write the **outgoing edges** and write the matrix.

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad \begin{aligned} S\,(1) &= \{2,4\} \\ S\,(2) &= \{3\} \\ S\,(3) &= \{4,5\} \\ S\,(4) &= \{2,5\} \\ S\,(5) &= \{4\} \end{aligned}$$

Each row represents a node, and we set the value 1 if the column index is a node that has the arc of the row node as its incoming edge. So row one (node one) has the value one in column two (node two) and column four (node four).

2.1.3 Graph reachability problem

In general the **graph reachability problem** can be formulated as follows.

Definition 3: graph reachability problem

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

Where N is the set of nodes and A is the set of edges.

The problem takes:

- As **input** a ***graph*** $G = (N, A)$ described by the successor lists and node $s \in N$.
- As **output** produces a ***subset*** $M \subseteq N$ ***of nodes*** of the graph G reachable from s.

Our goal is to devise an efficient algorithm that allows us to find all nodes reachable from s.

2.1.3.1 Description and algorithm

Definition 4: Breadth-First Search

Breadth-First Search (BFS) is an **algorithm for searching a tree data structure for a node that satisfies a given property**. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

```
1  Q ← {s};
2  M ← ∅;
3  while Q ≠ ∅ do:
4      u ← node in Q;
5      Q ← Q \ {u};
6      // label u
7      M ← M ∪ {u}
8      for (u, v) ∈ δ+ (u) do:
9          if v ∉ M and v ∉ Q:
10             Q ← Q ∪ {v}
```

Algorithm 1: Graph reachability problem: Breadth-First Search

Rows 1-2. Declare a queue `Q` containing the nodes reachable from the source `s` and **not yet processed**. It is managed as a FIFO (First-In First-Out) queue. By definition, we add the `s` node at the beginning because it is our entry point.

Then we declare the set `M`. It represents the subset of nodes of the graph that are reachable from the source `s`. Obviously, it is empty at the beginning of the algorithm.

Row 3. The BFS algorithm continues to process the nodes until the queue is empty. As long as there is an element, it continues.

Rows 4-5. Take a node from the queue `Q` and assign it to the variable `u`. Also remove the element `u` from the set `Q`. In other words, perform a difference operation between the sets `Q` and the set composed only of the element `u` (`Q` $\setminus$ {`u`}).

For example, in Python we can get the same result using the `popleft` method of the `deque` data structure.

Row 7. Using the union between sets, add the visited node `u` to the subset `M` of reachable nodes. This operation is also called "labeling" because you are *labeling* a node as *visited*.

Row 8. Iterate each tuple (node `u` just popped from the queue, node `v` adjacent to node `u`) in the outgoing cut set of node `u`.

Rows 9-10. If the adjacent node `v` is not in the reachable set `M` and it is not in the queue (so it is not waiting to be evaluated), add the adjacent node `v` to `Q` using the union set operation.

As we said, the algorithm continues until the queue is not empty. Note that the queue is updated each time a neighboring node is found that is not already in the solution set (`M`).

2.1.3.2 Complexity of algorithm

BFS Algorithm - Time Complexity

The BFS time complexity[1] can be expressed as $O(|N| + |A|)$, since **every node and every edge will be explored in the worst case**.

- $|N|$ is the number of **nodes**;
- $|A|$ is the number of **edges** in the graph.

Note that $O(|A|)$ may vary between $O(1)$ and $O(N^2)$, depending on how sparse the input graph is. For example, for **dense graphs**, exactly $O(N^2)$.

BFS Algorithm - Space Complexity

When the number of nodes (or vertices) in the graph is known ahead of time, and additional data structures are used to determine which vertices have already been added to the queue, the space complexity[2] can be expressed as $O(|N|)$, where $|N|$ is the number of vertices. This is in addition to the space required for the graph itself, which may vary depending on the graph representation used by an implementation of the algorithm.

In other words, the algorithm needs:

- The **space to store** the set N, i.e. the **set of all nodes** in the graph.
- The **space to store the graph itself** depends on the implementation used.

[1] In theoretical computer science, the time complexity is the computational complexity that describes the amount of computer time it takes to run an algorithm. Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, supposing that each elementary operation takes a fixed amount of time to perform. Thus, the amount of time taken and the number of elementary operations performed by the algorithm are taken to be related by a constant factor. (source)

[2] The space complexity of an algorithm or a data structure is the amount of memory space required to solve an instance of the computational problem as a function of characteristics of the input. It is the memory required by an algorithm until it executes completely. This includes the memory space used by its inputs, called input space, and any other (auxiliary) memory it uses during execution, which is called auxiliary space. (source)

References

[1] A. Abraham, L.C. Jain, and R. Goldberg. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Advanced Information and Knowledge Processing. Springer, 2005.

[2] Braz Pascoal Marta Margarida. Foundations of Operations Research. Slides from the HPC-E master's degree course on Politecnico di Milano, 2024.

[3] Wikipedia. Operations research - Wikipedia. https://en.wikipedia.org/wiki/Operations_research. [Accessed 08-09-2024].

Index