Course program: “Operations research and logistics” (9 e.c.t.s.) 2026/2027

Module 1 (6 ects):

- Introduction: Mathematical Programming, Linear Programming.
- Models: models of (Integer) Linear Programming.
- Essentials of Linear Programming: geometry of Linear Programming (vertices and basic solutions), Simplex method; Duality in Linear Programming: Dual Problem, fundamental properties, economic interpretation.
- Essentials of Integer Linear Programming: Unimodularity, Branch and Bound method.

Solution of (Integer) Linear Programming via Excel Solver.

- Five specific problems (with specific solution methods):

- Minimum Cost Path Problem: Djikstra algorithm;
- Project Planning Problem: PERT method;
- Maximum Flow Problem: Ford and Fulkerson algorithm;
- Production Planning Problem: Wagner and Whitin method;
- Plant Location Problem: Local Search algorithms.

Module 2 (3 ects):

- Introduction: recalls from Module 1.
- Some examples: vehicle loading; balancing vehicles loading; involving the minimum number of vehicles; distribution network (re-definition); service network (re-definition); supply (re-definition); a trip in highway.
- Minimum Spanning Tree Problem - MST.
- Transportation Problem: two generalizations.
- Traveling Salesman Problem - TSP.
- Vehicle Routing Problem - VRP.
- Simulations via Excel Solver.

Recommended reference books:

- R. Baldacci, M. Dell’Amico, *Fondamenti di Ricerca Operativa*, Pitagora Ed. Bologna (2002).
- M. Fischetti, *Lezioni di Ricerca Operativa*, Ed. Libreria Progetto Padova (1999, 2014)
- S. Martello, M.G. Speranza, *Ricerca Operativa per l’Economia e per l’Impresa*, Società Editrice Esculapio (2012).
- A. Sassano, *Modelli e Algoritmi della Ricerca Operativa*, Ed. Franco Angeli (1999).

Written material: the written material for the exam consists of the “course pack” (see below).

Exam format: written exam, optional oral exam.

Office hours: see https://www.dec.unich.it/home-mosca-raffaele-168 (email: r.mosca@unich.it)

1. Introduction

Operations Research is a discipline that – in a nutshell – deals with modeling and solving optimization problems. It is a relatively recent discipline, formalized during World War II, although its fundamental concepts had essentially been introduced prior to that time.

One way to describe the utility of Operations Research is through the following scheme:

optimization problem (informal) $\rightarrow$ mathematical model (formal) $\rightarrow$ solution

which summarizes the process as follows:

• first step: starting from a given optimization problem (in informal terms), one arrives – if possible – at a mathematical model of it (in formal terms); this step relies on human skill and experience;

• second step: starting from the aforementioned mathematical model, one arrives – if possible – at its solution, that is, the calculation of an "optimal solution" for the given optimization problem; this step is achieved using specialized software.

Specifically: an *optimization problem* is understood as a problem in which a decision-maker must choose, from a set of feasible decisions/solutions, a decision/solution that optimizes a certain utility; a *mathematical model* is understood, in accordance with the book [Fischetti], as a Mathematical Programming problem.

A ***Mathematical Programming*** problem consists of computing

$$\textbf{min}\ \{f(x) : x \in X\}$$

where:

$X \subseteq \mathbb{R}^n$ ($n \in \mathbb{N}$) it is a set called the *set of feasible solutions* and

$f : X \rightarrow \mathbb{R}$ is a function called the objective function.

Note: **min** can be replaced with **max** by changing the sign of the function.

Solving a Mathematical Programming problem means calculating an *optimal solution* to the problem, that is, a solution $x^* \in X$ such that $f(x^*) \leq f(x)$ for any $x \in X$.

A Mathematical Programming problem is called a ***Linear Programming*** problem if:
X is the set of solutions to a system of linear equations and/or inequalities;
f is a linear function.

Example:

$$\begin{aligned} \max \quad & 12x_1 + 7x_2 \\ & 3x_1 + 2x_2 \leq 20 \\ & 5x_1 + 3x_2 \leq 50 \\ & x_1, x_2 \geq 0 \end{aligned}$$

So, in this example: $n = 2$, which means there are 2 (real) variables, that is, $X \subseteq \mathbb{R}^2$; specifically:
$X = \{(x_1, x_2) \in \mathbb{R}^2 : 3x_1 + 2x_2 \leq 20 \; ; \; 5x_1 + 3x_2 \leq 50 \; ; \; x_1 \geq 0 \; ; \; x_2 \geq 0\}$, and
$f : X \to \mathbb{R}$ is defined as $f(x_1, x_2) = 12x_1 + 7x_2$, i.e., it assigns the value $12x_1 + 7x_2$ to each $(x_1, x_2) \in X$.

A Mathematical Programming problem is called an ***Integer Linear Programming*** problem if:
X is the set of "integer" solutions to a system of linear equations and/or inequalities;
f is a linear function.

Example:

$$\begin{aligned} \max \quad & 12x_1 + 7x_2 \\ & 3x_1 + 2x_2 \leq 20 \\ & 5x_1 + 3x_2 \leq 50 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \text{ integers} \end{aligned}$$

In other words, an Integer Linear Programming problem can be viewed as a variant of a Linear Programming problem involving the addition of "integrality constraints" on the variables – or on only some of them, in which case it is referred to as Mixed-Integer Linear Programming.

Although (Integer) Linear Programming represents a fundamental or partial aspect of Mathematical Programming, it is useful for modeling a wide range of real-world optimization problems.

We conclude this introduction by presenting several real-world optimization problems below. As we will see in detail later, these can be modeled as (Integer) Linear Programming problems using the first step of the scheme and subsequently solved using the second step of the scheme. Upon reviewing them, we can observe that in real life we already address such problems using common-sense criteria, arriving at feasible solutions that are presumably good, even if not necessarily optimal; at the same time, we can see that the essence of these problems may appear in other contexts where finding an optimal [or at least as good as possible] solution is significant.

A technical preliminary note for the sake of completeness.

Recall that a *graph* $G = (V, E)$ is a pair of sets, the set V of *vertices* and the set E of *edges*, where $E \subseteq V \times V$ (that is, E is a subset of pairs of elements from V). A graph is typically drawn by representing: each vertex i as a point, each possible edge, that is each possible pair (i, j) of vertices, as a line connecting point i and point j. Finally, a distinction is made between a *directed graph*, in which every edge is directed (like a one-way street), and an *undirected graph*, in which every edge is undirected (like a two-way street); unless otherwise specified, the graph is generally assumed to be undirected.□

• Academic credits (e.c.t.s.)

A person has a total of H hours available in the upcoming semester, to prepare for certain exams, belonging to a set $E = \{1, \ldots, n\}$. Each exam i (where $i = 1, \ldots, n$) requires an estimated preparation time of h_i. Conversely, each exam i (where $i = 1, \ldots, n$) awards a specific number of academic credits c_i. The person decides not to prepare for an exam partially: that is, either to prepare for it thoroughly (investing the full estimated time required) or not to prepare for it at all. The problem is to select which exams to prepare for, subject to the aforementioned time constraint, so as to maximize the total number of credits from the selected exams.

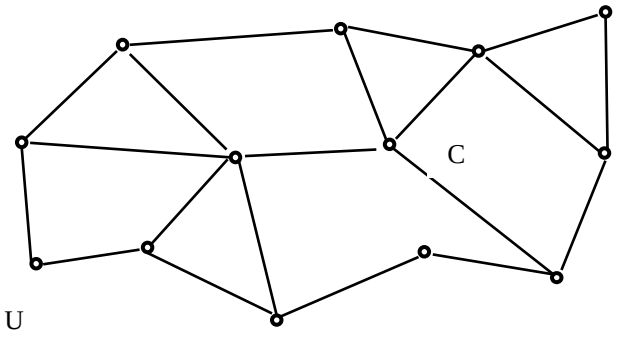
Comment: A problem of this type – as we shall see – can be modeled by defining a "binary" decision variable x_i for each exam i, where x_i takes the value 1 if exam i is selected and 0 otherwise.

• The optimal route

Every morning, a person travels from home to the university. The problem is to choose a route from home to the university (there may be more than one) that minimizes travel time.

Comment: A problem of this type can be visualized using a graph; each edge represents a stretch of road, while each vertex represents a specific location, such as an intersection (specifically, referring to the graph drawn below, "home" is vertex C and the "university" is vertex U). Furthermore, a number indicating the time (in minutes) required to traverse that edge (i.e., the corresponding stretch of road) can be associated with each edge. Consequently – as we shall see – such a problem can be modeled by defining a "binary" decision variable x_{ij} for each edge denoted by (i, j) (where i and j are

the vertices defining the edge); this variable takes the value 1 if the edge is part of the minimum-time path and 0 otherwise.



• Weekly physical activity

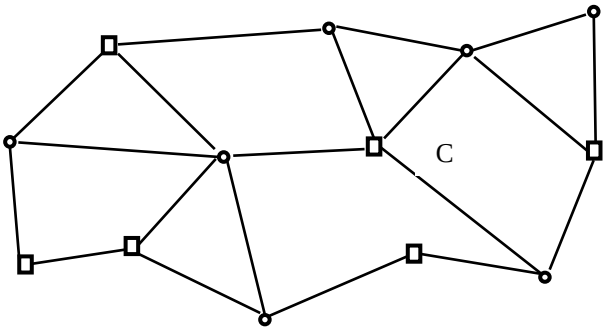
A person wishes to engage in physical activity, subject to the constraint of burning at least a certain number of calories, with the goal of minimizing the total time spent on these activities. The person can choose from a set $A = \{1, \ldots, n\}$ of physical activities. Each activity i (where $i = 1, \ldots, n$) can be performed for a maximum of h_i hours (in one week). Furthermore, each activity i burns b_i calories per hour. The person sets a requirement to burn at least B calories (in one week). The problem is to decide how many hours to allocate to each activity (in one week), while meeting the minimum calorie-burning target of B, so as to minimize the total time spent on physical activity.

Comment: A problem of this type – as we shall see – can be modeled by defining a "binary" decision variable x_i for each activity i, representing the number of hours dedicated to that activity.

• Errands

One morning, a person needs to run a series of errands; that is, the person must leave home, visit several locations, and return home. The challenge lies in selecting a sequence of these locations that allows the errands to be completed in the minimum amount of time, in other words, finding a route (there may be more than one) that minimizes total travel time.

Comment: A problem of this type can be visualized using a graph; each edge represents a stretch of road, while each vertex represents a specific location, such as an intersection (specifically, in the graph drawn below, the home is vertex C, and the errand locations are depicted as squares). Furthermore, a number indicating the time (in minutes) required to traverse that edge (i.e., the corresponding stretch of road) can be associated with each edge. At this point, such a problem can be modeled by initially defining a "binary" decision variable, x_{ij}, for each edge denoted by (i, j) (where i and j are the vertices defining the edge); this variable takes the value 1 if the edge is part of the minimum-time route and 0 otherwise.



• Shopping bags

A person is at a supermarket checkout and needs to pack the purchased items, identified by a set A, into two bags. Each item i (where $i \in A$) has a specific weight, a_i. The problem is to distribute the items between the two bags (assuming each bag is capable of holding all the items) so as to balance the weight between them.

Comment: A problem of this type – as we shall see – can be modeled by defining a "binary" decision variable x_i for each item i; this variable takes the value 1 if item i is placed in the first bag and the value 0 if it is placed in the second bag.

2. Examples of (Integer) Linear Programming models

In this chapter, we will look at many examples of the first step of the process for cases where the mathematical model is a (Integer) Linear Programming problem.

To carry out the first step of the process, one generally proceeds by:
(i) defining the problem's decision variables;
(ii) consequently defining the objective function and the constraints (i.e., the linear equations and/or inequalities and any integrality constraints) of the problem.

A. Basic examples

The following seven examples are presented as reference points (akin to constellations), bearing in mind that the initial step of the model relies on human skill and experience.

A1. *Resource composition*

A company manufactures two types of products, A and B, by combining three resources: R, S, and T. Specifically:

producing 1 unit of A requires: 2 units of R, 3 units of S, and 1 unit of T;

producing 1 unit of B requires: 0 units of R, 1 unit of S, and 3 units of T.

– *maximize revenue, given available resources*

There are 100, 70, and 80 units of R, S, T, respectively. The unit revenue from the sale of A and B is 15 and 20, respectively. The problem is to determine the quantities of A and B to produce in order to maximize total revenue.

Decision variables: x_A, x_B (units of A and B to be produced, respectively).

Model:	max	$15\,x_A + 20\,x_B$		
		$2\,x_A$	≤ 100	(availability R)
		$3\,x_A + 1\,x_B$	≤ 70	(availability S)
		$1\,x_A + 3\,x_B$	≤ 80	(availability T)
		$x_A, x_B \geq 0$		
		x_A, x_B integers		(if necessary)

– *minimize costs, given a specific production output*

It is necessary to produce 50 units of A and 70 units of B, respectively. The unit cost of resources R, S, T is 2, 1, 3, respectively. The problem is to determine the quantities of R, S, T to purchase in order to minimize the total cost. This problem is solved directly through multiplication (the quantities are: $2 \cdot 50 + 0 \cdot 70$; $3 \cdot 50 + 1 \cdot 70$; $1 \cdot 50 + 3 \cdot 70$); in other words, it has a “forced” solution, no modeling is required.

A2. *Resource decomposition*

A company produces two types of products, A and B, by processing three resources: R, S, and T. Specifically:

1 unit of R yields: 2 units of A, 3 units of B;
1 unit of S yields: 7 units of A, 2 units of B;
1 unit of T yields: 4 units of A, 0 units of B.

– *maximizing revenue, given available resources*

There are 40, 80, and 50 units of R, S, and T available, respectively. The unit revenue from the sale of A and B is 30 25, respectively. The problem is to determine the quantities of A and B to produce in order to maximize total revenue.

This problem is solved directly via multiplication (the quantities are: $2 \cdot 40 + 7 \cdot 80 + 4 \cdot 50$; $3 \cdot 40 + 2 \cdot 80 + 0 \cdot 50$); that is, it has a "forced" solution, no modeling is required.

– *minimizing costs, given a specific production outputs*

It is necessary to produce 80 and 100 units of A and B, respectively. The unit cost of resources R, S, T is 20, 40, 30, respectively. The problem is to determine the quantities of R, S, and T to purchase in order to minimize total cost.

Decision variables: x_R, x_S, x_T (units of R, S, and T to be purchased, respectively).

Model: min $20\,x_R + 40\,x_S + 30\,x_T$

$2\,x_R + 7\,x_S + 4\,x_T \geq 80$ (production di A)

$3\,x_R + 2\,x_S + 0\,x_T \geq 100$ (production di B)

$x_R, x_S, x_T \geq 0$

x_R, x_S, x_T integers (if necessary)

A3. *Assignment*

There are n jobs to be performed using n machines. Specifically: each machine performs exactly one job, and each job is performed by exactly one machine. Performing job i (for $i = 1, \ldots, n$) on machine j (for $j = 1, \ldots, m$) incurs a cost c_{ij}. The problem is to assign one job to each machine so as to minimize the total cost.

Decision variables: x_{ij} for $i = 1, \ldots, n$ and $j = 1, \ldots, n$ [variable : $x_{ij} \leftrightarrow$ arc (i, j) of the graph];

x_{ij} takes the value 1 if job i is assigned to machine j;

x_{ij} takes the value 0 if job i is not assigned to machine j.

Model: min $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$

$\sum_{i=1}^{n} x_{ij} = 1$ for $j = 1, \ldots, n$

(each machine performs exactly one job)

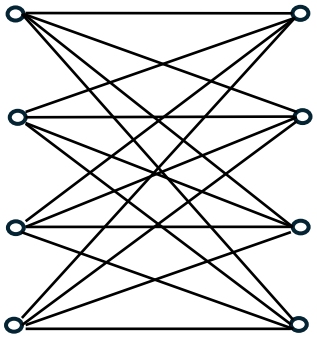
$\sum_{j=1}^{n} x_{ij} = 1$ for $i = 1, \ldots, n$

(each job is performed by exactly one machine)

$x_{ij} \geq 0$ for $i = 1, \ldots, n$ e $j = 1, \ldots, n$

$x_{ij} \leq 1$ for $i = 1, \ldots, n$ e $j = 1, \ldots, n$

x_{ij} integer for $i = 1, \ldots, n$ e $j = 1, \ldots, n$

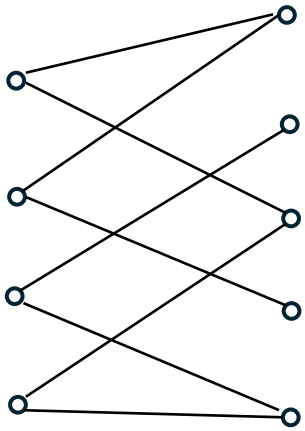


JOBS MACHINE

A4. *Matching*

A set N of n women and a set M of m men visit a matchmaking agency. Through interviews and questionnaires, the agency determines that woman i (for $i = 1, \ldots, n$) is presumably compatible with a certain subset M_i of M, and consequently, that man j (for $j = 1, \ldots,$ m) is compatible with a certain subset N_j of N. The problem is to form the largest possible number of pairs.

Let F be the set of *compatible* pairs (i, j), that is, $F = \{(i, j) : i \in N_j\} = \{(i, j) : j \in M_i\}$.



WOMEN MEN

Decision variables: x_{ij} for each pair $(i, j) \in F$ [variable $x_{ij} \leftrightarrow$ arc (i, j) of the graph];

x_{ij} takes the value 1 if woman i is paired with man j;

x_{ij} takes the value 0 if woman i is not paired with man j.

Model: max $\sum_{(i,j)\in F} x_{ij}$

$\sum_{(i,j)\in F} x_{ij} \leq 1$ for $i = 1, \ldots, n$

(each woman is paired with at most one man)

$\sum_{(i,j)\in F} x_{ij} \leq 1$ per $j = 1, \ldots, m$

(each man is paired with at most one woman)

$x_{ij} \geq 0$ for any $(i, j) \in F$

$x_{ij} \leq 1$ for any $(i, j) \in F$

x_{ij} integer for any $(i, j) \in F$

A5. *Packing*

A company has a budget b available to fund a selection of projects from a set of n projects. Each project i (where $i = 1, \ldots, n$) has a cost a_i. Conversely, each project i (where $i = 1, \ldots, n$) yields a profit p_i after one year. Partial funding of a project is not possible. The problem is to select which projects to fund in order to maximize the total profit after one year.

Decision variables: x_i for $i = 1, \ldots, n$;

x_i takes the value 1 if project i is selected;

x_i takes the value 0 if project i is not selected.

Model: max $p_1x_1 + p_2x_2 + \ldots + p_nx_n$

$a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b$

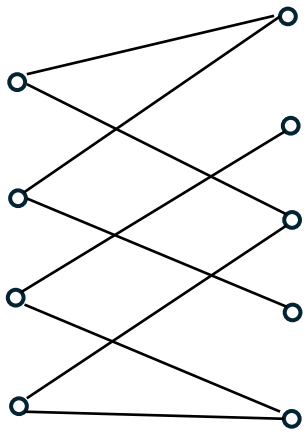
$x_i \geq 0$ for $i = 1, \ldots, n$

$x_i \leq 1$ for $i = 1, \ldots, n$

x_i integer for $i = 1, \ldots, n$

A6. *Covering*

A language school teaches m languages. The school wishes to engage external interpreters, selected from a set N of n interpreters, each of whom is qualified for only certain languages: for $j = 1, \ldots, m$, let N_j be the subset of N consisting of interpreters qualified for language j. The school requires that at least one qualified interpreter be engaged for each of the languages taught. The problem is to engage the minimum possible number of interpreters.



INTERPRETERS LANGUAGES

Decision variables: x_i for $i = 1, \ldots, n$

[variable $x_i \leftrightarrow$ vertex i in the graph's set of INTERPRETERS];

x_i will have the value 1 if interpreter i is selected;

x_i will have the value 0 if interpreter i is not selected.

Model: min $\sum_{i=1}^{n} x_i$

$\sum_{i \in N_j} x_i \geq 1$ for $j = 1, \ldots, m$

(for each language, there is at least one interpreter)

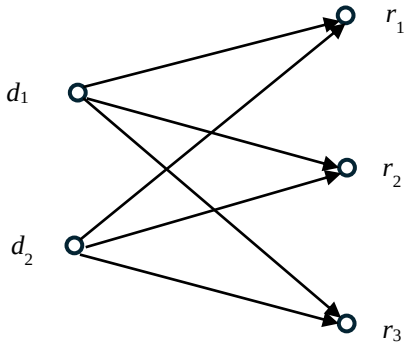
$x_i \geq 0$ for $i = 1, \ldots, n$

$x_i \leq 1$ for $i = 1, \ldots, n$

x_i integer for $i = 1, \ldots, n$

A7. *Transportation*

There are s source locations and t destination locations. Each source location $i \in \{1, \ldots, s\}$ has a supply of $d_i \geq 0$ units of a certain type of good, and each destination $j \in \{1, \ldots, t\}$ requires at least $r_j \geq 0$ units of the same good. Furthermore, for each pair (i, j), for $i \in \{1, \ldots, s\}$ and $j \in \{1, \ldots, t\}$, a unit transportation cost c_{ij} and a maximum transportable quantity q_{ij} are specified. The problem is to plan the transportation so as to minimize the total cost.



costi c_{ij}

SOURCES DESTINATIONS

Decision variables: x_{ij} for $i = 1, \ldots, s$ and $j = 1, \ldots, t$ [variable $x_{ij} \leftrightarrow$ arc (i, j) of the graph];
x_{ij} indicates the quantity transported from source i to destination j.

Model: min $\sum_{i=1}^{s}\sum_{j=1}^{t} c_{ij}x_{ij}$

$\sum_{j=1}^{t} x_{ij} \le d_i$ for $i = 1, \ldots, s$ (availability constraints)

$\sum_{i=1}^{s} x_{ij} \ge r_j$ for $j = 1, \ldots, t$ (request constraints)

$x_{ij} \le q_{ij}$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$ (capacity constraints)

$x_{ij} \ge 0$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

x_{ij} integer for $i = 1, \ldots, s$ and $j = 1, \ldots, t$ (if necessary)

B. Basic examples – Exercises

Formulate each of the following seven problems as (Integer) Linear Programming problems. Each of these problems can be reduced to one of the seven basic examples introduced earlier.

B1. A freelancer has 10 hours available each week to dedicate to expanding their business. Specifically, there is an opportunity to provide consulting services to five new clients: A, B, C, D, and E. Each client requires a certain amount of weekly working time, as follows: $t_A = 4$, $t_B = 5$, $t_C = 4$, $t_D = 3$, $t_E = 1$. Consequently, the freelancer cannot provide consulting services to all of them. Depending on the type of consulting involved, the freelancer can charge a weekly fee generating revenue as follows: $p_A = 70$, $p_B = 30$, $p_C = 70$, $p_D = 40$, $p_E = 30$. The problem is to select which clients to serve in order to maximize total revenue.

B2. A company decides to undertake four activities. Four existing agencies are available for this purpose. Due to various constraints, each agency can handle at most one of these activities, and each activity can be handled by at most one of these agencies. Finally, it is estimated beforehand that assigning activity j (where $j = 1, \ldots, 4$) to agency i (where $i = 1, \ldots, 4$) yields a profit of p_{ij}. The problem is to assign an activity to each agency so as to maximize total profit.

Variant: Under the assumptions stated above, suppose that one of the agencies must be closed for certain reasons. The problem is to choose which activities to undertake (given that only three can now be undertaken) so as to maximize total profit.

B3. A person wishes to follow a diet. Specifically, they need to consume two types of nutrients, namely, proteins and vitamins, which can be obtained by purchasing three types of food: fruit, milk, and eggs. In details:

1 unit of fruit contains: 0 units of protein, 7 units of vitamins;

1 unit of milk contains: 2 units of protein, 3 units of vitamins;

1 unit of eggs contains: 5 units of protein, 1 unit of vitamins.

The unit costs for fruit, milk, and eggs are 10, 20, and 10, respectively.

The diet requires the consumption of at least 15 units of protein and 25 units of vitamins.

The problem is to determine the quantities of fruit, milk, and eggs to purchase, satisfying the aforementioned requirements, so as to minimize the total cost.

B4. A winery wishes to produce two types of wine: a table wine and a dessert wine. The profit the company derives from producing one unit of table wine is 3, while the profit from producing one unit of dessert wine is 7. Production requires a specific combination of two types of grapes: let us call them type A and type B, respectively. Producing one unit of table wine requires 3 units of type A grapes and 2 units of type B grapes. Producing one unit of dessert wine requires 1 unit of type A grapes and 4 units of type B grapes. Finally, the company has 1000 units of type A grapes and 400 units of type B grapes available. The problem is to determine the quantities of table wine and dessert wine to produce in order to maximize total profit.

B5. A company dealing in sea salt has two warehouses, A and B, supplying four of its stores. The salt availability at each warehouse is $d_A = 190$ and $d_B = 370$ units, respectively. The salt demand from each store is $r_1 = 130$, $r_2 = 140$, $r_3 = 100$, $r_4 = 70$ units, respectively. The unit transport cost c_{ij} from warehouse i to store j is: $c_{A1} = 2$, $c_{A2} = 3$, $c_{A3} = 3$, $c_{A4} = 1$, $c_{B1} = 3$, $c_{B2} = 2$, $c_{B3} = 1$, $c_{B4} = 2$. (Optional: for logistical reasons, warehouse A cannot supply more than 100 units to any single store, while warehouse B cannot supply more than 150 units to any single store). The problem is to plan the supply to the stores so as to minimize the total cost.

B6. An agency wishes to open branches in a specific region comprising five provinces, labeled 1, 2, 3, 4, and 5. For each province $i \in \{1, \ldots, 5\}$, there is an option to open a branch F_i in the provincial capital. Based on the characteristics of these potential branches and the territory, it is estimated that:
potential branch F_1 would serve only provinces 1 and 3;
potential branch F_2 would serve only provinces 2 and 3;
potential branch F_3 would serve only provinces 3 and 4;
potential branch F_4 would serve only provinces 1, 2, and 4;
potential branch F_5 would serve only provinces 2, 3, and 5.
The problem is to determine which branches to open, ensuring that every province is served, so as to minimize the total number of branches opened (i.e., to open as few as possible).
Variant: assume that opening branch F_i entails a cost c_i ($i = 1, \ldots, 5$); determine which branches to open, ensuring that every province is served, so as to minimize the total cost.

B7. Five clients visit a travel agency. The agency offers trips to a set of three locations. Each client wishes to go on vacation to only a specific subset of these locations: client 1 only to location 3; client 2 only to locations 1 and 3; client 3 only to location 1; client 4 only to locations 1 and 3; client 5 only

to locations 1 and 2. Only one spot is available for each location. The goal is to accommodate the greatest possible number of clients.

Variant 1: Based on the assumptions above, suppose that for each location j, only b_j spots are available: specifically, there are 4 spots for location 1, 2 spots for location 2, and 2 spots for location 3. The goal is to accommodate the greatest possible number of clients.

Variant 2: Based on the assumptions above, suppose that sending a client to a location j yields a profit p_j for the agency: specifically, location 1 yields a profit of 150, location 2 yields a profit of 250, and location 3 yields a profit of 100. An alternative problem to solve is the following: maximize the agency's profit.

C. Five specific problems

Presented below, in terms of (integer) linear programming, are the models for five specific problems that will be revisited in the last part of Module 1.

C1. *Minimum-Cost Path*

Let $G = (V, A)$ be a directed graph, and let s and t be two vertices of G. A cost $q_{ij} \geq 0$ is associated with each arc $(i, j) \in A$. The cost of a path from *s* to *t* is the sum of the costs of the arcs forming the path. The problem is to determine a minimum-cost path from *s* to *t*.

Solution

Decision variables: x_{ij} for each arc $(i, j) \in A$;

x_{ij} takes the value 1 if the arc (i, j) is part of the path;

x_{ij} takes the value 0 if the arc (i, j) is not part of the path.

Model: min $\sum_{(i,j)\in A} q_{ij} x_{ij}$

$$\sum_{(s,j)\in A} x_{sj} = 1$$

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = 0 \quad \text{for any vertex } i \neq s, t \text{ di } G$$

$x_{ij} \geq 0$ for any $(i, j) \in A$

$x_{ij} \leq 1$ for any $(i, j) \in A$

x_{ij} integer for any $(i, j) \in A$

C2. *Project Planning*

A project consists of a set of *n* activities A_i (for $i \in \{1, \ldots, n\}$), each with a known duration greater than zero; in particular, precedence relationships of the form $A_i < A_j$ are specified between certain activities, indicating that the completion time of activity A_i must precede the start time of activity A_j. The problem is to schedule the activities so as to minimize the project completion time.

Solution

Decision variables: t_i^- e t_i^+, for $i = 1, \ldots, n$;

t_i^- represents the time at which activity A_i starts;

t_i^+ represents the time at which activity A_j finishes;

an auxiliary variable y is also introduced, such that $y = \max\{t_i^+ : i = 1, \ldots, n\}$.

Model: min y

$t_i^+ \geq t_i^- + d_i$ for $i = 1, \ldots, n$

$t_i^+ \leq t_j^-$ for each precedence relationship $A_i < A_j$

$t_i^- \geq 0$ for $i = 1, \ldots, n$

$t_i^+ \leq y$ for $i = 1, \ldots, n$

C3. *Maximum Flow*

Let $G = (V, A)$ be a directed graph, and let s and t be two vertices of G. A capacity $k_{ij} \geq 0$ is associated with each arc $(i, j) \in A$. A *flow* from s to t is an assignment of values $x_{ij} \geq 0$ for each arc $(i, j) \in A$ such that:

:: $\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = 0$ for every vertex $i \neq s, t$ of G (i.e., for every vertex $i \neq s, t$ of G, the sum of the incoming flow equals the sum of the outgoing flow);

:: $x_{ij} \leq k_{ij}$ for every arc $(i, j) \in A$.

The *value* of a flow from s to t is $\sum_{(s,j)\in A} x_{sj}$.

The problem is to determine a flow from s to t with maximum value.

Solution

Decision variables: x_{ij} for each arc $(i, j) \in A$;

x_{ij} represents the flow assigned to arc (i, j).

Model: max $\sum_{(s,j)\in A} x_{sj}$

$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = 0$ for any vertex $i \neq s, t$ of G

$x_{ij} \leq b_{ij}$ for any $(i, j) \in A$

$x_{ij} \geq 0$ for any $(i, j) \in A$

x_{ij} integer for any $(i, j) \in A$ (if necessary)

C4. *Production Planning*

A company must produce a good to meet a 4-month sales plan, which specifies the sales targets for the end of each month. Production capacity varies from month to month, as do the unit production cost and the unit inventory storage cost (for stock held in the company's warehouse at the end of each month). There is no initial inventory, and no inventory is desired at the end of the 4-month period. Details are as follows:

month	sales target	prod. capacity	prod. cost	storage cost
1	20 units	40 units	34	2
2	30 units	50 units	36	3
3	50 units	30 units	32	2
4	40 units	50 units	38	

The problem is to determine the quantity of the good to produce each month, while meeting the sales plan, so as to minimize the total cost.

Solution

Decision variables: x_i for i = 1, 2, 3, 4;

x_i represents the quantity of product to be produced in month i.

Model: min $34x_1 + 36x_2 + 32x_3 + 38x_4 + 2(x_1 - 20) + 3(x_1 + x_2 - 50) +$

$+ 2(x_1 + x_2 + x_3 - 100)$

$x_1 \leq 40$

$x_2 \leq 50$

$x_3 \leq 30$

$x_4 \leq 50$

$x_1 \geq 20$

$x_1 + x_2 \geq 50$

$x_1 + x_2 + x_3 \geq 100$

$x_1 + x_2 + x_3 + x_4 = 140$

$x_i \geq 0$ for $i = 1, 2, 3, 4$

x_i integer for $i = 1, 2, 3, 4$ (if necessary)

C5. *Facility Location*

A certain number of facilities must be activated, chosen from among m potential facilities, to supply n customers. Activating potential facility j (where $j \in \{1, \ldots, m\}$) incurs a cost $f_j \geq 0$. Each facility is assumed to have unlimited supply capacity. Each customer will be connected to exactly one facility; specifically, connecting customer i to facility j incurs a cost c_{ij} (where $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$). The problem is to select which facilities to activate and to connect each customer to a facility, so as to minimize the total cost of facility activation and customer connection.

Solution

Decision variables: x_{ij} for $i = 1, \ldots, n$ and $j = 1, \ldots, m$; y_j for $j = 1, \ldots, m$;

x_{ij} takes the value 1 if customer i is connected to facility j;

x_{ij} takes the value 0 if customer i is not connected to facility j;

y_j takes the value 1 if facility j is activated;

y_j takes the value 0 if facility j is not activated.

Model: min $\sum_{j=1}^{m} f_j y_j + \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij} x_{ij}$

$\sum_{j=1}^{m} x_{ij} = 1$ per $i = 1, \ldots, n$

(each customer is connected to a single system)

$x_{ij} \leq y_j$ for $i = 1, \ldots, n$, and $j = 1, \ldots, m$

(each system activates if there is at least one connected client)

$x_{ij} \geq 0$ for $i = 1, \ldots, n$, and $j = 1, \ldots, m$

$x_{ij} \leq 1$ for $i = 1, \ldots, n$, and $j = 1, \ldots, m$

x_{ij} integer for $i = 1, \ldots, n$, and $j = 1, \ldots, m$

$y_j \geq 0$ for $j = 1, \ldots, m$

$y_j \leq 1$ for $j = 1, \ldots, m$

y_j integer for $j = 1, \ldots, m$

D. Examples using the "Big M"

A very useful technique when formulating a problem in terms of (Integer) Linear Programming appears to be the "Big M" method; three examples are presented below, each within a specific context, though naturally, this technique can be applied in various other contexts.

D1. *First variant of the "transportation problem":* ***fixed costs.***

In the transportation problem, the cost function for transporting goods from source i to destination j is $c_{ij}x_{ij}$. Consider the variant where this function is $F_{ij} + c_{ij}x_{ij}$, $F_{ij} \geq 0$ representing a fixed cost (a setup cost), such that $F_{ij} \geq 0$ if $x_{ij} > 0$ (i.e., if transport between i and j takes place) and $F_{ij} = 0$ if $x_{ij} = 0$ (i.e., if transport between i and j does not take place).

Solution

New decision variables are introduced: y_{ij} for $i = 1, \ldots, s$ and $j = 1, \ldots, t$;

y_{ij} takes the value 1 if goods are transported from source i to destination j (i.e., if $x_{ij} > 0$)

y_{ij} takes the value 0 if goods are not transported from source j to destination j (i.e., if $x_{ij} = 0$)

Model: min $\sum_{i=1}^{s}\sum_{j=1}^{t} c_{ij}x_{ij} + \sum_{i=1}^{s}\sum_{j=1}^{t} F_{ij}y_{ij}$ [change in the objective function]

$\sum_{j=1}^{t} x_{ij} \leq d_i$ for $i = 1, \ldots, s$ (availability constraints)

$\sum_{i=1}^{s} x_{ij} \geq r_j$ for $j = 1, \ldots, t$ (request constraints)

$x_{ij} \geq 0$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

x_{ij} integer for $i = 1, \ldots, s$ and $j = 1, \ldots, t$ (if necessary)

[addition of constraints]

$x_{ij} \leq M\, y_{ij}$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

$y_{ij} \geq 0$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

$y_{ij} \leq 1$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

y_{ij} integer for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

Comment on the constraint $x_{ij} \leq M\, y_{ij}$:

M is a very large number (larger than any possible term in the formulation);

if $x_{ij} > 0$, then $x_{ij} \leq M\, y_{ij}$ implies $y_{ij} = 1$ (so the fixed cost in the objective function is activated);

if $x_{ij} = 0$, then $x_{ij} \leq M\, y_{ij}$ is always satisfied and causes no issues (note that, since the term F_{ij} is positive, the objective function will result in $y_{ij} = 0$).

D2. *Second variant of the "transportation problem":* ***minimum lot sizes****.*

In the transportation problem, any quantity actually transported, x_{ij} (i.e., where $x_{ij} > 0$), is not constrained to meet a specific minimum value. Consider the variant where any quantity actually transported, x_{ij} (i.e., where $x_{ij} > 0$), must be greater than or equal to a value L_{ij}, representing the minimum lot size to be transported if transport actually takes place.

Solution

New decision variables are introduced: y_{ij} for $i = 1, \ldots, s$ and $j = 1, \ldots, t$;

y_{ij} takes the value 1 if goods are transported from source i to destination j (i.e., if $x_{ij} > 0$$);

y_{ij} takes the value 0 if goods are not transported from source i to destination j (i.e., if $x_{ij} = 0$).

Model: min $\sum_{i=1}^{s}\sum_{j=1}^{t} c_{ij} x_{ij}$

$\sum_{j=1}^{t} x_{ij} \leq d_i$ for $i = 1, \ldots, s$ (availability constraints)

$\sum_{i=1}^{s} x_{ij} \geq r_j$ for $j = 1, \ldots, t$ (request constraints)

$x_{ij} \geq 0$ for $i = 1, \ldots, s$ e $j = 1, \ldots, t$;

x_{ij} integer for $i = 1, \ldots, s$ e $j = 1, \ldots, t$. (if necessary)

[addition of constraints]

$x_{ij} \leq M\, y_{ij}$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

$x_{ij} \geq L_{ij}\, y_{ij}$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

$y_{ij} \geq 0$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

$y_{ij} \leq 1$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

y_{ij} integer for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

Comment on the constraints $x_{ij} \leq M\, y_{ij}$ and $x_{ij} \geq L_{ij}\, y_{ij}$:

M is a very large number (larger than any possible term in the formulation);

if $x_{ij} > 0$, then $x_{ij} \leq M\, y_{ij}$ implies $y_{ij} = 1$, and consequently $x_{ij} \geq L_{ij}\, y_{ij}$ becomes $x_{ij} \geq L_{ij}$;

if $x_{ij} = 0$, then no problem arises (note that $0 \geq L_{ij}\, y_{ij}$ implies $y_{ij} = 0$, while $0 \leq M\, y_{ij}$ is always satisfied).

D3. *Single-processor scheduling (with deadlines and release times):* ***disjunctive constraints****.*

A set of n jobs must be processed on a single processor, which can handle only one job at a time. Each job i, for $i \in \{1, \ldots, n\}$, is characterized by:

p_i = processing time;

r_i = time before which job i cannot start (release time);

d_i = time by which job i must be completed (deadline).

Problem: assign a sequence of jobs to the processor, respecting release times and deadlines, so as to minimize the sum of the job completion times.

Solution

Decision variables: C_i for $i = 1, \ldots, n$; x_{ij} for i = 1, …, n and $j = 1, \ldots, n$ ($\neq i$);

C_i = completion time of job i;

$x_{ij} = 1$ if job i is executed before job j

$x_{ij} = 0$ if job i is not executed before job j

$$\min \quad \sum_{i=1}^{n} C_i$$

$C_i \geq r_i + p_i$	for $i = 1, \ldots, n$
$C_i \leq d_i$	for $i = 1, \ldots, n$
$C_i \leq C_j - p_j + M\,(1 - x_{ij})$	for $i = 1, \ldots, n$ and $j = 1, \ldots, n$ ($\neq i$)
$C_j \leq C_i - p_i + M\,x_{ij}$	for $i = 1, \ldots, n$ and $j = 1, \ldots, n$ ($\neq i$)
$x_{ij} \geq 0$	for $i = 1, \ldots, n$ and $j = 1, \ldots, n$ ($\neq i$)
$x_{ij} \leq 1$	for $i = 1, \ldots, n$ and $j = 1, \ldots, n$ ($\neq i$)
x_{ij} integer	for $i = 1, \ldots, n$ and $j = 1, \ldots, n$ ($\neq i$)

Comment on the two constraints $C_i \leq C_j - p_j + M\,(1 - x_{ij})$ and $C_j \leq C_i - p_i + M\,x_{ij}$:

M is a very large number (larger than any possible term in the formulation);

if $x_{ij} = 1$ (i.e., i precedes j), then we have $C_i \leq C_j - p_j$ (which is a meaningful/useful constraint for the problem) and $C_j \leq C_i - p_i + M$ (which becomes a constraint that is always satisfied);

if $x_{ij} = 0$ (i.e., i is preceded by j), then we have $C_i \leq C_j - p_j + M$ (which becomes a constraint that is always satisfied) and $C_j \leq C_i - p_i$ (which is a meaningful/useful constraint for the problem).

E. Other examples

E1. Production on a single machine

A machine M, operating for no more than 45 hours per week, can produce three types of products: P_1, P_2, and P_3. For each product type, the following are known: unit revenue, hourly production rate, and maximum weekly production capacity (as shown below).

	unit revenue	hourly production	max. weekly production
P_1	4	50 units	1,000 units
P_2	12	25 units	500 units
P_3	3	75 units	1,500 units

Determine the quantities of P_1, P_2, and P_3 to be produced in a week in order to maximize total revenue.

E2. Production at two plants

An automobile manufacturer operates two plants that produce four types of cars. The production capacities of the two plants are 8,000 and 12,000 cars, respectively (regarding capacity, it makes no difference to a plant which specific car type is produced); the required production quantities for each car type are 5,000, 4,800, 3,800, and 6,400 units, respectively. Producing a car of type j at plant i yields a profit of p_{ij}; these p_{ij} values are listed in the table below.

		Cars			
		1	2	3	4
Plants	1	2	3	4	1
	2	2	2	6	2

Determine the number of cars of each type to be produced at each plant in order to maximize total profit.

E3. Process Choice

A refinery can use two refining processes. In the Alpha process, 1 unit of Libyan crude oil and 2 units of Nigerian crude oil produce 5 units of gasoline and 2 units of diesel fuel. In the Beta process, 4 units of Libyan crude oil and 2 units of Nigerian crude oil produce 3 units of gasoline and 8 units of diesel fuel. The crude oil available is 100 units of Libyan crude oil and 150 units of Nigerian crude oil. The

expected sales (i.e., the minimum quantities to be produced) are 200 units of gasoline and 75 units of diesel fuel; the unit profits are p_B and p_G, respectively.

Determine how many times to use the Alpha and Beta processes to maximize total profit.

(hint: define decision variables x_1 and x_2 that indicate how many times to use the Alpha process and the Beta process respectively)

E4. Advertising

An advertising agency has announced that it can optimally invest its clients' money using linear programming.

There is a set N of n audience segments (e.g., singles, married couples, etc.) and a set M of m advertising media (e.g., magazines, radio, TV, etc.). For each audience segment i (where $i = 1, \ldots, n$), the client requires an exposure level E_i, representing the number of people in audience i to be reached. For each audience segment i (where $i = 1, \ldots, n$) and each advertising medium j (where $j = 1, \ldots, m$), a coefficient a_{ij} is defined, representing the number of people in the i-th audience reached when 1 Euro is spent on the j-th advertising medium. How can the agency use linear programming? (hint: define decision variables x_j = the amount allocated to advertising medium j).

Solutions

E1. Decision variables: x_i for $i = 1, 2, 3$;

x_i represents the quantity of product P_i to be produced (per week).

Model: max $4x_1 + 12x_2 + 3x_3$

$x_1 \leq 1000$

$x_2 \leq 500$

$x_3 \leq 1500$

$(1/50)\, x_1 + (1/25)\, x_2 + (1/75)\, x_3 \leq 45$

$x_i \geq 0$ for $i = 1, 2, 3$

x_i integer for $i = 1, 2, 3$

E2. Decision variables: x_{ij} for $i = 1, 2$, and $j = 1, 2, 3, 4$;

x_{ij} represents the quantity of type j automobiles produced at plant i.

Model: max $2x_{11} + 3x_{12} + 4x_{13} + 1x_{14} + 2x_{21} + 2x_{22} + 6x_{23} + 2x_{24}$

$x_{11} + x_{12} + x_{13} + x_{14} \leq 8000$

$x_{21} + x_{22} + x_{23} + x_{24} \leq 12000$

$x_{11} + x_{21} = 5000$

$x_{12} + x_{22} = 4800$

$x_{13} + x_{23} = 3800$

$x_{14} + x_{24} = 6400$

$x_{ij} \geq 0$ for $i = 1, 2$, e $j = 1, 2, 3, 4$

x_{ij} integer for $i = 1, 2$, e $j = 1, 2, 3, 4$ (se necessario)

E3. max $p_B (5x_1 + 3x_2) + p_G (2x_1 + 8x_2)$

$x_1 + 4x_2 \leq 100$

$2x_1 + 2x_2 \leq 150$

$5x_1 + 3x_2 \geq 200$

$2x_1 + 8x_2 \geq 75$

$x_1, x_2 \geq 0$

E4. min $\sum_{j=1}^{m} x_j$

$\sum_{j=1}^{m} a_{ij} x_j \geq E_i$ $\quad i = 1, \ldots, n$

$x_j \geq 0$ $\quad j = 1, \ldots, m$

F. Appendix: Examples in the healthcare sector

The literature contains many articles on the potential applications of Operations Research to problems in the healthcare sector; in particular, the value of such applications, which generally require adaptation to specific contexts, lies in providing decision support.

Links on the subject: http://www.choir.utwente.nl/en/orchestra/ and http://orahs.di.unito.it/

Below are some potential applications that utilize (Integer) Linear Programming.

1. Hospital shifts (see M. Fischetti's book; for various aspects, see also [a2] and [a8])
Nurse shifts need to be defined. A certain number of nurses must be present each day. Specifically, a nurse's shift consists of five consecutive days of work followed by two consecutive days off. The goal is to minimize the total number of nurses involved.

Parameters:
:: let $D = \{1, \ldots, 7\}$ be the set of days of the week;
:: for $i = 1, \ldots, 7$, let r_i be the estimated number of nurses required for day i;
:: shifts follow this pattern: five consecutive working days followed by two days off.

Decision variables:
x_i for i = 1, …, 7;
x_i = number of nurses starting their shift on day i;

Model:

$$\min \quad \sum_{i=1}^{7} x_i$$

$$x_4 + x_5 + x_6 + x_7 + x_1 \geq r_1$$
$$x_5 + x_6 + x_7 + x_1 + x_2 \geq r_2$$
$$x_6 + x_7 + x_1 + x_2 + x_3 \geq r_3$$
$$x_7 + x_1 + x_2 + x_3 + x_4 \geq r_4$$
$$x_1 + x_2 + x_3 + x_4 + x_5 \geq r_5$$
$$x_2 + x_3 + x_4 + x_5 + x_6 \geq r_6$$
$$x_3 + x_4 + x_5 + x_6 + x_7 \geq r_7$$

$x_i \geq 0$ for $i = 1, \ldots, n$,

x_i integer for $i = 1, \ldots, n$.

2. Operating room (cf. [a9] slides 22–28; also, for various aspects, cf. [a1] and [a6])

Operations must be performed in the operating room of an Orthopedics department. The daily availability duration of the operating room is known. Operation durations are known from historical data. The problem is to determine a sequence of operations to be performed, with the objective of minimizing the number of days the operating room is used.

Parameters:

:: let $P = \{1, \ldots, n\}$ be the set of operations to be performed;

:: for $i = 1, \ldots, n$, let di be the estimated duration (in hours) for operation i;

:: let $G = \{1, \ldots, m\}$ be the set of days within a time horizon of m days;

:: for $j = 1, \ldots, m$, let K_j be the time (in hours) during which the operating room can be used on day j.

Decision variables:

x_{ij} for $i = 1, \ldots, n$, and for $j = 1, \ldots, m$;

$x_{ij} = 1$ if operation i is performed on day j;

$x_{ij} = 0$ otherwise;

y_j for $j = 1, \ldots, m$;

$y_j = 1$ if the operating room is used on day j;

$y_j = 0$ otherwise;

Model:

$$\min \quad \sum_{j=1}^{m} y_j$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \text{for } i = 1, \ldots, n$$

$$\sum_{i=1}^{n} d_i x_{ij} \leq K_j \qquad \text{for } j = 1, \ldots, m$$

$$\sum_{i=1}^{n} x_{ij} \leq M y_j \qquad \text{for } j = 1, \ldots, m$$

(where M is a very large scalar)

$x_{ij} \in \{0, 1\}$ for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$

$y_j \in \{0, 1\}$ for $j = 1, \ldots, m$

3. Allocation of patients to wards (for various aspects, see [a8])

The problem is to allocate the maximum possible number of people, from among those requesting hospital admission to wards, taking into account that each ward has a limited number of available beds and that each person can only be admitted to specific wards.

Parameters:

:: let $P = \{1, \ldots, n\}$ be the set of people to be admitted to the hospital;

:: let $R = \{1, \ldots, m\}$ be the set of hospital wards;

:: for $j = 1, \ldots, m$, let d_j be the number of available beds in ward j;

:: let $F = \{(i, j)$: person i can be admitted to ward $j\}$

(i.e., F is the set of compatible person/ward pairs, considering that each person can only be admitted to certain wards, for various reasons, such as to prevent contagion).

Decision variables:

x_{ij} for each $(i, j) \in F$;

$x_{ij} = 1$ if person i is assigned to ward j;

$x_{ij} = 0$ if person i is not assigned to ward j.

Model: max $\sum_{(i, j) \in F} x_{ij}$

$\sum_{(i, j) \in F} x_{ij} \leq 1$ for $i = 1, \ldots, n$

$\sum_{(i, j) \in F} x_{ij} \leq d_j$ for $j = 1, \ldots, m$

$x_{ij} \in \{0, 1\}$ for any $(i, j) \in F$

4. Locating CUPs or Out-of-Hours Medical Services (cf. [Fischetti])

The problem involves determining in which neighborhoods of a large city to install CUPs (Unified Booking Centers), subject to the constraint that every user must be able to reach a CUP within a specified time limit, while aiming to install the minimum possible number of centers.

The same problem can be formulated with reference to municipalities within a region rather than neighborhoods in a large city, and regarding out-of-hours medical services instead of CUPs.

Parameters:

:: let $Q = \{1, \ldots, n\}$ be the set of neighborhoods in a large city;

:: for $i = 1, \ldots, n$ and $j = 1, \ldots, n$, let t_{ij} be the average travel time from neighborhood i to neighborhood j;

:: let T be the maximum tolerable time;

:: for $i = 1, \ldots, n$, let $R(i) = \{j \in Q: t_{ij} \leq T\}$; that is, $R(i)$ is the set of neighborhoods reachable from neighborhood i in a time less than T.

Decision variables:

x_i for $i = 1, \ldots, n$;

$x_i = 1$ if a CUP is installed in neighborhood i;

$x_i = 0$ otherwise.

Model: $\min \sum_{i=1}^{n} x_i$

$$\sum_{j \in R(i)} x_j \geq 1 \qquad \text{for } i = 1, \ldots, n$$

$$x_i \in \{0, 1\} \qquad \text{for } i = 1, \ldots, n$$

5. Purchase of machinery

The problem involves purchasing new hospital machinery within a fixed budget, with the goal of maximizing the utility of the purchased equipment (where the utility of a piece of machinery is defined as the estimated number of people who will benefit from it over a specific time horizon).

Parameters:

:: let $M = \{1, \ldots, n\}$ be the set of potential machinery to be purchased;

:: let b be the budget available for such purchases;

:: for $i = 1, \ldots, n$: let c_i be the cost of machine i, and let u_i be the utility of machine i (defined as the estimated number of people who will benefit from that machine over a specific time horizon).

Decision variables:

x_i for $i = 1, \ldots, n$;

$x_i = 1$ if machine i is purchased;

$x_i = 0$ otherwise.

Model: $\max \sum_{i=1}^{n} u_i x_i$

$$\sum_{i=1}^{n} c_i x_i \leq b$$

$$x_i \in \{0, 1\} \qquad \text{for } i = 1, \ldots, n$$

6. <u>Redefining the hospital network (resource cuts)</u>

The problem involves redefining the departments within a region's hospital network following resource cuts. For the sake of simplicity, let us focus on a single type of department, for example, Orthopedics.

Parameters:

:: $\boldsymbol{R} = \{\boldsymbol{R_1}, \ldots, \boldsymbol{R_n}\}$ = the set of Orthopedics departments in the region;

:: $\boldsymbol{D} = \{\boldsymbol{D_1}, \ldots, \boldsymbol{D_m}\}$ = set of districts into which the Region is divided;

:: for $i = 1, \ldots, n$: $\boldsymbol{r_i}$ = maximum number of expected patients estimated to be admissible to the R_i ward within a given time horizon;

:: for $j = 1, \ldots, m$: $\boldsymbol{d_j}$ = number of (expected) patients estimated to require admission to one of the R-type wards (i.e., an Orthopedics ward) from district D_j within a specific time horizon;

:: for $i = 1, \ldots, n$: $\boldsymbol{C_i}$ = "fixed cost" = estimated cost solely for opening the R_i department within a specific time horizon;

:: for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$: $\boldsymbol{c_{ij}}$ = "marginal cost" = estimated cost of admitting a patient from district D_j to department R_i within a specific time horizon [this cost can be understood as c_i +

q_{ij}, where: c_i is the estimated cost incurred by the Region to admit a generic patient to department R_i, and q_{ij} is the estimated cost incurred by the patient from district D_j to be admitted to department R_i]

Problem: determine which departments, from the set $\{R_1, \ldots, R_n\}$, should be closed, within the context of managing hospital admissions such that every patient is admitted to one of the departments in R, while minimizing the sum of "fixed costs" and "marginal costs."

Decision variables:

x_{ij} for $i = 1, \ldots, n$ and $j = 1, \ldots, m$

y_i for $i = 1, \ldots, n$

where:

x_{ij} = number of (expected) patients admitted to department R_i from district D_j

$y_i = 1$ if department R_i is to remain open

$y_i = 0$ if department R_i is to be closed

Model: min $\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n} C_i y_i$

$$\sum_{i=1}^{n} x_{ij} \geq d_j \qquad \text{for } j = 1, \ldots, m$$

$$\sum_{j=1}^{m} x_{ij} \leq r_i \qquad \text{for } i = 1, \ldots, n$$

$$\sum_{j=1}^{m} x_{ij} \leq M\, y_i \quad \text{for } i = 1, \ldots, n \quad (\text{where } M \text{ is a very large scalar})$$

$$x_{ij} \geq 0 \qquad \text{for } i = 1, \ldots, n \text{ and for } j = 1, \ldots, m$$

$$x_{ij} \geq \underline{\text{integer}} \qquad \text{for } i = 1, \ldots, n \text{ and for } j = 1, \ldots, m$$

$$y_i \in \{0, 1\} \qquad \text{for } i = 1, \ldots, n$$

7. Other

Some other possible applications include the following:

:: Blood resource management [a4],

:: Radiotherapy treatments [a3], [a5],

:: Inventory management (e.g., of pharmaceuticals) [a10].

Finally, it is worth noting that Linear (Integer) Programming models are highly adaptable to changes or variations in the scenarios being modeled.

References

[a1] J.T. Blake, J. Donald, Mount Sinai hospital uses integer programming to allocate operating room time, *Interfaces*, 32 (2002), pp. 63-73

[a2] B. Cheang, H. Li, A. Lim, and B. Rodrigues, Nurse rostering problems-a bibliographic survey, *European Journal of Operational Research*, 151**,** 447-460 (2003).

[a3] D. Conforti, F. Guerriero, R. Guido, Non-block scheduling with priority for radiotherapy treatments**,** *European Journal of Operational Research*, 201 (1) (2010), pp. 289-296

[a4] V. De Angelis, N. Ricciardi, and G. Storchi, Optimizing blood assignment in a donation - transfusion system, *International Transactions in Operational Research*, August 2001

[a5] M. Ehrgott, R. Johnston, Optimisation of beam directions in intensity modulated radiation therapy planning, *OR Spectr* 25 (2003) 251–264

[a6] A. Guinet, S. Chaabane, Operating theatre planning, *International Journal of Production Economics*, 85 (2003), pp. 69-81

[a7] A. Mazier, X. Xie, M. Sarazin, Real-Time Patients Assignment: a Method for Improving Emergency Department Flow. In: Proceedings of the IEEE workshop on Health Care Management. Venetia (Italia), 2010

[a8] B. Satheeshkumar, S. Nareshkumar and S. Kumaraghuru, Linear programming applied to nurses shifting problems, *International journal of science and research*, 3 (2014) 171–173

[a9] www.unive.it/persone/pesenti

[a10] https://www.vkok.ee/logontrain/wp.../Riga-3-july-2014.pdf

2.1 Exercises on Chapter 2

Group P: *Formulate the following problems in terms of (Integer) Linear Programming; each can be directly traced back to one of the basic examples in Group A. Specifically, the solutions are given below, indicating only the basic example.*

P1. The Civil Defense Department is organizing relief efforts for eight different zones, namely, 1, 2, 3, 4, 5, 6, 7, 8, devastated by a disaster. Seven rescue teams are available: A, B, C, D, E, F, and G. Each team can only provide relief to one zone. Furthermore, each team is only suitable for certain zones, depending on the damage caused by the disaster in the various zones. Specifically:
A is only suitable for zones 3, 5, 6, and 8;
B only for zones 3 and 7;
C only for zones 2, 3, and 7;
D only for 2, 4;
E only for 1, 3, 4, 5, 6, 7;
F only for 1, 4, 5;
G only for 3, 8.
The problem is to adequately address the greatest number of zones.
(Note: this problem is equivalent to maximizing the number of compatible matches.)

P2. A multinational company decides to open new branches, having a budget of 15. Specifically, it can choose between 7 potential branches, namely A, B, C, D, E, F, and G. Each of these potential branches requires a start-up cost, as follows: $c_A = 7$, $c_B = 5$, $c_C = 3$, $c_D = 9$, $c_E = 4$, $c_F = 8$, $c_G = 10$. On the other hand, each of them is estimated to guarantee a profit after one year, as follows: $p_A = 4$, $p_B = 5$, $p_C = 7$, $p_D = 8$, $p_E = 3$, $p_F = 5$, $p_G = 9$. It is understood that a branch cannot be opened partially (i.e., either a branch opens or it doesn't). The problem is to choose which branches to open in order to maximize the sum of the profits after one year.

P3. A tobacco manufacturing plant wishes to produce two types of cigars: one of high quality (Original) and the other of medium quality (Antique). The company expects a profit of 10 from producing one unit of the Original cigar and a profit of 5 from producing one unit of the Antique cigar. Production requires a specific combination of two types of tobacco, designated as Type A and Type B. Producing one unit of the Original cigar requires 7 units of Type A tobacco and 3 units of Type B tobacco. Producing one unit of the Antico cigar requires 2 units of Type A tobacco and 8 units of Type B tobacco. Finally, the plant has a supply of 1,000 units of Type A tobacco and 2,400 units of Type B tobacco. The problem is to determine the quantities of Original and Antique cigars to produce in order to maximize total profit.

P4. Four dealerships, namely, A, B, C, D, have run out of a specific type of car (due to a rail transport stoppage). This situation can be remedied by supplying these dealerships from two nearby dealerships that have this type of car in stock. These supplying dealerships have 8 and 15 cars available,

respectively. Dealerships A, B, C, D require 3, 4, 7, 2 cars, respectively. Delivering a single car from dealership i (where $i = 1, 2$) to dealership j (where j = A, B, C, D) incurs a cost c_{ij}. Specifically: $c_{1A} = 4$; $c_{1B} = 4$; $c_{1C} = 5$; $c_{1D} = 5$; $c_{2A} = 2$; $c_{2B} = 7$; $c_{2C} = 5$; $c_{2D} = 7$. The problem is to plan deliveries to dealerships A, B, C, D, so as to minimize the total cost.

P5. A certain firm's work consists of extracting three raw materials, namely, A, B, C, by decomposing the material it extracts from two quarries, quarry 1 and quarry 2. Specifically:
From 1 units of material extracted from quarry 1, the firm obtains 5 u. of A, 3 u. of B, and 0 u. of C;
From 1 unit of material extracted from quarry 2, the firm obtains 3 u. of A, 2 u. of B, and 3 u. of C.
The cost of extracting 1 unit of material from quarry 1 is 30.
The cost of extracting 1 unit of material from quarry 2 is 40.
The production plan requires that at least 70 u. of A, 40 u. of B, 20 u. of C be extracted.
The problem is to determine the number of units of material to be extracted from quarry 1 and quarry 2 respectively so as to minimize the total cost (satisfying the production plan).

P6. A person inherits five villas. In general, he or she plans to sell them, but wants to keep a certain subset S such that: at least one of the villas in S is by the sea; at least one of the villas in S is in a quiet location; at least one of the villas in S is near a hospital; at least one of the villas in S is green. The situation is as follows:
Villa 1: not by the sea; not in a quiet location; near the hospital; green;
Villa 2: not by the sea; in a quiet location; near the hospital; not green;
Villa 3: by the sea; not in a quiet location; not near the hospital; green;
Villa 4: by the sea; in a quiet location; not near the hospital; not green.
Villa 5: by the sea; not in a quiet location; near the hospital; not green.
The problem is to maximize the number of villas to sell.
(Note: this problem is equivalent to minimizing the number of villas to keep.)

P7. A private TV station wants to determine its broadcast schedule for one of the upcoming weeks, focusing specifically on prime-time programs. There are 10 programs available; the station needs to decide which program to air on each of the 7 evenings of the week (with the understanding that each program can be aired on only one evening). Broadcasting program i incurs a cost c_i (for production, external acquisition, etc.), where $i = 1, \ldots, 10$. Based on agreements already reached with various sponsors, the station estimates that if program i is aired on evening j, it will generate revenue r_{ij} (thanks to the sponsors), where $i = 1, \ldots, 10$ and $j = 1, \ldots, 7$. The problem is to determine which program to air on each of the 7 evenings of the week so as to maximize total profit (calculated as total revenue minus total cost).

Solutions

P1. Referable to “Matching”

P2. Referable to “Packing”

P3. Referable to “Resource composition”

P4. Referable to “Transports”

P5. Referable to “Resource decomposition”

P6. Referable to “Covering”

P7. Referable to “Assignment”.

Group Q: *Formulate the following problems using (Integer) Linear Programming; each can be traced back, more or less, to one of the basic examples from Group A, albeit with variations (specifically, the "Big M" variant often appears). Solutions are provided below.*

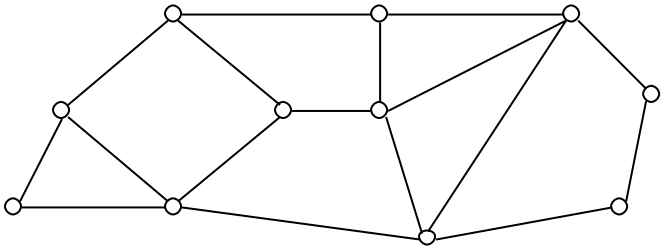
Q1. A company manufactures two products, A and B. Production takes place on the company's single machine, M; M can produce only one unit (of either product) at a time and can operate for a maximum of 1,500 hours per week. Additionally, producing each product requires a certain amount of raw material P, of which the company holds a stock of 1,200 units.
Producing one unit of A requires 5 hours of machine time on M and 4 units of P. Producing one unit of B requires 4 hours of machine time on M and 3 units of P. Furthermore, the company has the following requirements: (i) for product A, it wants to produce between 100 and 250 units; (ii) for product B, if any units are produced at all, the quantity must be at least 100 units. The profit generated from producing one unit of A is 25, while the profit from producing one unit of B is 15.
The problem is to determine the number of units of A and B to produce in order to maximize total weekly profit.

Q2. A city municipality has issued calls for tenders for 4 contracts, namely, A, B, C, D, corresponding to 4 separate projects. There are 7 companies in the city, labeled 1, 2, …, 7. Each company i (where i = 1, 2, …, 7) submits a cost estimate s_{ij} for each contract j (where j = A, B, C, D). Due to precautionary or legal reasons, the municipality cannot assign more than two contracts to the same company.
The problem is to determine which company should be assigned each contract so as to minimize total costs.

Q3. A pharmaceutical company produces vitamins B and C by extracting them from three types of food sources: A1, A2, A3. From 1 unit of A1, 3 units of B and 2 units of C are extracted. From 1 unit of A2, 2 units of B and 3 units of C are extracted. From 1 unit of A3, 1 unit of B and 5 units of C are extracted. These food sources must be purchased. Purchasing A1 and A2 is straightforward (one simply goes to a location near the company), and their unit costs are 3 and 4, respectively. Purchasing A3 is more complex: the unit cost is 2, but if any are purchased (i.e., a quantity greater than 0), a minimum of 30 units must be bought, and a fixed transport cost of 200 must be paid. The company must produce at least 150 units of vitamin B and at least 300 units of vitamin C.
The problem is to determine how many units of A1, A2, A3, to purchase so as to minimize total costs.

Q4. One morning, an employment agency receives an email containing a set M of m job requests, corresponding to a set N of n people registered to look for work. Each person i (for i = 1, …, n) can be matched to only one job from a specific subset M_i of M; conversely, each job j (for j = 1, …, m) can be matched to only one person from a specific subset N_j of N.
The problem is to place as many people as possible, that is, to maximize the number of compatible person-job pairings (hint: define a set of compatible pairs).

Q5. In a museum, cameras need to be installed so that all the corridors are visible. In the graph $G = (V, E)$ below [V is the set of nodes, E is the set of edges], each edge represents a corridor, and each node represents the intersection point between the corridors that intersect it. By installing a camera on a node, all and only the edges that intersect it are displayed.
The problem is to determine the nodes on which to install the cameras, ensuring that all the edges are displayed, so as to install the smallest possible number of cameras.
[As a notation, for each edge $e \in E$, $V(e)$ can be denoted as the set formed by the pair of nodes that form the edge e; thus, if a camera is placed on a node $v \in V$, then all and only the edges e such that $v \in V(e)$ are displayed].



Q6. A car manufacturer has 3 production plants: one in Italy (I), one in Poland (P), and one in Slovenia (S). The manufacturer produces 4 types of cars, labeled 1, 2, 3, 4. Each plant is capable of producing any of these car types. In general, it is estimated that each plant can produce a maximum of d_I, d_P, d_S, cars per year, respectively (regardless of the car type). On the other hand, it is estimated that r_1, r_2, r_3, r_4 cars of types 1, 2, 3, 4, respectively, must be produced per year. The cost of producing one car of type j (where $j = 1, 2, 3, 4$) at plant i (where $i = I, P, S$) is c_{ij}. Furthermore, the annual cost of keeping each of these plants open is C_I, C_P, C_S, respectively [note: a plant is kept open only if at least one car of any type is produced there].
The problem is to determine the production volume of each car type at each plant so as to minimize total annual costs.

Q7. A company has the opportunity to make investments, choosing from n possible options. Each investment i (where $i = 1, \ldots, n$) is financed over two years, namely, Year 1 and Year 2; that is, investing in i requires paying a sum a_{i1} at the beginning of Year 1 and a sum a_{i2} at the beginning of Year 2. Then, at the end of Year 2, investment i yields a profit p_i. The company can invest a sum b_1 at the beginning of Year 1 and a sum b_1 at the beginning of Year 2 (these available funds do not allow for all investments to be made). Investments cannot be made partially; that is, they are either undertaken or not.
The problem is to select the investments to undertake so as to maximize total profit at the end of Year 2.

Solutions

Q1.

Decision variables:

x_A = produced quantity of A

x_B = produced quantity of B

y_B binary variable

$y_B = 1$ if something *B* is produced (that is, if $x_B > 0$)

$y_B = 0$ otherwise (that is, if $x_B = 0$)

$$\max \; 25x_A + 15x_B$$

$$5x_A + 4x_B \leq 1500$$

$$4x_A + 3x_B \leq 1200$$

$$100 \leq x_A \leq 250$$

$x_B \leq My_B$ (where *M* is a very large scalar)

$$x_B \geq 100y_B$$

$$x_A, x_B \geq 0$$

$$0 \leq y_B \leq 1$$

y_B integer

Q2.

Decision variables:

x_{ij} for $i = 1, \ldots, 7$ and $j = 1, \ldots, 4$

x_{ij} = 1 if contract *i* is awarded to company *j*

x_{ij} = 0 otherwise

$$\min \quad \sum_{i=1}^{7}\sum_{j=1}^{4} s_{ij}x_{ij}$$

$$\sum_{i=1}^{7} x_{ij} = 1 \qquad \text{for } j = 1, \ldots, 4$$

$$\sum_{j=1}^{4} x_{ij} \leq 2 \qquad \text{for } i = 1, \ldots, 7$$

$$0 \leq x_{ij} \leq 1 \qquad \text{for } i = 1, \ldots, 7 \text{ and for } j = 1, \ldots, 4$$

x_{ij} integer for $i = 1, \ldots, 7$ and for $j = 1, \ldots, 4$

Q3.

Decision variables:

x_i = amount of food A*i* you need to buy, for $i = 1, 2, 3$

y_3 binary variable

$y_3 = 1$ if you buy something from A3 (that is, if $x_3 > 0$)

$y_3 = 0$ otherwise (that is, if $x_3 = 0$)

$$
\begin{aligned}
\min \quad & 3x_1 + 4x_2 + 2x_3 + 200\, y_3 \\
& 3x_1 + 2x_2 + \quad x_3 \geq 150 \\
& 2x_1 + 3x_2 + \ 5x_3 \geq 300 \\
& x_3 \leq My_3 \quad \text{(where } M \text{ is a very large scalar)} \\
& x_3 \geq 30y_3 \\
& x_1, x_2, x_3 \geq 0 \\
& 0 \leq y_3 \leq 1 \\
& y_3 \ \underline{\text{integer}}
\end{aligned}
$$

Q4.

Let F be the set of *compatible pairs* (i, j), that is, $F = \{(i, j) : i \in P_j \text{ e } j \in M_i\}$;

Decision variables:

x_{ij} for each pair $(i, j) \in F$

$x_{ij} = 1$ if person i is placed for job j

$x_{ij} = 0$ otherwise

$$
\begin{aligned}
\max \quad & \sum_{(i,j)\in F} x_{ij} \\
& \sum_{(i,j)\in F} x_{ij} \leq 1 \qquad \text{for } i = 1, \ldots, n \\
& \sum_{(i,j)\in F} x_{ij} \leq 1 \qquad \text{for } j = 1, \ldots, m \\
& 0 \leq x_{ij} \leq 1 \qquad \text{for any } (i, j) \in F \\
& x_{ij} \ \underline{\text{integer}} \qquad \text{for any } (i, j) \in F
\end{aligned}
$$

Q5.

Decision variables:

x_v for each node v of G

$x_v = 1$ if a camera is installed on node v

$x_i = 0$ otherwise

$$
\begin{aligned}
\min \quad & \sum_{v\in V} x_v \\
& \sum_{v\in V(e)} x_v \ \geq 1 \qquad \text{for any edge } e \text{ of } G \\
& 0 \leq x_v \leq 1 \qquad \text{for any vertex } v \text{ of } G \\
& x_v \ \underline{\text{integer}} \qquad \text{for any vertex } v \text{ of } G
\end{aligned}
$$

Q6.

Decision variables: x_{ij} for each i = I, P, S and $j = 1, \ldots, 4$;

x_{ij} represents the quantity of cars of type j produced in plant i

y_i binary variable, for i = I, P, S

$y_i = 1$ if something is being produced at plant j (that is, if $\sum_{j=1}^{4} x_{ij} > 0$)

$y_i = 0$ otherwise (that is, if $\sum_{j=1}^{4} x_{ij} = 0$)

$$\min \quad \sum_{i=I,P,S} \sum_{j=1}^{4} c_{ij} x_{ij} + \sum_{i=I,P,S} C_i y_i$$

$$\sum_{j=1}^{4} x_{ij} \leq d_i \qquad \text{for } i = \text{I, P, S}$$

$$\sum_{i=I,P,S} x_{ij} \geq r_j \qquad \text{for } j = 1, \ldots, 4$$

$x_{ij} \geq 0$ for i = I, P, S e $j = 1, \ldots, 4$;

x_{ij} integer for i = I, P, S e $j = 1, \ldots, 4$;

$\sum_{j=1}^{4} x_{ij} \leq My_i$ for i = I, P, S (where M is a very large scalar)

$0 \leq y_i \leq 1$ for i = I, P, S

y_i integer for i = I, P, S

Q7.

Decision variables:

x_i for $i = 1, \ldots, n$

$x_i = 1$ if the investment i is made

$x_i = 0$ otherwise

$$\max \quad \sum_{i=1}^{n} p_i x_i$$

$$\sum_{i=1}^{n} a_{i1} x_i \leq b_1$$

$$\sum_{i=1}^{n} a_{i2} x_i \leq b_2$$

$0 \leq x_i \leq 1$ for $i = 1, \ldots, n$

x_i integer for $i = 1, \ldots, n$

Group R: *The following three problems—for which no solutions are provided—are linked to real-world scenarios (for example, the "Siege" problem relates to the apparent use of mathematical programming by the Anglo-Americans during World War II, specifically during the naval blockade imposed on the British by the Germans in the early stages of the war) and are introduced as topics for further study.*

R1. *Siege*

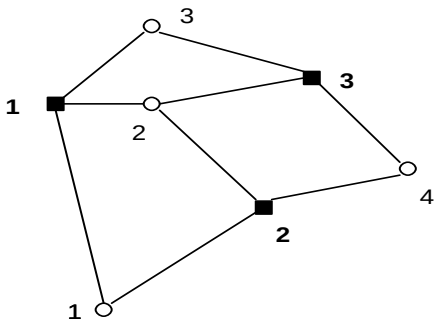
During a war, a city with a population of 25,000 is placed under siege by its enemies. The challenge is to ration the distribution of food. Food reserves consist of three items, R, S, T, in quantities of 7,500,000, 5,000,000, 12,000,000 units, respectively. To survive, each person is estimated to require three nutritional components, let us call them A, B, C, in daily amounts of at least 30, 25, 70 units:

1 unit of R yields 2 units of A, 1 unit of B, and 10 units of C;
1 unit of S yields 10 units of A, 4 units of B, and 15 units of C;
1 unit of T yields 7 units of A, 12 units of B, and 0 units of C.

The problem is to determine an optimal food distribution strategy that ensures survival for the greatest possible number of days (in other words: to calculate the maximum number of days the city can hold out while providing the food necessary for survival).

R2 (Version 1). *Warehouses*

In an existing distribution network for a certain good, consisting of warehouses that supply stores, the goal is to examine the possibility of reducing the number of warehouses, thereby retaining (keeping active) only a subset of them. For various reasons, not all warehouses can supply all stores. In the figure, warehouses are represented by black squares and stores by white circles; the presence of an arc indicates that a specific warehouse can supply a specific store.



Problem 1: The problem is to select which warehouses to retain so as to minimize the total number of retained warehouses, while ensuring that every store can be supplied.

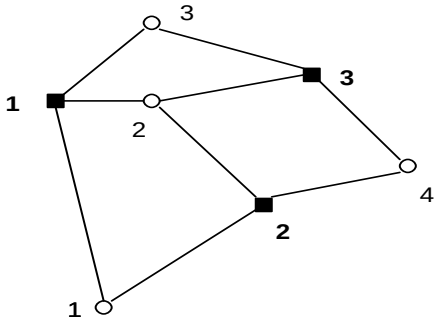
Variant of Problem 1: Under the assumptions of the previous problem, suppose that:
:: retaining warehouse i incurs a cost C_i.
The problem is to select which warehouses to retain so as to minimize the total cost of retaining warehouses, while ensuring that every store can be supplied.

Problem 2: Suppose that:
:: retaining warehouse i incurs a cost C_i ;
:: each warehouse i has a monthly supply capacity of d_i ;
:: each store j has a monthly demand of r_j .
The problem is to select which warehouses to retain so as to minimize the total cost of retaining warehouses, while ensuring that every store can be supplied and that it receives its required monthly supply.

Variant of Problem 2: Under the assumptions of the previous problem, suppose that:
:: monthly supply from warehouse i to store j incurs a unit cost of c_{ij}.
The problem involves selecting which warehouses to confirm and planning transportation from the confirmed warehouses to the stores, so as to minimize the total costs associated with warehouse confirmation and transportation over a 40-month time horizon, while ensuring that every store can be supplied and receives monthly deliveries in accordance with its demand.

R2 (second version). *Humanitarian aid*

A humanitarian organization wishes to activate distribution centers to supply certain villages, with a set of potential centers available for use. For various reasons, not all potential centers can supply all villages. In the figure, potential centers are represented by black squares and villages by white circles; an arc indicates the possibility of supplying a village from a potential center.



Problem 1: The goal is to select which centers to activate so as to minimize the total number of activated centers, while ensuring that every village can be supplied.

Variant of Problem 1: Under the assumptions of the previous problem, assume that:
:: activating center i incurs a cost C_i.
The goal is to select which centers to activate so as to minimize the total activation costs, while ensuring that every village can be supplied.

Problem 2: Assume that:
:: activating center i incurs a cost C_i;
:: each center i has a monthly supply capacity of d_i;
:: each village i has a monthly demand of r_j.
The goal is to select which centers to activate so as to minimize the total activation costs, while ensuring that every village can be supplied and receives its required monthly supply.

Variant of Problem 2: Under the assumptions of the previous problem, assume that:
:: the monthly supply from center i to village i incurs a unit cost of c_{ij}.
The goal is to select which centers to activate and to plan the transport from activated centers to villages so as to minimize the total activation and transport costs over a 40-month time horizon, while ensuring that every village can be supplied and receives its required monthly supply.

R3. *Weekly Shopping*

A person shops once a week at one of the following three supermarkets, namely, AAA, BBB, or CCC, depending on chance. This week, however, the person wants to minimize the total cost of the shopping trip. The list of products to be purchased, along with their unit prices at each supermarket, is shown in the table below:

	AAA (unit price)	BBB (unit price)	CCC (unit price)
Tomato: 2 unità	€ 2.80	€ 3.00	€ 3.50
Mozzarella: 4 unità	€ 1.50	€ 1.60	€ 1.70
Pasta: 5 unità	€ 0.80	€ 0.80	€ 0.90
Minestrone: 2 unità	€ 2.50	€ 3.20	€ 3.80
Wine: 2 unità	€ 4.50	€ 4.30	€ 4.20

The following fixed costs must also be considered (representing costs in terms of time and potential car usage): traveling to AAA incurs a cost of € 4.00, traveling to BBB incurs a cost of € 2.50, and traveling to CCC incurs a cost of € 0.50.

The problem is to determine the best course of action, specifically, what to buy at each supermarket. in order to minimize the total shopping cost.

3. Essentials of Linear Programming (LP)

A *Linear Programming* problem, or LP for short, is a Mathematical Programming problem

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

where:

$X \subseteq \mathbb{R}^n$ ($n \in \mathbb{N}$) is the set of solutions to a system of linear equations and/or inequalities;

$f : X \to \mathbb{R}$ is a linear function.

Observation

Any LP problem can be equivalently represented in one of the following two ways:

min **cx**	**min** **cx**
Ax $\leq$ **b**	**Ax** = **b**
x $\geq$ **0**	**x** $\geq$ **0**
CANONICAL form	STANDARD form

In other words, one can always reduce the problem to either the CANONICAL form [where, in addition to the *non-negativity constraints* **x** $\geq$ **0**, there are only inequalities] or the STANDARD form [where, in addition to the *non-negativity constraints* **x** $\geq$ **0**, there are only equalities].

The formulations above employ vector notation; specifically, **x** is a vector with n components, **c** is a vector with n components, **A** is an $n \times m$ matrix, and **b** is a vector with m components. If there is any uncertainty, this notation can be reviewed in greater detail, for instance, by revisiting the definitions of the product of two vectors and the product of a matrix and a vector (e.g. see in Google "vector notation for systems of linear equations").

To demonstrate that any LP problem can be equivalently represented in either of these two ways (i.e., reduced to them), there are certain *transformation rules*, outlined below in a concise manner:

1) conversion from **max** to **min**

   in fact **max** **wx** = – **min** (– **wx**)

2) conversion of contraints from "$\leq$" to "="

   in fact $a_i\mathbf{x} \leq b_i \rightarrow a_i\mathbf{x} + s_i = b_i \ , \ s_i \geq 0$

   where a_i is the i-th row of $\mathbf{A}$, b_i is the i-th component of $\mathbf{b}$, and s_i is a new variable.

3) arrangement of variables x_i unconstrained in sign

   in fact $x_i \rightarrow x_i = x^+_i - x^-_i \ , \ x^+_i \geq 0 \ , \ x^-_i \geq 0$

   where x_i is the i-th component of $\mathbf{x}$, while x^+_i and x^-_i are new variables. □

At this point, for convenience, we denote an LP problem as $\mathbf{min}\{\mathbf{cx} : \mathbf{x} \in P\}$, where P is the set of solutions to a system of linear equations and/or inequalities.

***Definition*:** An LP problem is said to be

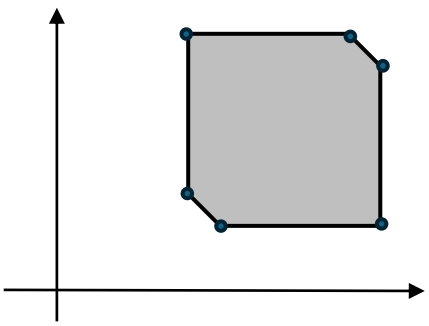
:: *unbounded* if $\mathbf{min}\{\mathbf{cx} : \mathbf{x} \in P\} = -\infty$

:: *infeasible* if $P = \varnothing$. □

The definition above describes two limiting cases.

***Definition*:** The set of solutions to a system of linear equations and/or inequalities is called a *polyhedron*; a bounded polytope, that is, one that can be enclosed within a bounding region, is called a *polytope*. □

Definition: A point of a polyhedron P is called a *vertex* of P if it does not lie on a line segment whose endpoints are two points of P. □

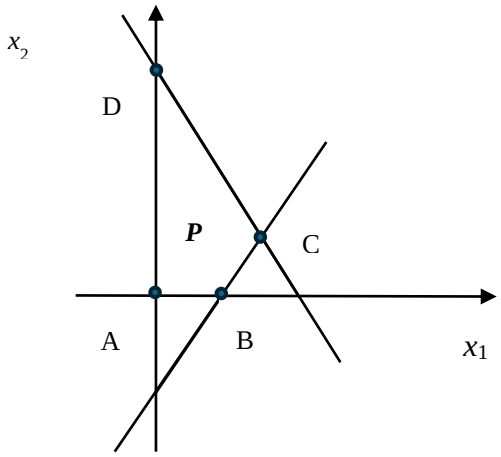


The figure above depicts a two-dimensional polyhedron (specifically, a polytope) in gray, with its vertices marked by dots; thus, the vertices correspond exactly to what we intuitively understand them to be, in accordance with the definition.

***Theorem* 3.1:** Given an LP problem **min**{**cx** : **x** $\in$ P}, if P is a polytope, then there exists at least one optimal solution that coincides with a vertex of P. □

An example of application: Consider the following LP problem.

$$\begin{aligned} \min \quad & -x_1 - x_2 \\ & 6x_1 + 4x_2 \leq 24 \\ & 3x_1 - 2x_2 \leq 6 \\ & x_1 , x_2 \geq 0 \end{aligned}$$



Based on Theorem 3.1, we can devise the following algorithm to determine an optimal solution:
list all the vertices, calculate the function value for each of them, and choose the "best" one.

A = (0, 0) $\rightarrow$ the function value at A is: $-0-0=0$

B = (2, 0) $\rightarrow$ the function value at A is: $-2-0=-2$

C = (3, 3/2) $\rightarrow$ the function value at A is: $-3-3/2=-9/2$

D = (0, 6) $\rightarrow$ the function value at A is: $-0-6=-6$

Thus, by Theorem 3.1, D is an optimal solution to the problem; that is, (x_1 , x_2) = (0, 6) is an optimal solution to the problem. □

However, in general, it is not possible to identify the vertices "graphically" as in the example above. Therefore, an *algebraic characterization of the vertices* is needed, that is, a result that allows us to identify the vertices without relying on a graphical representation.

Consider an LP problem in STANDARD form:

min **cx**

Ax = **b**

x $\geq$ **0**

with $P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} = \mathbf{b}\ ,\ \mathbf{x} \geq \mathbf{0}\}$.

Let us assume that **A** is an $n \times m$ matrix and that rank(**A**) = m; this implies the existence of a $m \times m$ submatrix **B** of **A** such that det(**B**) $\neq 0$.

***Definition*:** An $m \times m$ submatrix **B** of **A** with det(**B**) $\neq 0$ is called a *basis* of **A**. The variables associated with the columns of **B** are called *basic variables*, while the other variables are called *non-basic variables*. □

Observation

Given a basis **B** of **A**, if the values of the non-basic variables are "fixed", then the values of the basic variables are uniquely determined. Indeed:

A can be written as [**B**, **F**], where **F** is the submatrix of **A** that complements **B** to form **A**;

x can be written as [$\mathbf{x_B}$, $\mathbf{x_F}$], where $\mathbf{x_B}$ is the (sub)vector of basic variables and $\mathbf{x_F}$ is the (sub)vector of non-basic variables;

the system **Ax** = **b** can then be written as $\mathbf{Bx_B} + \mathbf{Fx_F} = \mathbf{b}$, which yields $\mathbf{Bx_B} = \mathbf{b} - \mathbf{Fx_F}$, which yields $\mathbf{Bx_B} = \mathbf{b'}$ (setting $\mathbf{b'} = \mathbf{b} - \mathbf{Fx_F}$, since the values of the non-basic variables are fixed). This is a system in the variables $\mathbf{x_B}$ involving a square matrix **B** with det(**B**) $\neq 0$; thus, according to Cramer's Rule, the system has a unique solution. □

***Definition*:** Given a basis **B** of **A**, if the values of the non-basic variables are set to $\mathbf{x_F} = \mathbf{0}$ (i.e., if the value of each non-basic variable is fixed at 0), then one obtains $\mathbf{x_B} = \mathbf{B^{-1}b}$; the solution [$\mathbf{x_B}$, $\mathbf{x_F}$] = [$\mathbf{B^{-1}b}$, **0**] is termed the *basic solution associated with basis* **B**. In particular, this solution is feasible if [$\mathbf{x_B}$, $\mathbf{x_F}$] $\geq$ **0,** that is, if $\mathbf{B^{-1}b} \geq \mathbf{0}$. □

***Theorem* 3.2:** A point **x** of the polyhedron $P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} = \mathbf{b}\ ,\ \mathbf{x} \geq \mathbf{0}\}$ is a vertex of P if and only if **x** is a basic solution associated with a basis **B** of **A**. □

Thus, Theorem 3.2 provides the desired *algebraic characterization of the vertices*.
In particular, the following corollary is obtained from Theorem 3.1 and Theorem 3.2:

***Corollary* 3.3:** Given an LP problem $\mathbf{min}\{\mathbf{cx} : \mathbf{x} \in P\}$ with $P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} = \mathbf{b}\ ,\ \mathbf{x} \geq \mathbf{0}\}$, if P is a polytope, then there exists at least one optimal solution that coincides with a basic solution associated with a basis **B** of **A**. □

The Simplex method

The Simplex method is a technique for solving a Linear Programming (LP) problem.
The underlying idea is to "visit" only the vertices of the corresponding polyhedron (calculating the corresponding objective function value at each one and selecting the best result).
At the same time, the goal is to visit the fewest possible vertices, given that their number is of the order $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ representing the total number of possible bases according to Theorem 3.2.
To achieve this, the Simplex method employs the following theoretical tools:

--- OPTIMALITY CRITERION: allows for the identification of an optimal solution once it is visited;
--- IMPROVING MOVE: allows for moving from one solution to another (i.e., from one vertex to another) that is no worse than the current one.

The above constitutes the core of the Simplex method. This method was devised by George Dantzig in 1947. There are certain aspects and details that warrant further discussion, such as how to prevent the method from cycling through the same set of vertices (where each is no worse than the last), for which one may refer to the book by Fischetti.

We conclude by highlighting an aspect regarding worst-case computational complexity. The Simplex method has a worst-case computational complexity that is considered inefficient (specifically, non-polynomial); however, the method is considered efficient in practice. Regarding this apparent contradiction, it has been shown that while the Simplex method is theoretically inefficient, it is highly unlikely to be inefficient in practice (this was explained to me by Prof. Luca Moscardelli, who

specifically pointed to the following link: https://en.wikipedia.org/wiki/Smoothed_analysis). On the other hand, there is another method for solving LP problems known as the "Ellipsoid method" which, under non-restrictive assumptions, has a worst-case computational complexity considered efficient; however, the Ellipsoid method proves difficult to use in practice, that is, it is considered practically inefficient. Ultimately, therefore, the Simplex method remains the algorithm commonly used to solve LP problems.

3.1 Duality in LP

Duality is a principle that assigns to every LP problem, let us call it P, the *primal*, another LP problem, let us call it D, the *dual* [of P]. To understand how duality is defined, bearing in mind recalling that any LP problem can be reduced to either canonical form or standard form, we shall examine how duality is defined for these two cases.

- CANONICAL form

min	**cx**	**max**	**ub**
	$\mathbf{Ax} \geq \mathbf{b}$		$\mathbf{A^T u} \leq \mathbf{c}$
	$\mathbf{x} \geq \mathbf{0}$		$\mathbf{u} \geq \mathbf{0}$
	primal		*dual*

- STANDARD form

min	**cx**	**max**	**ub**
	$\mathbf{Ax} = \mathbf{b}$		$\mathbf{A^T u} \leq \mathbf{c}$
	$\mathbf{x} \geq \mathbf{0}$		
	primal		*dual*

Let us look at just one example, in the case of the CANONICAL form, below:

min	$5x_1 + 7x_2$	max	$10u_1 + 40u_2$
	$4x_1 + x_2 \geq 10$		$4u_1 + 2u_2 \leq 5$
	$2x_1 + 3x_2 \geq 40$		$u_1 + 3u_2 \leq 7$
	$x_1, x_2 \geq 0$		$u_1, u_2 \geq 0$
	primal		*dual*

It can be observed, considering only constraints other than non-negativity constraints (i.e., those other than $x_1, x_2 \geq 0$ e $u_1, u_2 \geq 0$), that: the coefficients of the primal objective function correspond to the right-hand side values of the dual, and vice versa; each primal constraint corresponds to a dual variable (specifically, the constraint $4x_1 + x_2 \leq 10$ corresponds to variable u_1, and the constraint $2x_1$

+ $3x_2 \leq 40$ corresponds to variable u_2), and vice versa; and the constraint matrix of the dual is the transpose of the constraint matrix of the primal.

Fundamental Properties of Duality

***Proposition* 3.4:** Let P be a primal problem and D be the dual of P; then the dual of D is P. □

In other words, if Duality associates a problem D with a problem P, then Duality associates problem P with problem D. Put another way, applying Duality repeatedly to a problem P yields problem P again.

Theorem* 3.5 (Strong Duality***):** Consider a primal-dual pair of problems:

min	**cx**	**max**	**ub**
	$\mathbf{Ax} \geq \mathbf{b}$		$\mathbf{A^T u} \leq \mathbf{c}$
	$\mathbf{x} \geq \mathbf{0}$		$\mathbf{u} \geq \mathbf{0}$
	primal		*dual*

If the primal admits a solution, then we have
$\mathbf{min}\{\mathbf{cx} : \mathbf{Ax} = \mathbf{b}\ ,\ \mathbf{x} \geq \mathbf{0}\} = \mathbf{max}\{\mathbf{ub} : \mathbf{A^T u} \leq \mathbf{c}\ ,\ \mathbf{u} \geq \mathbf{0}\}$. □

Theorem* 3.6 (Weak Duality***):** Consider a primal-dual pair of problems:

min	**cx**	**max**	**ub**
	$\mathbf{Ax} \geq \mathbf{b}$		$\mathbf{A^T u} \leq \mathbf{c}$
	$\mathbf{x} \geq \mathbf{0}$		$\mathbf{u} \geq \mathbf{0}$
.	*primal*		*dual*

If both the primal and the dual admit a solution, then for every pair (**x', u'**) where **x'** is a feasible solution to the primal and **u'** is a feasible solution to the dual we have $\mathbf{u'b} \leq \mathbf{cx'}$. □

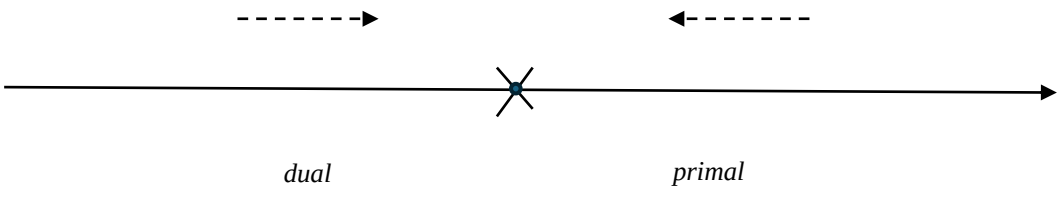
***Corollary* 3.7:** Consider a primal-dual pair of problems:

min	**cx**	**max**	**ub**
	$\mathbf{Ax} \geq \mathbf{b}$		$\mathbf{A^T u} \leq \mathbf{c}$
	$\mathbf{x} \geq \mathbf{0}$		$\mathbf{u} \geq \mathbf{0}$
	primal		*dual*

Then only four cases are possible:

1) both problems have a solution and
   $\mathbf{min}\{\mathbf{cx} : \mathbf{Ax} = \mathbf{b}\ ,\ \mathbf{x} \geq \mathbf{0}\} = \mathbf{max}\{\mathbf{ub} : \mathbf{A^T u} \geq \mathbf{b}\ ,\ \mathbf{u} \geq \mathbf{0}\}$;
2) the primal problem is unbounded and the dual problem is infeasible;
3) the primal problem is infeasible and the dual problem is unbounded;
4) both problems are infeasible. □

Note that the formula $\mathbf{min}\{\mathbf{cx} : \mathbf{Ax} = \mathbf{b}\ ,\ \mathbf{x} \geq \mathbf{0}\} = \mathbf{max}\{\mathbf{ub} : \mathbf{A^T u} \geq \mathbf{b}\ ,\ \mathbf{u} \geq \mathbf{0}\}$ indicates that the optimal values of the two respective solutions are equal, that is, for instance, $\mathbf{min}\{\mathbf{cx} : \mathbf{Ax} = \mathbf{b}\ ,\ \mathbf{x} \geq \mathbf{0}\} = 70$ if and only if $\mathbf{max}\{\mathbf{ub} : \mathbf{A^T u} \geq \mathbf{b}\ ,\ \mathbf{u} \geq \mathbf{0}\} = 70$, while leaving the two respective optimal solutions themselves unrelated to one another. In the figure below, referring again to case (1) of Corollary 3.7, one can visualize, with respect to a value, say X, on the real number line, how the values of the feasible solutions to the dual problem lie to the left of X, the values of the feasible solutions to the primal problem lie to the right of X, and the values of the two respective optimal solutions (for the dual and primal problems) coincide at X.



The Utility of Duality in LP

The utility of duality in Linear Programming (LP) appears to be multifaceted and, in some respects, unexpected. Below, we outline just three possible uses.

:: The first use relates to the Strong Duality Theorem: suppose a decision-maker wishes to solve the primal problem P; if solving P is difficult, then [based on this theorem] one can attempt to solve the dual problem D. In particular, considering the principles of duality, the dual problem D might, for instance, have fewer variables than the primal problem P.

:: The second use relates to the Weak Duality Theorem: suppose a decision-maker wishes to solve the primal problem P; if solving P is difficult, then [based on this theorem] the decision-maker can obtain an *upper bound* on the value of the optimal solution to P by calculating the value of any feasible solution to the dual problem D, or a *lower bound* on the value of the optimal solution to P by calculating the value of any feasible solution to the dual problem D.

:: The third use is known as the "economic interpretation of duality". In terms of application, when we model a real-world problem as an LP problem, say P, each [decision] variable in P carries a specific meaning. The "economic interpretation of duality" highlights the possibility of assigning meaning to the variables of the dual problem D associated with P, or, more precisely, to the value that each variable of the dual problem D assumes in the optimal solution.

Examples of the economic interpretation of duality

1) *Resource combination*

A winery wishes to produce two types of wine: a table wine and a dessert wine. The profit the company derives from producing one unit of table wine is 3, while the profit from producing one unit of dessert wine is 7. Production requires a specific combination of two types of grapes: let us call them type A and type B, respectively. Producing one unit of table wine requires 3 units of type A grapes and 2 units of type B grapes. Producing one unit of dessert wine requires 1 unit of type A grapes and 4 units of type B grapes. Finally, the company has 1000 units of type A grapes and 400 units of type B grapes available. The problem is to determine the quantities of table wine and dessert wine to produce in order to maximize total profit.

Decision variables: x_1, x_2
x_1 = quantity of table wine produced
x_2 = quantity of dessert wine produced

$$
\begin{aligned}
(1) \qquad \max \quad & 3x_1 + 7x_2 \\
& 3x_1 + x_2 \leq 1000 \\
& 2x_1 + 4x_2 \leq 400 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

The optimal solution is $x^*_1 = 0$, $x^*_2 = 100$, with a value of 700.

Consider the dual problem of problem (1) below.

$$
\begin{aligned}
(2) \qquad \min \quad & 1000\, y_1 + 400\, y_2 \\
& 3y_1 + 2y_2 \geq 3 \\
& y_1 + 4y_2 \geq 7 \\
& y_1, y_2 \geq 0
\end{aligned}
$$

The optimal solution is $y^*_1 = 0$, $y^*_2 = 7/4$, with a value of 700.

Let y_j^* be a component of the optimal solution to the dual problem. Recalling that each constraint of the primal problem is associated with a variable of the dual problem, let b_j be the right-hand side value of the primal constraint associated with variable y_j, that is, the availability of resource j. Specifically, $b_j y_j$ is a term in the dual problem's objective function; at the optimum, its value is $b_j y^*_j$. Now, assume b_j can be varied: let it become $b_j + \Delta b_j$.

• If $b_j + \Delta b_j > 0$ (representing a purchase of resource j), then by the Strong Duality Theorem, the optimal profit changes by $y^*_j \Delta b_j \geq 0$ (in the case of a free acquisition); however, in general, it changes by $y^*_j \Delta b_j - y'_j \Delta b_j$, where y'_j is the unit purchase price of resource j. Therefore, such a purchase is profitable only if $y^*_j \Delta b_j - y'_j \Delta b_j > 0$, that is, only if $y'_j < y^*_j$.

• If $b_j + \Delta b_j < 0$ (representing a sale of resource j), then by the Strong Duality Theorem, the optimal profit changes by $y^*_j \Delta b_j \leq 0$ (in the case of a sale at zero price); however, in general, it changes by $y^*_j \Delta b_j + y'_j |\Delta b_j| = y^*_j \Delta b_j - y'_j \Delta b_j$, where y'_j is the unit selling price of resource j. Therefore, such a sale is profitable only if $y^*_j \Delta b_j - y'_j \Delta b_j > 0$, that is, only if $y'_j > y^*_j$ (given that $\Delta b_j < 0$).

Thus, the optimal solution to the dual problem $(y^*_1, \ldots, y^*_m)$ represents the intrinsic value of each resource j, for $j = 1, \ldots, m$. The value y_j^* is also called as the *shadow price* of resource j in this sense:

• it is advantageous to <u>purchase</u> resource j (within a certain quantity limit) if purchased at a unit price <u>below the shadow price y_j^*</u>;

• it is advantageous to <u>sell</u> resource j (within a certain quantity limit) if sold at a unit price <u>above the shadow price y_j^*</u>.

<u>Observation</u>:

The quantity limits within which it is advantageous to purchase or sell a resource depend on the fact that the optimal basic solution may change due to variations in resource availability, specifically, the corresponding right-hand side value. These limits can be calculated using simple operations found in sensitivity (or post-optimality) analysis and are computed by any software for LP problems.

Returning to our example, we have that:

y^*_1 = shadow price of grapes A

y^*_2 = shadow price of grapes B

Consider, for example, the shadow price $y^*_2 = 7/4$ for grapes B.

It is profitable to purchase this resource (within certain quantity limits) at a unit price below 7/4, as this guarantees an increase in total profit.
It is profitable to sell this resource (within certain quantity limits) at a unit price above 7/4, as this guarantees an increase in total profit.
It makes no difference whether one purchases or sells this resource (within certain quantity limits) at a unit price of 7/4, as this does not change the total profit.

2) *Resource decomposition*

A person wants to go on a diet. Specifically, they need to consume two types of nutrients, say proteins and vitamins, which can be obtained by purchasing three types of food: fruit, milk, and eggs. In detail:
1 unit of fruit contains: 0 units of protein, 7 units of vitamins;
1 unit of milk contains: 2 units of protein, 3 units of vitamins;
1 unit of eggs contains: 5 units of protein, 1 unit of vitamins.
The unit cost of fruit, milk, and eggs is 10, 20, and 10, respectively.
The diet requires the consumption of at least 15 units of protein and 25 units of vitamins.
The problem is to determine the quantities of fruit, milk, and eggs to purchase, satisfying the aforementioned requirements, so as to minimize the total cost.

Decision variables: x_F, x_L, x_U
x_F = quantity of fruit to purchase
x_L = quantity of milk to purchase
x_U = quantity of eggs to purchase

$$
\begin{aligned}
(1) \qquad \min \quad & 10x_F + 20x_L + 10x_U \\
& 0x_F + 2x_L + 5x_U \geq 15 \\
& 7x_F + 3x_L + 1x_U \geq 25 \\
& x_F, x_L, x_U \geq 0
\end{aligned}
$$

The optimal solution is $x^*_F = 3{,}14$, $x^*_L = 0$, $x^*_U = 3$, with a value of 61.42.

Consider below the dual problem of problem (1).

$$
(2) \qquad \max \quad 15\, y_1 + 25\, y_2
$$

$$0y_1 + 7y_2 \leq 10$$
$$2y_1 + 3y_2 \leq 20$$
$$5y_1 + 1y_2 \leq 10$$
$$y_1, y_2 \geq 0$$

The optimal solution is $y^*_1 = 1{,}71$ and $y^*_2 = 1.42$, with a value of 61.42.

Using an argument similar to that of Example 1, it can be verified that the optimal solution to the dual problem ($y^*_1, \ldots, y^*_m$) represents the intrinsic value of each product j, for $j = 1, \ldots, m$ (proteins and vitamins). The value y_j^* is also called as the *shadow price* of product j in this sense:
• it is advantageous to purchase product j (within a certain quantity limit) if purchased at a unit price below the shadow price y_j^*.

Observation:
The quantity limits within which it is advantageous to purchase a resource depend on the fact that the optimal basic solution may change due to variations in resource availability, specifically, the corresponding right-hand side value. These limits can be calculated using simple operations found in sensitivity (or post-optimality) analysis and are computed by any software for LP problems.

Returning to our example, we have that:
y^*_1 = shadow price of proteins
y^*_2 = shadow price of vitamins

Consider, for example, the shadow price $y^*_2 = 1.42$ of vitamins.
It is advantageous to purchase this product, namely, vitamins, such as in the form of a dietary supplement (within certain quantity limits) at a unit price below 1.42, since doing so ensures a reduction in total expenditure.
It makes no difference whether one purchases this product (within a certain quantity limit) at a unit price of 1.42, since doing so does not change the total expenditure.

4. Essentials of Integer Linear Programming (ILP)

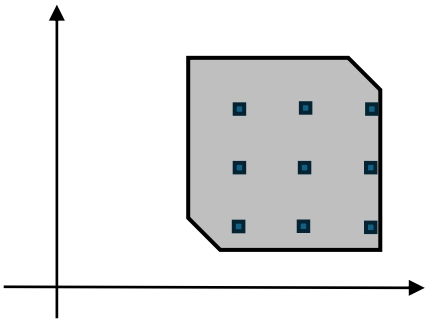
An Integer Linear Programming problem, for short ILP, can be understood as a variant of a Linear Programming (LP) problem in which all the variables are constrained to take on integer values.

An ILP problem, in CANONICAL form, is defined as follows:

$$
\begin{aligned}
Z_{PLI} = \quad & \textbf{min} \quad \mathbf{cx} \\
& \mathbf{Ax} \geq \mathbf{b} \\
& \mathbf{x} \geq \mathbf{0} \\
& \mathbf{x} \ \text{INTEGER}
\end{aligned}
$$

where Z_{PLI} represents the value of the problem's optimal solution (i.e., the value the objective function takes at the optimal solution).
In cases where only some variables are constrained to take integer values, the term Mixed-Integer Linear Programming is used.

Let **P** denote the ILP problem formulated as above; specifically, let $P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} \geq \mathbf{b}\ ,\ \mathbf{x} \geq \mathbf{0}\}$ be the *polyhedron associated with problem* **P**. Then (in terms of graphical visualization), the set of feasible solutions for problem **P** is given by all the integer points in P, that is, by $P \cap \mathbb{Z}^n$. In the figure below, P is the polyhedron shown in gray, while $P \cap \mathbb{Z}^n$ is the set of small squares representing the integer solutions within P.



Thus, it can be observed that problem **P** may admit various formulations based on different polyhedra, specifically, on any polyhedron P' such that $P \cap \mathbb{Z}^n = P' \cap \mathbb{Z}^n$; we denote the collection of such polyhedra as the *set of polyhedra associated with problem* **P**.
In the following, we consider ILP problems whose associated polyhedra are polytopes, that is, ILP problems that admit a finite optimal solution (if one exists).

An initial approach

An initial approach to attempting to solve an ILP problem **P**, as defined above, is the following procedure: eliminate the integrality constraints and then solve the resulting LP problem – referred to as the *continuous relaxation* of the original problem and shown below:

$$Z_{PL} = \mathbf{min}\ \mathbf{cx}$$
$$\mathbf{Ax} \geq \mathbf{b}$$
$$\mathbf{x} \geq \mathbf{0}$$

where Z_{PL} represents the optimal solution value of the continuous relaxation (i.e., the value the objective function takes at the optimal solution of the continuous relaxation).

Observation
The relationship between the ILP problem **P** and its continuous relaxation is characterized by the following two points:
• $Z_{PL} \leq Z_{PLI}$;
indeed, recalling the notation $P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$, $P \supseteq P \cap \mathbb{Z}^n$,
thus yielding $Z_{PL} = \mathbf{min}\{\mathbf{cx} : \mathbf{x} \in P\} \leq \mathbf{min}\{\mathbf{cx} : \mathbf{x} \in P \cap \mathbb{Z}^n\} = Z_{PLI}$.
• if the optimal solution $\mathbf{x}^*$ of the continuous relaxation of problem **P** is integer (i.e., all its components are integers), then $\mathbf{x}^*$ is also the optimal solution for problem **P**, that is, $\mathbf{cx}^* = Z_{PLI}$;
indeed, on the one hand, we have $\mathbf{cx}^* = Z_{PL} \leq Z_{PLI}$; on the other hand, we have $\mathbf{cx}^* \geq Z_{PLI}$, since $\mathbf{x}^* \in P \cap \mathbb{Z}^n$. □

This procedure for attempting to solve an ILP problem **P**, as defined above, cannot guarantee finding an optimal solution for problem **P**; in particular, the vertices of a polyhedron P associated with problem P may be non-integer, meaning the optimal solution to the continuous relaxation of problem P may also be non-integer [recalling the results introduced for Linear Programming]. However, the following theoretical result holds:

***Theorem* 4.1:** Given an ILP problem **P**, as defined above, there exists a polytope $P' = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A'x} \geq \mathbf{b'}, \mathbf{x} \geq \mathbf{0}\}$ associated with problem **P** such that:
:: P' has only integer vertices;
:: $\mathbf{min}\{\mathbf{cx} : \mathbf{x} \in P\} = \mathbf{min}\{\mathbf{cx} : \mathbf{x} \in P'\}$ for every vector **c** (of rational numbers). □

In other words, Theorem 4.1 states that, in theory, there exists a polytope associated with the ILP problem P that has only integer vertices, an *ideal* associated polytope, so to speak. This implies that, in theory, every ILP problem is equivalent to an LP problem. However, in practice, this *ideal* associated polytope may be difficult to determine and/or may define a huge system of linear inequalities.

Total Unimodularity

This section introduces some results that allow for the identification of certain ILP problems for which the associated *ideal* polytope is easily determined.

Notation: given an $n \times m$ matrix $\mathbf{A}$, we denote its elements by $\{a_{ij}\}$ (for $i = 1, \ldots, n$ and $j = 1, \ldots, m$) and say that $\mathbf{A}$ is *integer* if all its elements are integers; furthermore, we recall that an $n \times m$ matrix $\mathbf{A}$ is said to be *square* if $n = m$.

***Definition*:** An integer matrix $\mathbf{A}$ of size $n \times m$ is said to be *totally unimodular* (*TUM* for short) if $\det(\mathbf{Q}) \in \{-1, 0, 1\}$ for every square submatrix $\mathbf{Q}$ of $\mathbf{A}$. □

Note that, by definition, if $\mathbf{A}$ is a TUM matrix, then for every element a_{ij} of $\mathbf{A}$, $a_{ij} \in \{-1, 0, 1\}$; indeed, every element of $\mathbf{A}$ is a 1×1 square submatrix.

***Theorem* 4.2:** Let $\mathbf{A}$ be an $n \times m$ TUM matrix and let $\mathbf{b}$ be an integer vector with m components. Then both the polyhedron $P^{=} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} = \mathbf{b}\ ,\ \mathbf{x} \geq \mathbf{0}\}$ and the polyhedron $P^{\geq} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} \geq \mathbf{b}\ ,\ \mathbf{x} \geq \mathbf{0}\}$ have only integer vertices.

The theorem above then implies the following corollary of a general nature.

***Corollary* 4.3:** Given an ILP problem **P** as follows

min **cx**
Ax $\geq$ **b**
x $\geq$ **0**
x INTEGER

if $\mathbf{A}$ is a TUM matrix and $\mathbf{b}$ is an integer vector, then the constraint "**x** INTEGER" can be dropped; that is, the ILP problem **P** can be solved as an LP problem by solving its continuous relaxation. □

The corollary above thus implies the following corollary of an applied nature, with reference to certain problems we introduced as ILP problems, thereby highlighting that these problems are ultimately *easy*, in accordance with Appendix 1.

***Corollary* 4.4:** The following problems can be solved as LP problems:
Assignment,
Matching,
Transportation,
Minimum-cost path,
Maximum flow. □

We will examine the proof of Corollary 4.4, using examples, after introducing some properties of TUM matrices.

Let us now return to Corollary 4.3, which highlights that – consistent with Appendix 1 – it is worth studying how to recognize TUM matrices; indeed, this makes it possible to determine whether an optimization problem, even if formulated as an ILP (and thus seemingly *hard*), can ultimately be formulated as an LP (and thus be *easy*).

As a preliminary step, let us consider the following question:
Given a matrix **A**, how can one determine [efficiently] whether **A** is TUM?
A natural [though inefficient] method would be to enumerate all possible square submatrices **Q** of **A** and then check whether $\det(\mathbf{Q}) \in \{-1, 0, 1\}$. Efficient methods for determining whether a matrix is TUM do exist; however, these methods are sophisticated, so in the following we will look at some simple partial results (either necessary or sufficient conditions) that are nonetheless very useful.

A necessary condition (as previously noted) is that for every element a_{ij} of **A**, $a_{ij} \in \{-1, 0, 1\}$; indeed, every element of **A** is a 1×1 square submatrix. However, this necessary condition is not sufficient (one can easily construct matrices whose elements all belong to $\{-1, 0, 1\}$ but which are not TUM).

A sufficient condition is provided by the following theorem.

***Theorem* 4.5:** Let **A** be a matrix such that each element a_{ij} of **A** satisfies $a_{ij} \in \{-1, 0, 1\}$.
A is TUM if the following two conditions hold:
:: every column of **A** has at most two non-zero elements;
:: there exists a partition $\{I_1, I_2\}$ of the set of rows of **A** such that, for every column with exactly two non-zero elements, these two non-zero elements belong to the same set I_k (for $k \in \{1, 2\}$) if and only if they have the same sign. □

However, this sufficient condition is not necessary (one can easily construct matrices that are TUM, and thus have all elements in $\{-1, 0, 1\}$, but do not satisfy at least one of the two conditions).

The following proposition provides other useful properties of TUM matrices.

***Proposition* 4.6:** Let **A** be an integer matrix. The following statements are equivalent:
• **A** is TUM;
• $\mathbf{A}^T$ is TUM;
• the matrix obtained from **A** by permuting and/or changing the signs of some columns and/or rows is TUM;
• the matrix obtained from **A** by adding a column or a row consisting entirely of zeros, except for at most one element equal to ±1 (i.e., in $\{-1, 1\}$), is TUM. □

In the following, we attempt to demonstrate Corollary 4.4 using only examples; specifically, we will address only the Transportation problem and the Minimum-Cost Path problem. The Assignment problem and the Matching problem can be handled similarly to the Transportation problem, while the Maximum Flow problem can be handled similarly to the Minimum-Cost Path problem.

Example: Transportation problem

There are s source locations and t destination locations. Each source location $i \in \{1, \ldots, s\}$ has a supply of $d_i \geq 0$ units of a certain type of good, and each destination $j \in \{1, \ldots, t\}$ requires at least $r_j \geq 0$ units of the same good. For each pair (i, j), where $i \in \{1, \ldots, s\}$ and $j \in \{1, \ldots, t\}$, the unit transportation cost c_{ij} and the maximum transportable quantity q_{ij} are also specified. Plan the transportation so as to minimize the total cost.

Decision variables: x_{ij} per $i = 1, \ldots, s$ e $j = 1, \ldots, t$;
x_{ij} indicates the quantity transported from source i to destination j.

Model: min $\sum_{i=1}^{s}\sum_{j=1}^{t} c_{ij}x_{ij}$

$\sum_{j=1}^{t} x_{ij} \leq d_i$ for $i = 1, \ldots, s$ (availability constraints)

$\sum_{i=1}^{s} x_{ij} \geq r_j$ for $j = 1, \ldots, t$ (request constraints)

$x_{ij} \leq q_{ij}$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$ (capacity constraint)

$x_{ij} \geq 0$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$

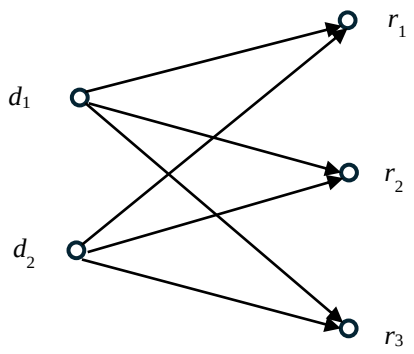
x_{ij} intero for $i = 1, \ldots, s$ and $j = 1, \ldots, t$ (if necessary)

Note that the fact that the variables are constrained to be integers implies that the units of the good to be transported are indivisible; thus, it can be assumed that the right-hand side values d_i and r_j are integers.

Let us assume (for now) that $q_{ij} = \infty$.

Thus, the matrix **A** will involve only the supply and demand constraints.

Consider the following instance of the problem.



costi c_{ij}

SOURCES DESTINATIONS

Let us now explicitly examine the inequalities associated with matrix **A**.

$$
\begin{aligned}
x_{11} + x_{12} + x_{13} &\leq d_1 \\
x_{21} + x_{22} + x_{23} &\leq d_2 \\
x_{11} + x_{12} &\geq r_1 \\
x_{12} + x_{22} &\geq r_2 \\
x_{13} + x_{23} &\geq r_3
\end{aligned}
$$

They can be displayed as follows, so as to (better) highlight matrix **A**.

$$
\begin{aligned}
x_{11} + x_{12} + x_{13} &\leq d_1 \\
x_{21} + x_{22} + x_{23} &\leq d_2 \\
x_{11} + x_{21} &\geq r_1 \\
x_{12} + x_{22} &\geq r_2 \\
x_{13} + x_{23} &\geq r_3
\end{aligned}
$$

In particular, with regard to the first two inequalities, we multiply both “sides” of these inequalities by −1, so as to prepare the formulation in accordance with Corollary 4.3:

$$
\begin{aligned}
-x_{11} - x_{12} - x_{13} &\geq d_1 \\
-x_{21} - x_{22} - x_{23} &\geq d_2 \\
x_{11} + x_{21} &\geq r_1 \\
x_{12} + x_{22} &\geq r_2 \\
x_{13} + x_{23} &\geq r_3
\end{aligned}
$$

At this point, with reference to Corollary 4.3, the matrix **A** is as follows:

$$\begin{array}{rrrrrr} -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}$$

We now apply Theorem 4.5 to verify that A is TUM. Note that: for any element a_{ij} of **A**, $a_{ij} \in \{-1, 0, 1\}$; each column of **A** has at most two non-zero elements; and there exists a partition $\{I_1, I_2\}$ of the set of rows R of **A** as specified in Theorem 4.4, specifically, this partition is given by $I_1 = R$ and $I_2 = \varnothing$.

Finally, we observe that even assuming $q_{ij} \geq 0$, that is, even considering the constraints $x_{ij} \leq q_{ij}$ ($i = 1, \ldots, s$, e $j = 1, \ldots, t$), the thus-extended matrix **A*** is TUM: indeed, the constraints $x_{ij} \leq q_{ij}$ ($i = 1, \ldots, s$, e $j = 1, \ldots, t$) extend the original matrix **A** with an identity matrix [having diagonal elements equal to 1 and all other elements equal to 0], such that Proposition 4.6 can be applied.

This proves Corollary 4.4 for the transportation problem, given that **A** is TUM and the right-hand side values (such as r_j) are integers [i.e., given that the "vector **b**" is integer-valued].

Example: Minimum-Cost Path problem
Let $G = (V, A)$ be a directed graph, and let s and t be two vertices of G. A cost $q_{ij} \geq 0$ is associated with each arc $(i, j) \in A$. The cost of a path from s to t is the sum of the costs of the arcs forming the path. Determine a path from s to t with minimum cost.

Decision variables: x_{ij} per ogni arco $(i, j) \in A$;
x_{ij} takes the value of 1 if the arc (i, j) is in the path
x_{ij} takes the value of 1 if the arc (i, j) is not in the path

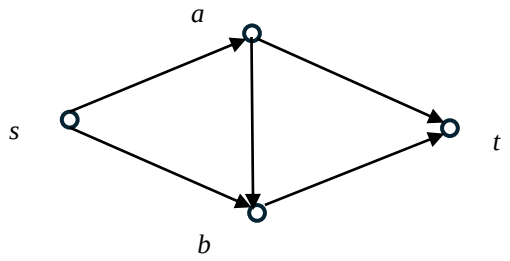
Model: min $\sum_{(i,j)\in A} q_{ij} x_{ij}$

$$\sum_{(s,j)\in A} x_{sj} = 1$$

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = 0 \quad \text{for any vertex } i \neq s, t \text{ di } G$$

$x_{ij} \geq 0$ for any $(i, j) \in A$
$x_{ij} \leq 1$ for any $(i, j) \in A$
x_{ij} integer for any $(i, j) \in A$

Let us consider the following instance of the problem.



$$
\begin{aligned}
x_{sa} + x_{sb} &= 1 \\
-x_{sa} + x_{ab} + x_{at} &= 0 \\
-x_{sb} + x_{bt} &= 0
\end{aligned}
$$

They can be displayed as follows, so as to (better) highlight matrix **A**.

$$
\begin{aligned}
x_{sa} + x_{sb} &= 1 \\
-x_{sa} + x_{ab} + x_{at} &= 0 \\
-x_{sb} + x_{bt} &= 0
\end{aligned}
$$

At this point, with reference to Corollary 4.3, the matrix **A** is as follows:

$$
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 \\
-1 & 0 & 1 & 1 & 0 \\
0 & -1 & 0 & 0 & 1
\end{bmatrix}
$$

We now apply Theorem 4.5 to verify that A is TUM. Note that: for any element a_{ij} of **A**, $a_{ij} \in \{-1, 0, 1\}$; each column of **A** has at most two non-zero elements; and there exists a partition $\{I_1, I_2\}$ of the set of rows R of **A** as specified in Theorem 4.5, specifically, this partition is given (once again) by $I_1 = R$ and $I_2 = \varnothing$.

This proves Corollary 4.4 for the minimum-cost path problem, given that **A** is TUM and the right-hand side values are integers.

The "Branch and Bound" method

The "Branch and Bound" method is a technique for solving an Integer Linear Programming (ILP) problem. Two preliminary points should be noted: (1) this is not the only method for solving ILP problems; notably, [Fischetti] describes the "Cutting Plane" method, namely, a geometric approach that involves iteratively solving the continuous relaxation of the current problem, that at each iteration [specifically when the optimal solution, say **x**, to the continuous relaxation is non-integer] adds a new

linear inequality, known as a "cutting plane", in order to exclude solution **x** from the next iteration without eliminating any feasible solutions; (2) the “Branch and Bound” method is not geometric; rather, it can be described as a common-sense approach based on intuitive observations that guides the search for an optimal solution in a context where the only guaranteed strategy would otherwise be an exhaustive search.

We denote an ILP problem of the form **min**{**cx** : **x** $\in$ *S*} [where *S* is the set of feasible solutions, that is, a set of integer points within a polyhedron] as (*S*, **c**), and we assume that *S* is finite.

The "Branch and Bound" method consists of the *interaction* of two phases: the BRANCH phase and the BOUND phase.

• The BRANCH phase is based on the following observation:
if the problem (*S*, **c**) is difficult to solve, *a partition of S*, say {S_1, …, S_t} with $t \geq 2$, *is defined*; by the definition of a partition, the sets S_1, …, S_t are mutually disjoint, yet their union is the set *S*; Specifically, let:
z_i be the optimal solution value for the sub-problem (S_i, **c**), for $i \in \{1, \ldots, t\}$, and
z be the optimal solution value for the problem (*S*, **c**);
then:

$$z = \min \{z_i : i \in \{1, \ldots, t\}\}.$$

This process can be iterated for each (S_i, **c**). However, it should be noted that excessive iteration could lead to an unmanageable number of sub-problems. To avoid this, the BOUND phase, described below, proves useful.

• The BOUND phase is based on the following observation:
let $\check{z}$ be the value of the best solution found so far, that is, $\check{z}$ is the *current optimum*;
for each subproblem (S_i, **c**), a *lower bound* L_i is calculated for the value of the optimal solution to (S_i, **c**), such that $L_i \leq z_i$; note that:
if $\check{z} \leq L_i$, then the subproblem (S_i, **c**) *can be closed*;
indeed, given that $\check{z} \leq L_i \leq z_i$, it follows that $\check{z} \leq z_i$; this means that the subproblem (S_i, **c**) contains no solutions with a value lower (and thus better) than $\check{z}$, the current optimum.

The effectiveness of the “Branch and Bound” method depends precisely on:
branching strategies, that is, the way subproblems are successively partitioned;
solution strategies, that is, the way *lower bounds* are successively calculated.

An example of a branching strategy is as follows: assume that the variables of the subproblem (S_i, **c**), namely, x_1, …, x_n , are binary (i.e., taking values in {0, 1}); then, a possible partition of S_i can be obtained by fixing a variable, say x_j, and defining two subsets of S as follows: {**x** $\in$ S_i : = 1} and {**x** $\in$ S_i : = 0}.

One example of a solution strategy – apparently the most common one – is as follows: the continuous relaxation of the subproblem (S_i, **c**) is solved; then, as noted when introducing the *continuous relaxation* of an ILP problem, the value of the optimal solution **x** to the continuous relaxation of (S_i, **c**) serves as a *lower bound* on the optimal solution value of (S_i, **c**); furthermore, if **x** is integer, then **x** is also the optimal solution to (S_i, **c**).

5. Five specific problems (with specific solution methods)

In this chapter, we will focus on the following five problems:
--- Minimum-Cost Path problem,
--- Project Planning problem,
--- Maximum Flow problem,
--- Production Planning problem,
--- Facility Location problem.

With reference to Appendix 1:

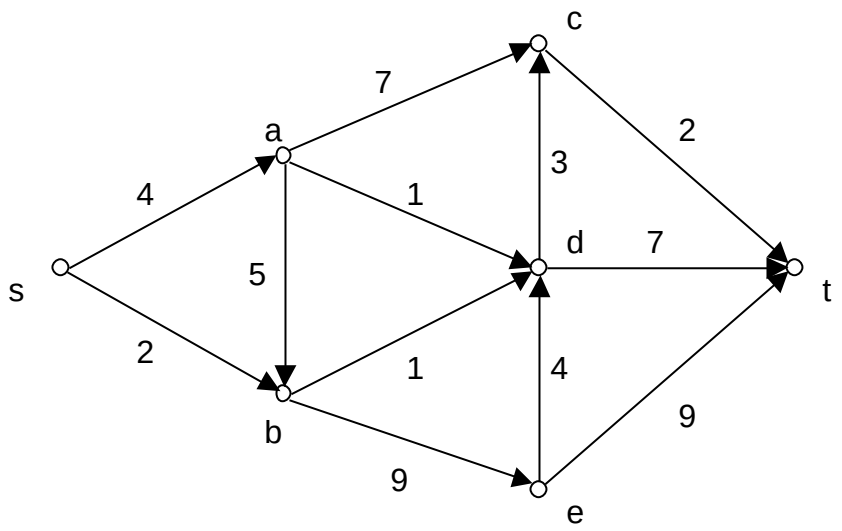
:: the first four problems are *easy* problems; specifically, they can be formulated as Linear Programming (LP) problems (regarding the fourth, this holds true under certain assumptions), so in general, we already know how to solve them using the Simplex method; however, for each of these four problems, we introduce specific properties that "characterize" optimal solutions and allow for the definition of "ad hoc" solution methods (which are faster than the Simplex method);

:: the fifth problem is a *difficult* problem; specifically, it can be formulated as an Integer Linear Programming (ILP) problem, so in general, we already know how to solve it using the "Branch and Bound" method; however, we introduce a "heuristic method" to find a solution that is presumably good, though not necessarily optimal.

5.1 Minimum-Cost Path problem

Problem definition:

Let $G = (V, A)$ be a directed graph, and let s and t be two vertices of G. A cost $q_{ij} \geq 0$ is associated with each arc $(i, j) \in A$. The cost of a path from s to t is the sum of the costs of the arcs forming the path. The problem is to determine a minimum-cost path from s to t.



Observations

:: Note that this problem can be formulated as an ILP problem (see Chapter 2, Example C1).

:: Note that this problem, even if initially formulated as an ILP problem, can be formulated (and thus treated) as an LP problem; indeed, as we saw regarding TUM matrices, it is possible to eliminate the integrality constraints [from the ILP formulation] and then solve the continuous relaxation [of that formulation]; in particular, the Shortest Path problem can be solved using the Simplex method.

:: The ILP formulation (in general) can be adapted to variants such as the following.

For example: a time $t_{ij} \geq 0$ can be associated with each arc $(i, j) \in A$, and a maximum time T can be set for traveling from s to t; the time of a path from s to t is the sum of the times of the arcs forming the path; the problem then becomes: determine a minimum-cost path from s to t with a total time not exceeding T; to model this variant, one simply adds the constraint $\sum_{(i,j)\in A} t_{ij}\, x_{ij} \leq T$ [to the ILP formulation]. □

To introduce an "ad hoc" solution method for the Minimum-Cost Path problem – in addition to the Simplex method mentioned above – we consider two properties.

For the sake of brevity, we refer to an optimal solution to the Minimum-Cost Path problem as an *optimal path*. The following result holds:

***Theorem* 5.1.1:** A path is optimal if and only if it is composed of optimal sub-paths. □

This result can be understood by visualizing, for instance, the optimal path indicated by a navigation system for a journey from Pescara to Rome; such a path implicitly defines the optimal paths from Pescara to all the intermediate locations (such as Avezzano, Carsoli, etc.) along that route. The proof of the theorem above is quite intuitive, relying on a proof by contradiction and the fact that $q_{ij} \geq 0$. At this point, one might surmise the existence of a combinatorial method for calculating an optimal path. The following theorem describes the core of precisely such a combinatorial method.

***Theorem* 5.1.2:** Let L_v be the cost of the minimum-cost path from s to v for each $v \in V$, and let $\{W, Z\}$ be a partition of V such that $L_w \leq L_z$ for every $(w, z) \in W \times Z$. Let $(w^*,z^*) = \text{argmin}\ \{L_w + q_{wz} : (w, z) \in W \times Z\}$. Then $L_{z^*} \leq L_z$ for every $z \in Z$, meaning z^* can be added to W with $L_{z^*} = L_{w^*} + q_{w^*z^*}$. □

Dijkstra's algorithm

Input: $G = (V, E)$ with $|V| = n$; $s, t \in V$; $q_{ij} \geq 0$ for every $(i,j) \in E$.

Output: a minimum-cost path from s to t

Initialization

:: $W := \{s\}$

:: $q_{ij} = \infty$, per ogni $(i,j) \notin E$.

:: construct a vector α with n components, one for each vertex v of G, as follows:

$\alpha(s) = 0$,

$\alpha(v) = q_{sv}$, for any $v \in V \setminus S$

Generic step:

until $t \in W$ do:

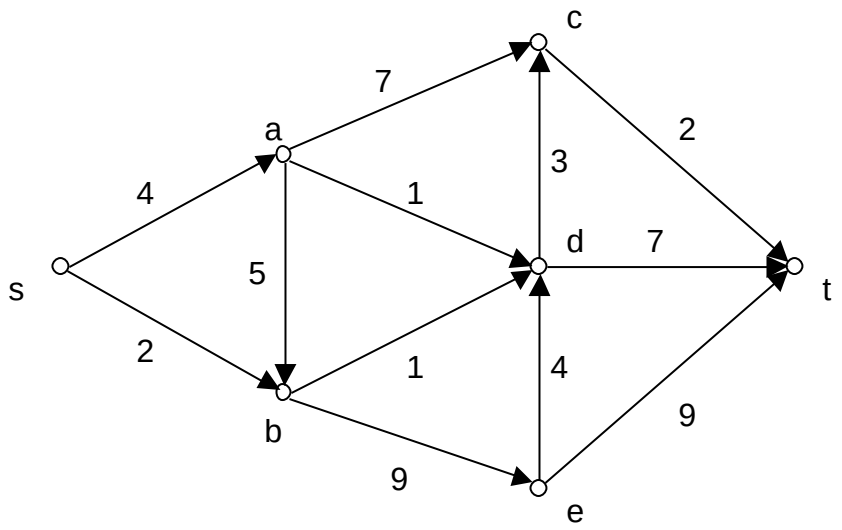
:: choose a vertex $z^* \in V \setminus W$ such that $\alpha(z^*)$ is minimum;

:: insert z^* into W;

:: update the vector α by setting $\alpha(z) := \min \{\alpha(z) , \alpha(z^*) + q_{z^*z} \}$ for every $z \in V \setminus W$. □

Solved Exercise

Calculate a minimum-cost path from *s* to *t* using Dijkstra's algorithm in the following directed graph $G = (V, A)$, where the associated cost $q_{ij} \geq 0$ is indicated for each arc (i, j).



Solution

The problem is to determine a minimum-cost path from *s* to *t*, where the cost of a path is the sum of the costs associated with the edges of the path (for example, the path $s \to a \to d \to t$ has a cost of 4 + 1 + 7 = 12). The solution method is based on Theorem 5.1.2.

The method iteratively constructs this set *W* until *t* is included in *W*.

Initialization

$W := \{s\}$. Construct the vector α, which has a component for each vertex *v* of the graph, with $\alpha(v)$ equal to: q_{sv} if the edge *sv* exists, and ∞ otherwise.

s	*a*	*b*	*c*	*d*	*e*	*t*
0	4	2	∞	∞	∞	∞

General step

• choose a vertex $z^* \in V \setminus W$ such that $\alpha(z^*)$ is minimum;
• add z^* to *W*;
• update the vector α by setting $\alpha(z) := \min \{\alpha(z) , \alpha(z^*) + q_{z^*z} \}$ for every $z \in V \setminus W$. □

Step 1

- choose b
- $W := W \cup \{b\}$
- update the vector α as follows

s	*a*	***b***	*c*	*d*	*e*	*t*
0	4	**2**	∞	3	11	∞

Step 2

- choose d
- $W := W \cup \{d\}$
- update the vector α as follows

s	*a*	***b***	*c*	***d***	*e*	*t*
0	4	**2**	6	**3**	11	10

Step 3

- choose a
- $W := W \cup \{a\}$
- update the vector α as follows

s	***a***	***b***	*c*	***d***	*e*	*t*
0	**4**	**2**	6	**3**	11	10

Step 4

- choose c
- $W := W \cup \{c\}$
- update the vector α as follows

s	***a***	***b***	***c***	***d***	*e*	*t*
0	**4**	**2**	**6**	**3**	11	8

Step 5

- choose t
- $W := W \cup \{t\}$
- The method terminates as t has been inserted into W.

The value $z(t) = 8$ represents the cost of a minimum-cost path from s to t. To identify this path, one can work backwards from t to s as follows:

$\alpha(t)$ was last updated when c entered W;

$\alpha(c)$ was last updated when d entered W;

$\alpha(d)$ was last updated when b entered W;

$\alpha(b)$ was last updated when s entered W;

This yields the path: $s \rightarrow b \rightarrow d \rightarrow c \rightarrow t$ (with cost = 8)

Concluding remarks on applications

1) The Minimum Cost Path problem has many practical and theoretical applications; we list just a few below:

--- Dynamic Programming: Theorem 5.1.1 [i.e., a path is optimal if and only if it is composed of optimal sub-paths; more generally, a solution is optimal if and only if it is composed of optimal sub-solutions] is a result that characterizes so-called Dynamic Programming problems; thus, the Minimum Cost Path problem is a Dynamic Programming problem. In particular, it plays a special role in this context, in the sense that any Dynamic Programming problem can be viewed as an instance of the Minimum Cost Path problem;

--- road networks: the problem of finding a minimum-cost (or minimum-time) path between two points in a road network is clearly an instance of the Minimum Cost Path problem; thus, a GPS navigator, in its basic function, solves precisely this problem;

--- location: finding the *center* of a network can mean finding a point in the network that minimizes the sum of the shortest paths from that point to every other point in the network;

--- sub-procedure: the Minimum Cost Path problem can be used as a sub-procedure in solving other problems—for example, the Traveling Salesperson Problem (or "Commission Problem") introduced at the beginning of these notes.

2) What happens if the condition $q_{ij} \geq 0$ does not hold for the Minimum Cost Path problem, that is, if the edge costs in graph G can be negative? Referring again to the [Fischetti] book, briefly put: if G

contains a “negative cycle” (i.e., a cycle where the sum of the edge costs is negative), then the problem is unbounded; If *G* does not have a *“*negative cycle”, an optimal path can still be found efficiently using the Floyd-Warshall algorithm [which, moreover, simultaneously finds an optimal path between every pair of vertices in the graph *G*]. □

5.2 Project Scheduling Problem

Problem definition:

A *project* consists of a set of n activities A_i (for $i \in \{1, \ldots, n\}$), each with a known duration $d_i \geq 0$; in particular, precedence relationships of the form $A_i < A_j$ are specified for certain activities, indicating that the completion time of activity A_i must precede the start time of activity A_j. The problem is to schedule the activities so as to minimize the project completion time.

Observations

:: Note that this problem can be formulated as an LP problem (cf. Chapter 2, Example C2).

:: The LP formulation (in general) can be adapted to variants such as the following.

Suppose we introduce additional relationships between activities, specifically, an *incompatibility relationship*, defined as a situation where two activities must be performed during disjoint time intervals (for example, consider the "project" consisting of the activities a person performs from the moment they wake up until they leave the house: there are activities with precedence relationships, such as having breakfast and brushing one's teeth, but there are also incompatible activities, such as having breakfast and getting dressed). To model the presence of mutually incompatible activities, one can use the technique seen in Chapter 2, Example D3 (disjunctive constraints), which allows for the establishment of a precedence relationship between two incompatible activities. □

To introduce an "ad hoc" solution method for the project scheduling problem – in addition to the Simplex method mentioned above – we introduce a specific construction.

We construct a directed graph, G, such that:

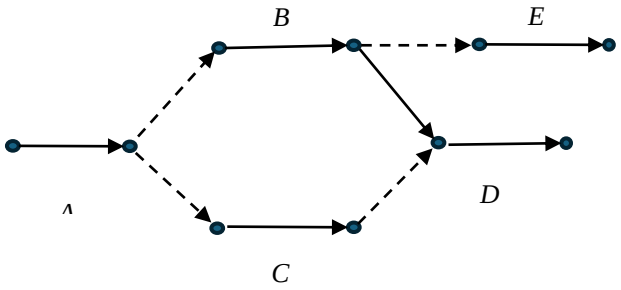
for every activity A_h with duration d_h, there exists an arc (x, y) with duration $d_{xy} = d_h$;

for every precedence relationship $A_i < A_j$, there exists a dummy arc (x, y) with duration $d_{xy} = 0$.

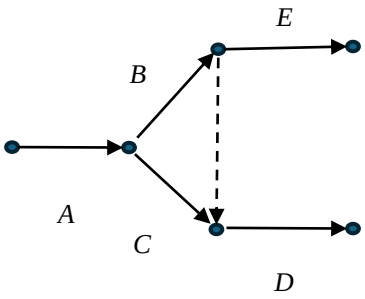
Esempio

Activities: A, B, C, D, E

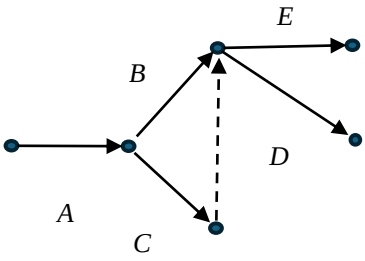
Precedence relationships: $A < B, A < C, B < D, B < E, C < D$



It is possible to simplify by "contracting" certain dummy arcs, but without adding or removing precedence constraints.



YES



NO ($C < E$ is added)

Property 1: The graph G is acyclic: indeed, there are no activities, say, $A_1, \ldots, A_k$, such that $A_1 < \ldots < A_k < A_1$, as this would constitute a logical inconsistency. □

Let V be the set of vertices of G, and let:

$V^- = \{v \in V$: no arc "enters" $v\}$

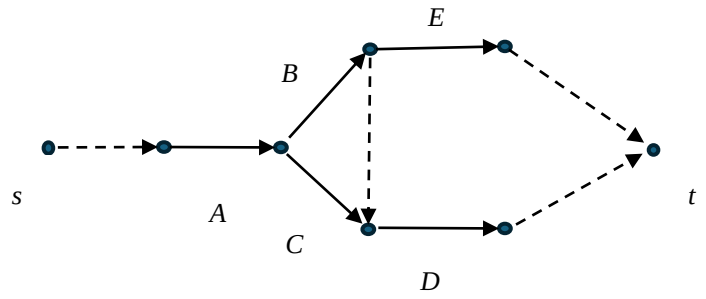
$V^+ = \{v \in V$: no arc "leaves" $v\}$

Property 2: $V^- \neq \varnothing$ and $V^+ \neq \varnothing$: this property is a consequence of Property 1. □

At this point, we extend the graph G as follows:

:: add a vertex s and add the edges (s, v) for every $v \in V^-$, with duration $d_{sv} = 0$;

:: add a vertex t and add the edges (v, t) for every $v \in V^+$, with duration $d_{vt} = 0$.



To summarize, *G* is an acyclic graph with a vertex *s* into which no edge enters and a vertex *t* from which no edge leaves; furthermore, a "duration" is associated with each edge of *G*. Specifically, the duration of a path is the sum of the durations of the edges that compose it. From the construction above, one can infer that the minimum project duration cannot exceed the duration of the longest path from *s* to *t* in *G*, since such a path serves as a lower bound within the project context; more precisely, the following result holds, which forms the basis of the "ad hoc" method we are introducing.

***Theorem* 5.2.1:** The length of the maximum-duration path from *s* to *t* in *G* is equal to the minimum project completion time. □

At this point, in technical terms, the problem consists of determining a path of maximum duration (which, in general, could represent cost, distance, etc.) between two vertices in a directed graph. However, the PERT method we are about to examine not only solves this problem but also provides additional insights that can be useful to the decision-maker; let us first introduce some preliminaries.

Topological ordering of a directed acyclic graph *G*.

1. Assign the number 1 to any vertex $v \in V^-$.
2. Remove *v*, along with the edges incident to *v*, from the graph.
3. Repeat the process to assign the numbers 2, 3, …, *n*, in sequence, to the remaining vertices. □

Events, $T_{min}[h]$, $T_{max}[h]$

While each arc of G represents an activity (real or dummy), each vertex of *G* can be viewed as an *event*; specifically, for each vertex *h* of *G*, that is, for each event *h,* let:

$T_{min}[h]$ = the earliest start time at which event h can occur;

$T_{max}[h]$ = the latest start time by which event *h* must occur (so as not to delay the project's minimum completion time).

PERT method

Input: graph $G = (V, E)$ with $|V| = n$; $s, t \in V$; duration $d_{ij} \geq 0$ for each $(i, j) \in E$.

Output: a path of maximum duration from *s* to *t*, i.e., the minimum project completion time equal to $T_{min}[n]$; other information useful to the decision-maker.

Step 1) Perform a topological sort in *G*.

Step 2) $T_{min}[1] := 0$.

Step 3) for $h = 2, \ldots, n$, set $T_{min}[h] := \max \{ T_{min}[i] + d_{ih} : (i, h) \in E\}$

Step 4) $T_{max}[n] := T_{min}[n]$.

Step 5) for $h = n–1, \ldots, 1$, set $T_{max}[h] := \min \{ T_{max}[j] + d_{hj} : (h, j) \in E\}$.

Some observations on the PERT method:

the method can be viewed as consisting of two phases: a “forward phase” from *s* to *t*, during which the $T_{min}[h]$ values (for $h = 2, \ldots, n$,) are determined, with the value $T_{min}[n]$ indicating the project's minimum duration; a “backward phase” from *s* to *t*, during which the $T_{max}[h]$ values ($h = n–1, \ldots, 1$) are determined, thereby providing additional information to the decision-maker, recalling the meaning of the $T_{max}[h]$ value introduced earlier, as detailed below.

An activity $(i, j) \in E$ is *critical* when $T_{min}[i] = T_{min}[j]$, $T_{max}[i] = T_{max}[j]$, $T_{min}[i] + d_{ij} = T_{max}[j]$; a path from *s* to *t* is *critical* if it consists of critical activities; the ***property*** holds that *every project has at least one critical path* [in details, a critical path is a just a maximum-duration path, thus a the duration of a critical path equals the project's minimum completion time]. Specifically, to ensure the project's minimum completion time, critical activities are those that must proceed seamlessly (i.e., start as soon as possible and finish without delay).

Finally, we note that once the PERT method has been executed, as illustrated in [Fischetti], the decision-maker can visualize all activities on a timeline using the so-called *Gantt chart*; this chart highlights not only a critical path but also the potential to “parallelize” the various activities.

Solved Exercise

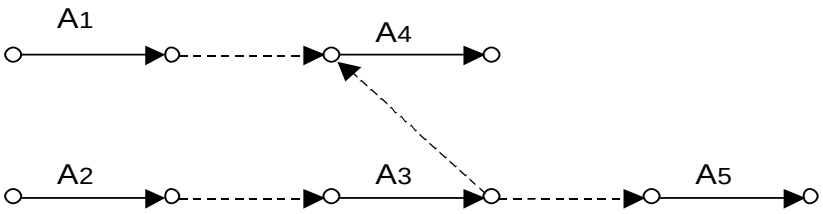
Using the PERT method, calculate the minimum completion time, the critical activities, and the maximum allowable slack (for each activity) for the following project:

there are 5 activities: A_1, …, A_5; each activity A_i has a duration d_i, where d = [5, 7, 2, 4, 3];

the precedence relationships are: $A_1 < A_4$; $A_2 < A_3$; $A_3 < A_4$; $A_3 < A_5$.

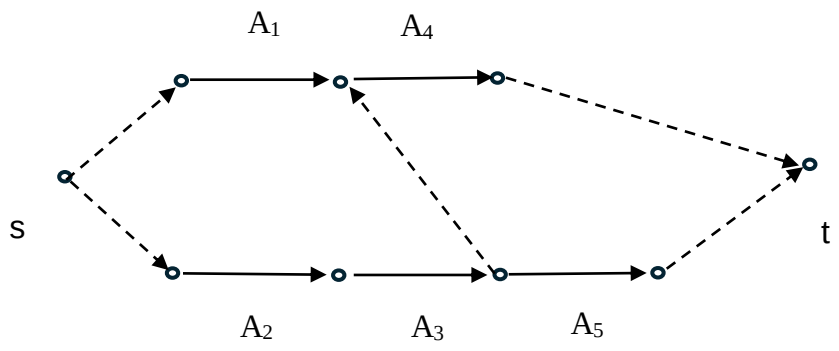
Solution

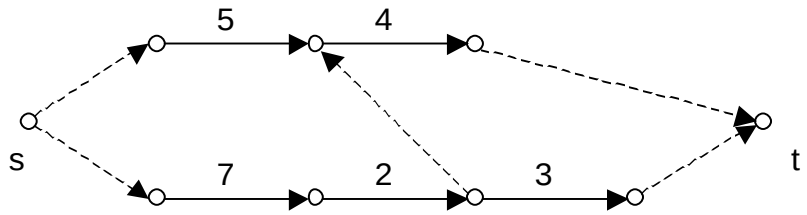
• Construct a directed graph *G* by associating an arc with each activity and introducing precedence relationships using dummy arcs.



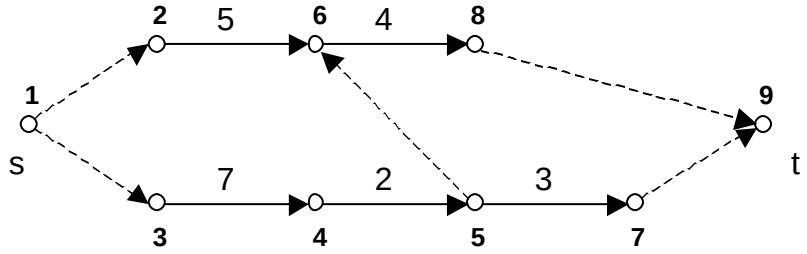
• Construct a graph *G'* by modifying graph *G* as follows: (i) remove any obsolete dummy edges – this operation must neither remove nor add precedence constraints; (ii) add a vertex *s* and dummy edges *sv* for every vertex *v* in *G* that has no incoming edges; (iii) add a vertex *t* and dummy edges *vt* for every vertex *v* in *G* that has no outgoing edges; (iv) assign the value to each activity *Ai* the value il valore d_i (for i = 1, . . ., 5); (v) assign the value 0 to each dummy edge (not shown in the figure).

Note: the activity values d_i and the values 0 of the dummy arcs can be viewed as "durations" d_{ij} associated with the arcs of *G'*; thus, in accordance with Theorem 5.2.1, the duration of a longest path from *s* to *t* in *G'* is equal to the minimum project completion time.

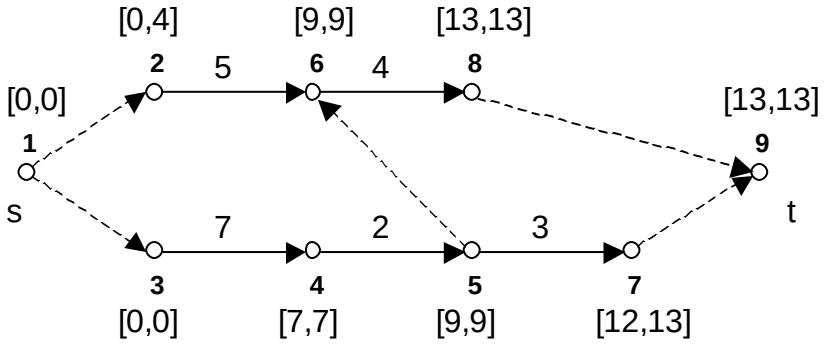




• Topologically order the nodes of *G'* [i.e., assign a progressive numbering to the nodes such that if the number associated with node *x* is less than the number associated with node *y*, then there is no path in *G'* from *y* to *x*].

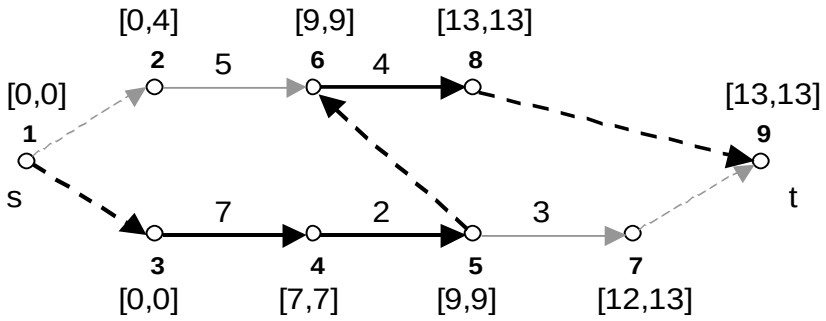


• Assign a double label [T_{min} , T_{max}] to each vertex; that is, for each vertex *h* of *G'*, calculate two values, $T_{min}(h)$ and $T_{max}(h)$, as follows: (i) $T_{min}(1) = 0$; (ii) for $h = 2, \ldots, 9$, set $T_{min}(h) = \max\{T_{min}(i) + q_{ih} : (i, h) \in E\}$; (iii) $T_{max}(9) = T_{min}(9)$; (iv) for $h = 8, \ldots, 1$, set $T_{max}(h) = \min\{T_{max}(i) + q_{hi} : (h, i) \in E\}$.



The minimum project completion time is given by $T_{min}[9] = 13$.

The *critical activities* (i.e., the arcs (i, j) such that: $T_{min}[i] = T_{max}[i]$; $T_{min}[j] = T_{max}(j)$; $T_{max}[j] - T_{max}[i] = q_{ij}$) are A_2, A_3, A_4. These are the activities for which any delay in execution directly causes a delay in the minimum project completion time. Together with dummy arcs corresponding to dummy critical activities, they define at least one *critical path*, as shown in the figure below.



5.3 Maximum Flow problem

Problem definition:

Let $G = (V, A)$ be a directed graph, and let s and t be two vertices of G. A capacity $k_{ij} \geq 0$ is associated with each arc $(i, j) \in A$. A *flow* in G (from s to t) is an assignment of values $x_{ij} \geq 0$ for each arc $(i, j) \in A$ such that:

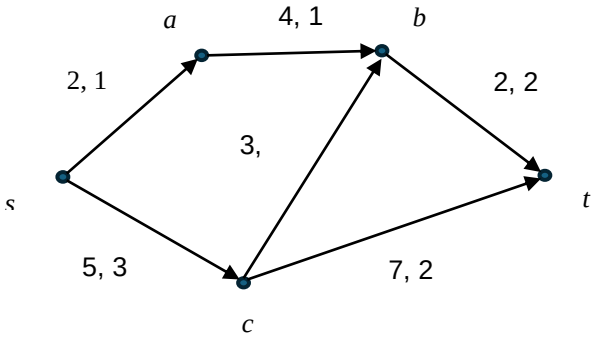
:: $\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = 0$ for every vertex $i \neq s, t$ of G (i.e., for every vertex $i \neq s, t$ of G, the sum of the incoming flow equals the sum of the outgoing flow);

:: $x_{ij} \leq k_{ij}$ for every arc $(i, j) \in A$.

The value of a flow from s to t is $\sum_{(s,j)\in A} x_{sj}$.

The problem is to determine a flow from s to t with maximum value. □

Notation: A directed graph G defined as above, with two designated vertices (s and t) and a capacity (the values b_{ij}) associated with each arc, is called a *network*; thus, the problem defined above will henceforth also be referred to as the problem of finding a maximum flow in a network G. □



Two examples to illustrate the problem:

:: network G can be visualized as a water distribution network: each edge is a pipe with a specific capacity, and each node is a junction connecting multiple pipes; a flow in G represents the volume of water moving through each pipe such that, at every junction, the total inflow equals the total outflow; thus, a maximum flow represents the greatest amount of water that can be injected at s and transported – under steady-state conditions – to t (as if the entire network were a single pipe);

:: network G can be visualized as an airline network; specifically, suppose a direct flight from airport s to airport t has been cancelled, and the airline is looking for a way to transport as many passengers as possible from s to t using the remaining network G (within a single day); every vertex in the network other than s and t represents a potential transit airport, and every edge (i, j) represents a flight between airport i and airport j, with the capacity b_{ij} representing the number of available seats on that flight; thus, a maximum flow represents the maximum number of passengers the airline can successfully transport to t.

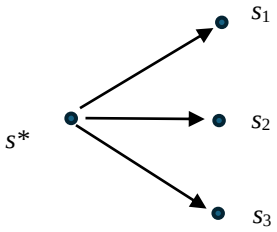
Observations

:: Note that this problem can be formulated as an ILP problem (see Chapter 2, Example C3).

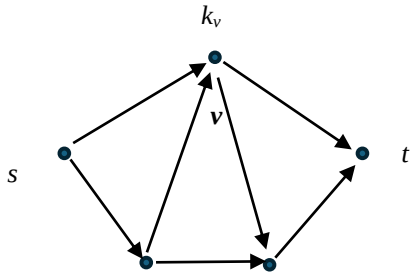
:: Note that although this problem is initially formulated as an ILP problem – observing that the requirement for integer variables implies the indivisibility of the units of the good being transported, and thus the constants b_{ij} can also be assumed to be integers – it can be formulated (and thus treated) as an LP problem; indeed, as seen with TUM matrices, the integrality constraints can be dropped [from the ILP formulation] and the continuous relaxation [of that formulation] solved; specifically, the Maximum Flow problem can be solved using the Simplex method.

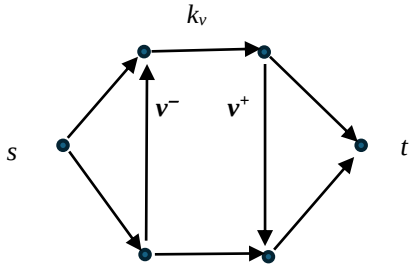
:: The ILP formulation can (in general) be adapted to variants such as the following:

:: :: if there are multiple sources, the problem can be reduced to the single-source case as follows:



:: :: if there is a node v assigned a "capacity" k_v, then the problem can be reduced to the case where "capacities" are assigned only to the edges:





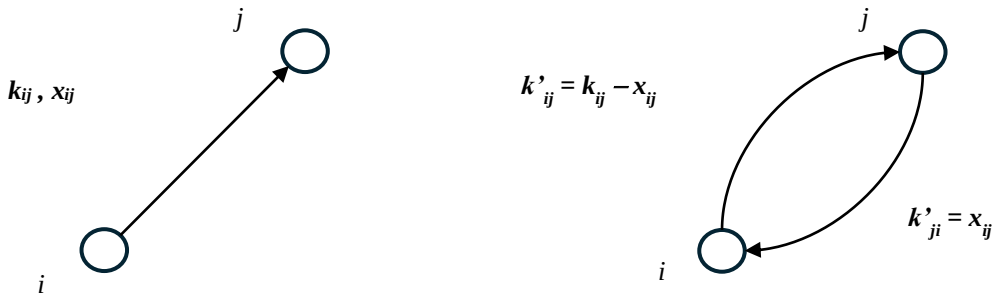
;

***Definition*:** Let **x** be a flow in a network $G = (V, A)$. The *incremental network* $G' = (V, A')$ *with respect to flow* **x** is obtained from the original network $G = (V, A)$ by replacing each arc $(i, j) \in A$ with two arcs:

• a forward arc (i, j) with residual capacity $k'_{ij} = k_{ij} - x_{ij}$

• a reverse arc (j, i) with residual capacity $k'_{ji} = x_{ij}$

and subsequently removing arcs with zero capacity. □



***Theorem* 5.3.1:** A flow **x** in a network $G = (V, A)$ is optimal if and only if there is no path from s to t in the incremental network $G' = (V, A')$ with respect to flow **x**. □

SIDE NOTE: max flow / min cut

***Definition*:** Given a network $G = (V, A)$ [with $s, t \in V$], we define:

:: a *section* (or *cut*) of G as a partition, say $\{S, T\}$, of V such that $s \in S$ e $t \in T$;

:: the *capacity of the section* {S, T} as the quantity $K(S, T) = \sum_{\substack{i \in S \\ j \in T}} k_{ij}$. □

***Theorem* 5.3.2:** Given a network $G = (V, A)$ [with $s, t \in V$], the value of a maximum flow in G is equal to the capacity of a minimum cut in G. □

On one hand, one can verify (even if only intuitively) that the value of a maximum flow in G is less than or equal to the capacity of any cut in G. Now Theorem 5.3.2 states that the value of a maximum flow in G is equal to the capacity of a minimum cut in G. In other words, the theorem allows one to visualize the network as if it were a single hypothetical arc (that is, a single pipe, in the illustration above), thereby calculating the hypothetical capacity of this single hypothetical arc, which is naturally equal to the value of a maximum flow within that single hypothetical arc. [End of SIDE NOTE]

Ford-Fulkerson algorithm

Input: a network G, i.e., a directed graph $G = (V, E)$; $s, t \in V$; $k_{ij} \geq 0$ for every $(i, j) \in E$.

Output: a maximum-value flow in network G.

Step 1) flow **x := 0**, i.e., $x_{ij} := 0$ for every $(i, j) \in E$;

Step 2) construct the incremental network with respect to flow **x**, let $G' = (V, A')$;

Step 3) if there is no path from s to t in $G' = (V, A')$, then STOP [flow **x** is optimal];

otherwise:

--- "saturate" a path from s to t in $G' = (V, A')$;

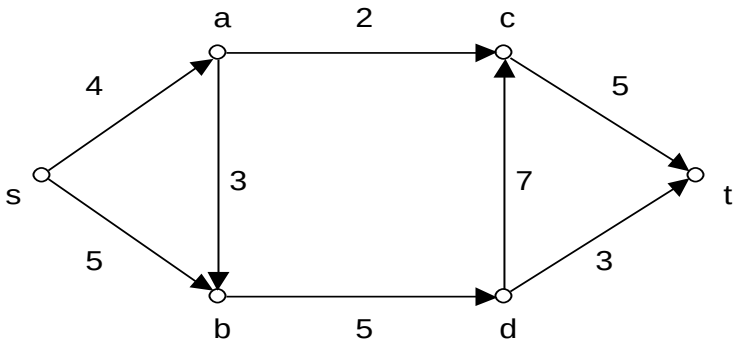
--- update flow **x** in G;

--- go to Step 2. □

To illustrate the meaning of "saturating" an s-t path in $G' = (V, A')$, let us first imagine that $G' = (V, A')$ is exactly an s-t path; in this case, the meaning is to increase the current flow along that path as much as possible. Specifically, this maximum increase will equal the residual capacity of the arc on that path with the minimum residual capacity, that is, the path's *bottleneck*. If $G' = (V, A')$ is not exactly a path, one can still designate an s-t path and then proceed as described above.

Solved Exercise

Using the Ford-Fulkerson algorithm, calculate a maximum flow from s to t in the following network, where the corresponding capacity is indicated next to each edge.

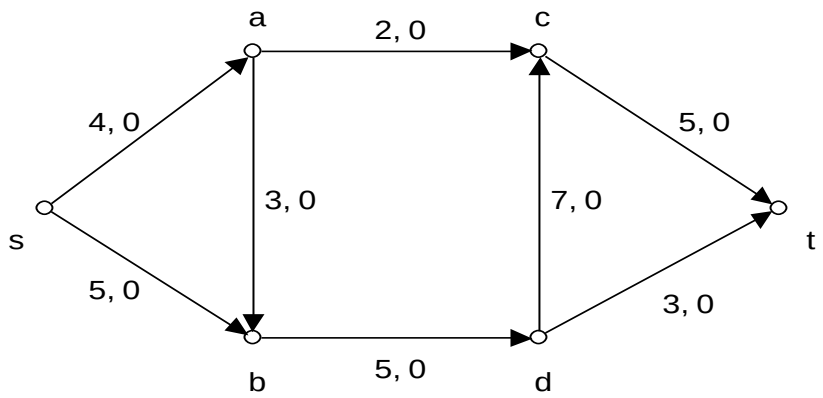


Solution

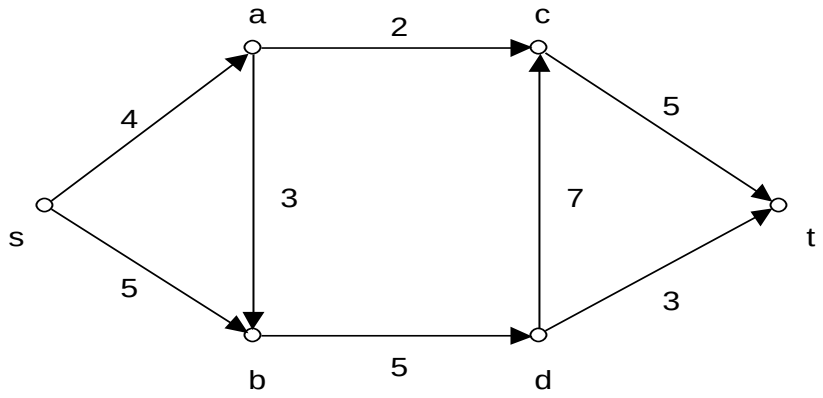
We apply the Ford-Fulkerson algorithm on the following pages; specifically, it is understood that each "iteration" begins with the update of the current flow, which is performed at the end of Step 3.

Initialization

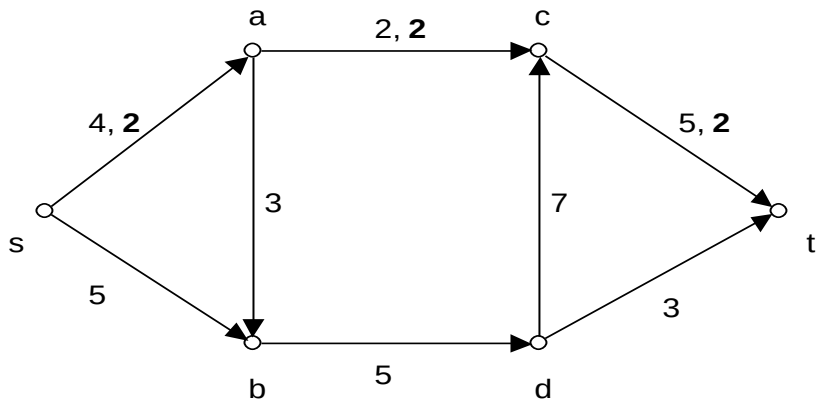
• set each flow component on every arc to 0



• construct an incremental network relative to the flow

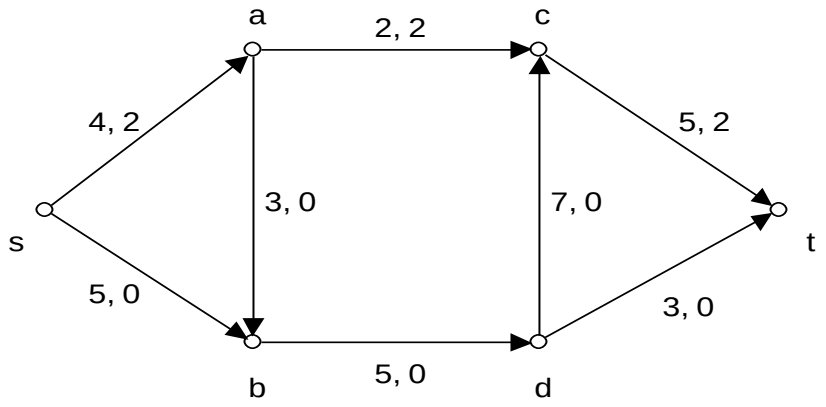


• identify a path from *s* to *t* in the incremental network and saturate it; the path is *s*, *a*, *c*, *t*; saturate it by "sending" as much flow as possible, namely, 2 (arc (*a*, *c*) is the bottleneck)

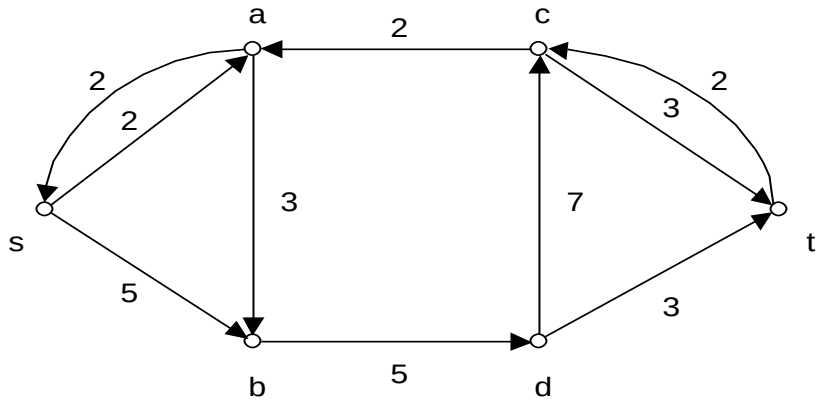


Iteration 1

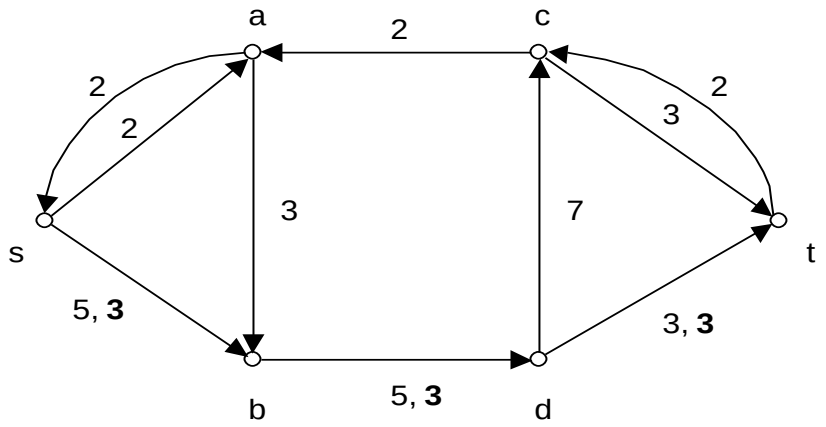
• update the flow (based on the augmenting path found)



• construct the incremental flow network

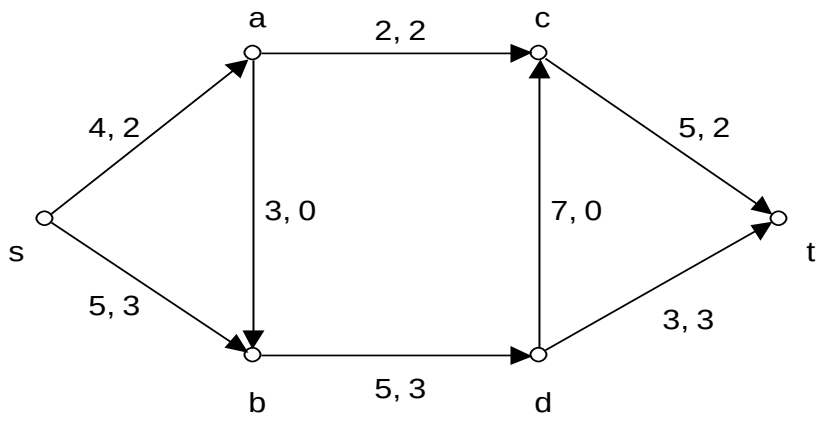


• identify a path from *s* to *t* in the incremental network and saturate it; the path is *s*, *b*, *d*, *t*; saturate it by "sending" as much flow as possible, namely, 3 (the arc (*d*, *t*) is the bottleneck)

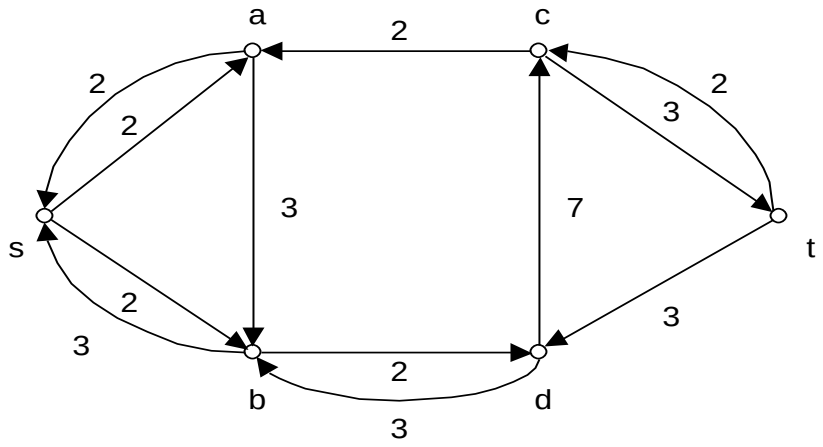


Iteration 2

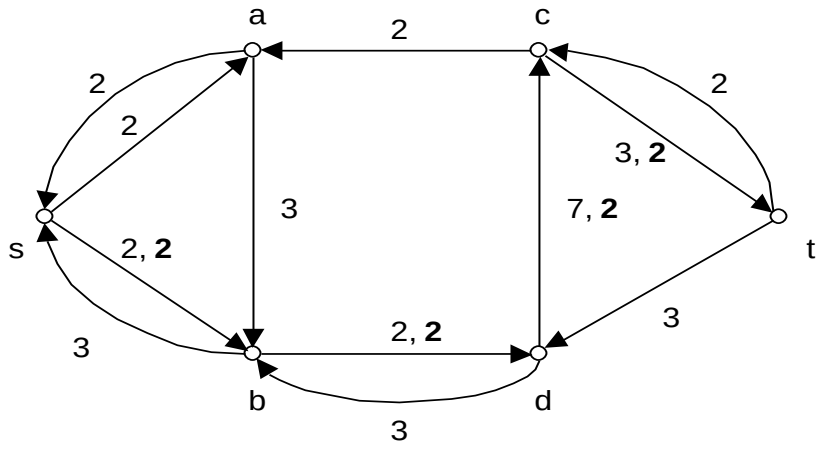
• update the flow (based on the augmenting path found)



• construct the incremental flow network

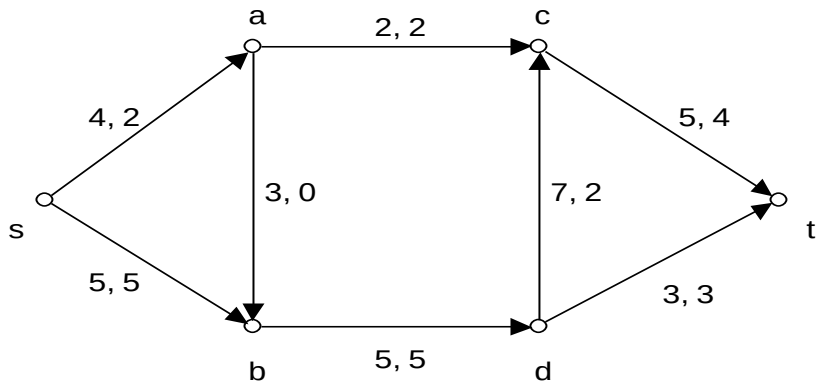


• identify a path from *s* to *t* in the incremental network and saturate it; the path is *s*, *b*, *d*, *c*, *t*; saturate it by "sending" as much flow as possible, namely, 2 (the arc (*s*, *b*) is the bottleneck)

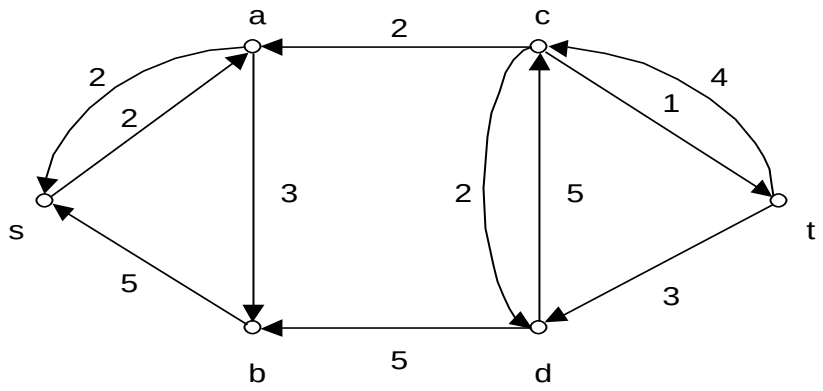


Iteration 3

- update the flow (based on the augmenting path found)

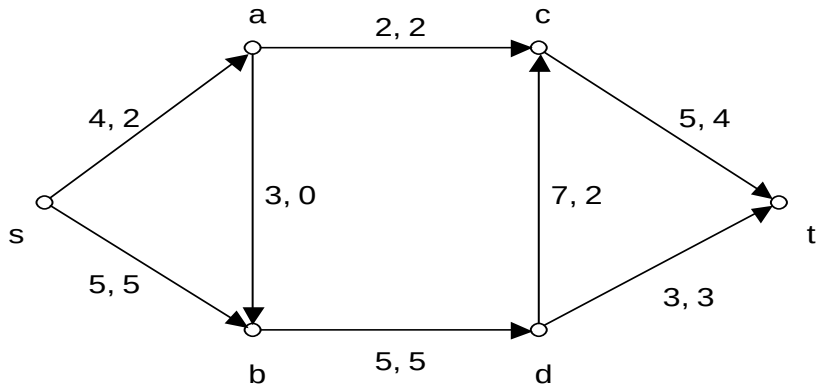


- construct the incremental flow network



- identify a path from *s* to *t* in the incremental network and saturate it;

in this case, there is no path from *s* to *t* in the residual network; therefore, the current flow (i.e., the one resulting from the last update – shown in the figure below) is optimal, and its value is the sum of the flow components on the edges leaving *s*: 2 + 5 = 7.



Concluding remarks on applications

Theorem 5.3.2 stands out as the most general of the so-called "Mengerian Theorems", referring to the theorem formulated by the scholar Karl Menger. Over the last century, various scholars introduced a range of theorems involving min-max relationships across different topics. It eventually became evident that this group of theorems largely implied one another; specifically, Theorem 5.3.2 implied all of them, including Menger's Theorem.

Nevertheless, we present this theorem here (where # denotes "number") primarily due to its practical significance:

Menger's Theorem: Given a simple network G [i.e., a directed graph $G = (V, E)$; $s, t \in V$; arc-capacities are not specified], the following hold:

:: max # of arc-disjoint paths from s to t = min # of arcs whose removal disconnects s and t;

:: max # of node-disjoint paths from s to t = min # of nodes whose removal disconnects s and t. □

Note that Menger's Theorem has a natural application in network monitoring; for instance, it helps determine the minimum number of arcs or nodes that must be "monitored" (such as by establishing a checkpoint) to control all traffic flowing through network G from s to t. Another similar application involves calculating the minimum number of arcs or nodes that would need to be removed to disconnect a simple network (this can be achieved by considering all pairs of nodes in G, applying the Ford-Fulkerson algorithm to each pair, and then selecting the "best" solution found). In accordance with Theorem 5.3.2, all the values specified in Menger's Theorem can be determined by solving the maximum flow problem in G [in the special case where $k_{ij} = 1$ for every $(i,j) \in E$] using the Ford-Fulkerson algorithm; specifically, once a maximum flow has been determined, a minimum-capacity cut can easily be identified. □

5.4 Production Planning problem

Problem definition:

A company must manufacture a product to meet a sales plan over a specific time horizon (e.g., one month) divided into periods (e.g., weeks), which specifies the sales targets for each period. Production capacity may vary from period to period, as may the unit production cost and the unit cost of holding inventory at the end of each period in the company's warehouse. There is no initial inventory, nor is any inventory desired at the end of the time horizon.

In detail:

:: let $\{1, \ldots, T\}$ be the finite set of periods in the [known] time horizon;

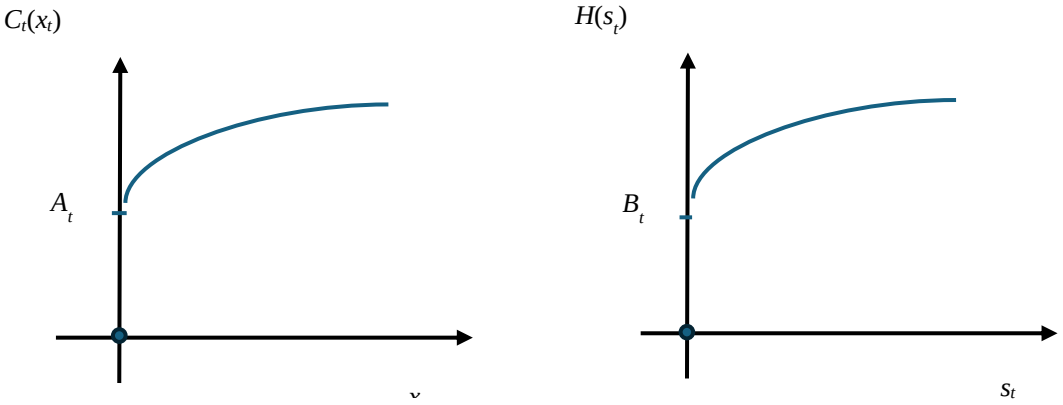
:: for each $t \in \{1, \ldots, T\}$, let d_t be the sales to be made in period t [known / estimated];

:: for each $t \in \{1, \ldots, T\}$, let x_t be the production level in period t [to be determined];

:: for each $t \in \{0, 1, \ldots, T\}$, let s_t be the inventory level at the end of period t [to be determined];

:: for each $t \in \{1, \ldots, T\}$, let $C_t(x_t) = A_t \eta(x_t) + c_t(x_t)$ be the *production cost function* in period t [known / estimated], where: A_t is a constant representing a fixed cost [i.e., $\eta(x_t) = 0$ if $x_t = 0$, while $\eta(x_t) = 1$ if $x_t > 0$], and $c_t(x_t)$ is a concave function of x_t;

:: for each $t \in \{1, \ldots, T\}$, let $H_t(x_t) = B_t \eta(s_t) + h_t(s_t)$ be the *storage cost function* at the end of period t [known / estimated], where: B_t is a constant representing a fixed cost [i.e., $\eta(s_t) = 0$ if $s_t = 0$, while $\eta(s_t) = 1$ if $s_t > 0$], and $h_t(s_t)$ is a concave function of s_t.



The problem is to determine the quantities x_t and s_t , for $t \in \{1, \ldots, T\}$, subject to the constraint of meeting the required sales for each period, so as to minimize $\sum_{t=1}^{T}[\, C_t(x_t) + H_t(s_t)]$.

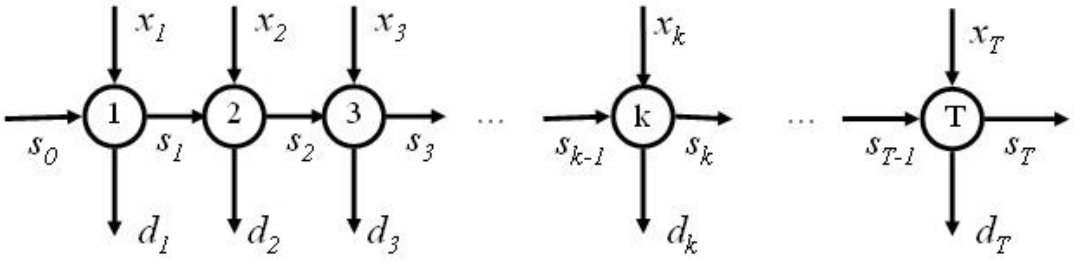
Note: as specified above, there is no initial inventory, nor is any desired at the end of the time horizon; thus, we can set $s_0 = 0$ and $s_T = 0$. □

Observations

:: This problem cannot be formulated as an LP (or ILP) problem, given that the objective function would be non-linear. However, we observe that: if $A_t = 0$ and if $c_t(x_t)$ is a *linear* function (which is a specific case of a convex function), then $C_t(x_t)$ is a *linear* function; if $B_t = 0$ and $h_t(s_t)$ is a *linear* function (which is a specific case of a convex function), then $H_t(x_t)$ is a *linear* function; in conclusion, if these assumptions hold, the problem can be formulated as an LP problem (cf. Chapter 2, Example C4).

:: In defining this problem, we consider a good to be "produced"; however, the problem can equally well consider a good to be "purchased."

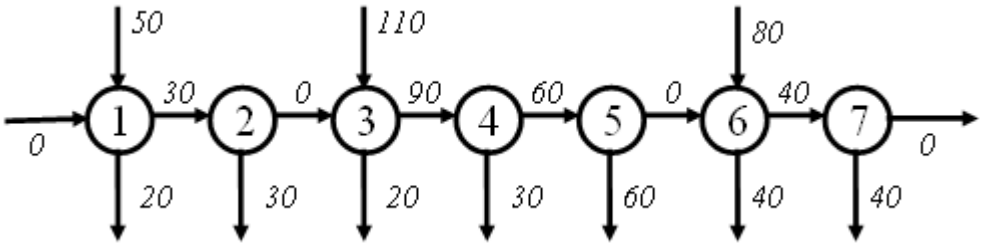
:: Finally, we observe that the only variables to be determined are the quantities x_t (for $t \in \{1, \ldots, T\}$), since the quantities s_t (for $t \in \{1, \ldots, T\}$) are uniquely determined once the quantities x_t (for $t \in \{1, \ldots, T\}$) are known; indeed, as illustrated below, the relationship $s_{k-1} + x_k = s_k + d_k$ holds for every $k \in \{1, \ldots, T\}$. □



The "ad hoc" solution method we introduce is based on the following two properties.

<u>*Property 1*</u>: There exists an optimal solution $\{x_1, \ldots, x_T\} \cup \{s_1, \ldots, s_{T-1}\}$ such that, for every $t \in \{1, \ldots, T\}$, exactly one of the two equations holds: $x_t = 0$ OR $s_{t-1} = 0$. In other words, there exists an optimal solution such that production levels are activated (i.e., > 0) if and only if the respective preceding inventory levels are zero (i.e., = 0). □

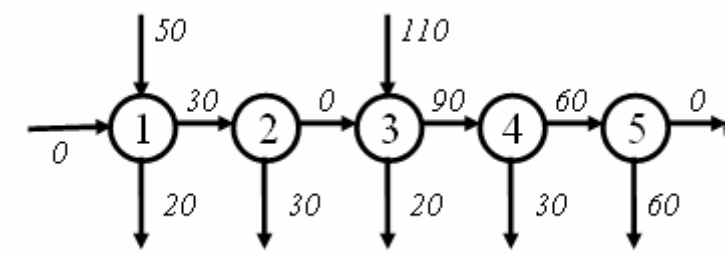
The following figure shows a hypothetical optimal solution that satisfies Property 1.

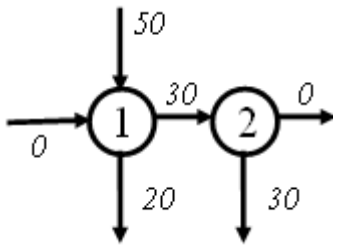


Property 2: The optimal solution of Property 1 is composed of optimal sub-solutions. □

For each $k \in \{1, \ldots, T\}$, let $\mathbf{P}_k$ be the version of our problem restricted to the time horizon $\{1, \ldots, T\}$, such that, in particular, $\mathbf{P}_T$ is the problem itself.

The significance of Property 2 is as follows. It can be observed that the optimal solution mentioned above partitions the set $\{1, \ldots, T\}$ into three *blocks*: $\{1, 2\} \cup \{3, 4, 5\} \cup \{6, 7\}$. Specifically, this optimal solution not only provides the optimal solution for problem $\mathbf{P}_7$ but also yields the optimal solutions for problems $\mathbf{P}_5$ and $\mathbf{P}_2$. In this sense, the optimal solution from Property 1 is composed of optimal sub-solutions.



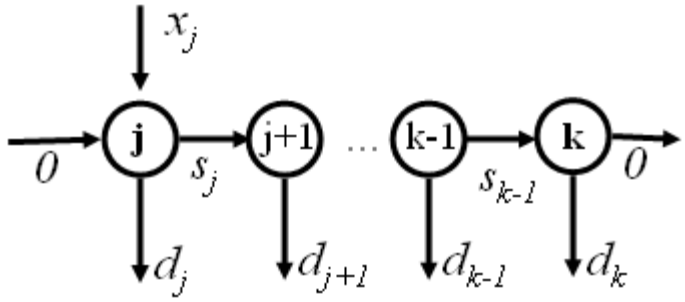


To formulate the "ad hoc" solution method, we introduce the following notation, which focuses on the costs associated with each possible block.

For each j , $k \in \{1, \ldots, T\}$, with $j \leq k$, let

$M(j, k) = C_j(x_j) + \sum_{t=j}^{k} H_t(s_t)$, where $x_j = \sum_{r=j}^{k} d_r$ and $s_t = \sum_{r=t+1}^{k} d_r$ (for $t \in \{j , \ldots, k–1\}$).

In other words, with reference to the block $\{j, \ldots, k\}$, $M(j, k)$ represents the cost required to produce only in period j (starting with zero inventory) and satisfy demand through period k (ending with zero inventory).



For each $k \in \{1, \ldots, T\}$, let F_k be the optimal solution value of problem $\mathbf{P}_k$.

Then, by Property 2, we have:

$$F_k = \min_{1 \le j \le k} \{F_{j-1} + M(j, k)\} \text{ , for every } k \in \{1, \ldots, T\}.$$

The preceding is a ***recursive expression*** for the value F_k in terms of the values F_j, where $1 \le j \le k-1$. Thus, thanks to this recursive expression, and by setting $F_0 = 0$, it is possible to calculate the values $F_1, F_2, \ldots, F_T$ sequentially. This is the core of the "ad hoc" method.

Wagner-Whitin method

The Wagner-Whitin method consists of three phases:

PHASE I: calculation of the values $M(j, k)$, for $j, k \in \{1, \ldots, T\}$, with j < k.

This calculation is based on the formula introduced above.

PHASE II: calculation of the values $F_1, F_2, \ldots, F_T$

This calculation is based on the recursive expression introduced above.

PHASE III: reconstruction of the optimal solution.

This reconstruction is based on a backward process involving the calculations performed in Phase II; specifically, starting from the calculation of F_T, one identifies the period j such that $F_T = F_{j-1} + M(j, T)$, thereby identifying the block $\{j, T\}$ (which is the final block) of the optimal solution. This procedure is then iterated, starting from period j and proceeding back to period 1; in this way, all blocks of the optimal solution are identified, and, implicitly, all values of the optimal solution $\{x_1, \ldots, x_T\} \cup \{s_1, \ldots, s_{T-1}\}$. □

Solved Exercise

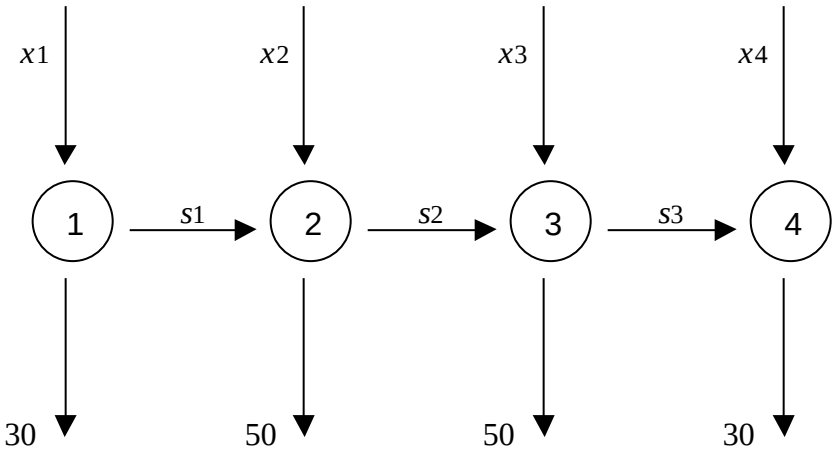
A good must be produced over a time horizon consisting of 4 periods. For each period $t = 1, \ldots, 4$, the demand dt to be met in that period is known. There is no initial inventory at the start of period 1, and no inventory is desired at the end of period 4.

Using the Wagner-Whitin method, determine the production quantity x_t and the inventory quantity s_t for each period $t = 1, \ldots, 4$, so as to minimize total costs and satisfy demand as it arises. The production cost function is $C_t = A_t w(x_t) + c_t x_t$, where $w(x_t) = 1$ if $x_t > 0$, and $w(x_t) = 0$ otherwise; the storage cost function is $H_t = h_t s_t$;; the relevant data are provided in the table.

Period	d_t	A_t	c_t	h_t
1	30	40	2	1
2	50	20	3	2
3	50	10	4	2
4	30	20	2	

Solution

The situation can be represented as shown in the figure below, where the quantities for demand, production (x_i), and storage (s_i = the amount remaining at the end of the period) are associated with each period i. Thus, given the quantities and the cost functions, the problem is to determine the quantities x_i for each period $i = 1, \ldots, 4$ and the quantities x_i for each period $i = 1, \ldots, 3$.



Step 1: calculate the values $M(i, j)$ for each pair of periods (i, j) with $i \leq j$.

The value $M(i, j)$ is the total cost incurred to satisfy the demands of periods $i, i+1, \ldots, j$, by producing only in period i (starting with zero inventory and ending with zero inventory).

$M(1,1) = 40 + 2 \cdot 30 = 100$

$M(1,2) = 40 + 2 \cdot (30+50) + 1 \cdot 50 = 250$

$M(1,3) = 40 + 2 \cdot (30+50+50) + 1 \cdot (50+50) + 2 \cdot 50 = 500$

$M(1,4) = 40 + 2 \cdot (30+50+50+30) + 1 \cdot (50+50+30) + 2 \cdot (50+30) + 2 \cdot 30 = 710$

$M(2,2) = 20 + 3 \cdot 50 = 170$

$M(2,3) = 20 + 3 \cdot (50+50) + 2 \cdot 50 = 420$

$M(2,4) = 20 + 3 \cdot (50+50+30) + 2 \cdot (50+30) + 2 \cdot 30 = 630$

$M(3,3) = 10 + 4 \cdot 50 = 210$

$M(3,4) = 10 + 4 \cdot (50+30) + 2 \cdot 30 = 390$

$M(4,4) = 20 + 2 \cdot 30 = 80$

Step 2: recursively calculate F_k for $k = 1, \ldots, 4$.
The value F_k is the value of an optimal solution to the problem restricted to the first k periods. Thus, F_4 is the value of the sought-after optimal solution. The calculation is performed recursively, setting $F_0 = 0$ and applying the formula $F_k = \min_{1 \le j \le k} \{F_{j-1} + M(j, k)\}$.

$F_0 = 0$

$F_1 = \min \{F_0 + M(1,1)\} = 100$

$F_2 = \min \{F_0 + M(1,2), F_1 + M(2,2)\} = \{250, 270\} = 250$

$F_3 = \min \{F_0 + M(1,3), F_1 + M(2,3), F_2 + M(3,3)\} = \{500, 520, 460\} = 460$

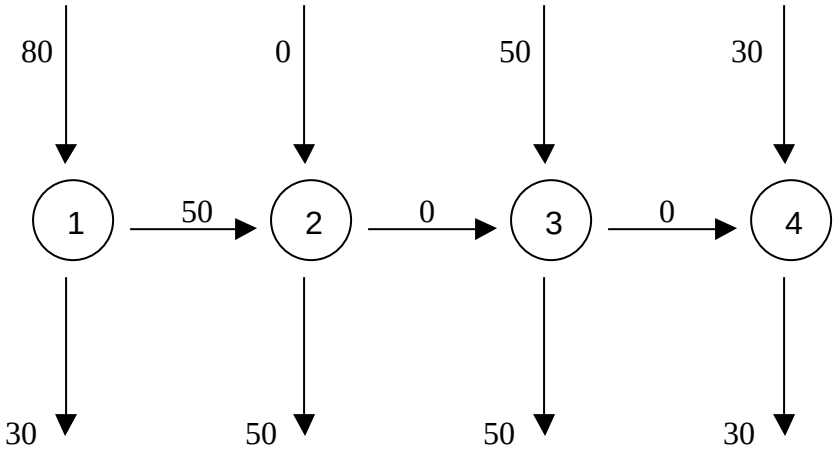
$F_4 = \min \{F_0 + M(1,4), F_1 + M(2,4), F_2 + M(3,4), F_3 + M(4,4)\} = \{710, 730, 640, 540\} = 540$

Step 3: Highlight the optimal solution found.

To do this, start from F_k and work backwards, iteratively substituting the F_k values (based on the calculations just performed) until reaching F_0, as follows:

$F_4 = F_3 + M(4,4) = F_2 + M(3,3) + M(4,4) = F_0 + M(1,2) + M(3,3) + M(4,4)$.

The optimal solution is thus identified as $M(1,2) + M(3,3) + M(4,4)$; that is, production takes place in period 1 (meeting demand for periods 1 and 2), in period 3 (meeting demand for period 3), and in period 4 (meeting demand for period 4). The optimal values for $x_1, \ldots, x_4$ and $s_1, \ldots, s_3$ can then be deduced from their definitions and the demand figures, as shown in the diagram.



Concluding remarks on applications

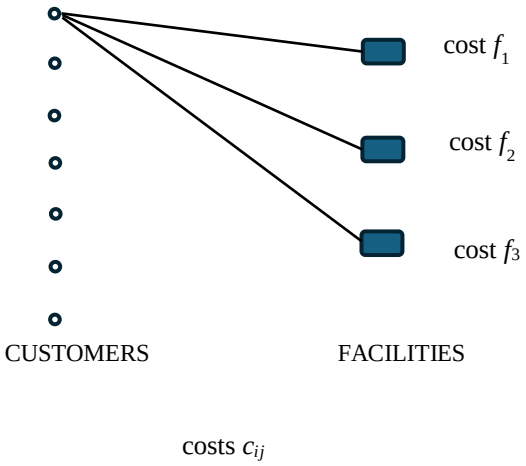
:: The Production Planning problem is a dynamic programming problem, as evidenced by Property 2, in light of our previous discussion regarding the shortest-path problem. Notably, the final stage of the Wagner-Whitin algorithm is dedicated to reconstructing the optimal solution, just as the final stage of Dijkstra's algorithm is dedicated to reconstructing the optimal solution.

:: The Production Planning problem, as presented above, is not merely an idealized abstraction far removed from practical application. For instance, the reference [Sassano] provides a simulation demonstrating that applying the Wagner-Whitin method would have resulted in significant cost savings. Furthermore, the production scheduling problem remains relevant even if production and/or storage costs are constant across periods, and even if initial and final inventory levels are set to a value $Q > 0$ rather than to 0. □

5.5 Facility Location problem

Problem definition:

A certain number of facilities must be activated, chosen from among m potential facilities, to supply n customers. Activating potential facility j (where $j \in \{1, \ldots, m\}$) incurs a cost $f_j \geq 0$. Each facility is assumed to have unlimited supply capacity. Each customer is connected to exactly one facility; specifically, connecting customer i to facility j incurs a cost c_{ij} (where $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$). The problem is to select which facilities to activate and to assign each customer to a facility, so as to minimize the total cost of facility activation and customer connection.



Let $N = \{1, \ldots, n\}$ be the set of customers and $M = \{1, \ldots, m\}$ be the set of facilities.

A solution to the problem [a candidate for the optimal solution] can be identified by a subset $T \subseteq M$, with which a connection of customers to elements of T is automatically associated, specifically, by connecting each customer $i \in N$ to a facility $j \in T$ for which c_{ij} is *minimum*. In detail, for every subset $T \subseteq M$ and every customer $i \in N$, let $c(T, i) = \min \{c_{ij} : j \in M\}$.

Then, the value of the solution identified by $T \subseteq M$ is $Z(T) = \sum_{j \in T} d_j + \sum_{i \in N} c(T, i)$.

Esempio. Di seguito vediamo un'istanza del problema e un esempio di calcolo di $Z(T)$ per due possibili sottinsiemi $T \subseteq M$:

matrix c_{ij}

		FACILITIES				
		1	2	3	4	5
	1	12	13	6	0	1
	2	8	4	9	1	2
	3	2	6	6	1	2
CUSTOMERS	4	3	5	2	10	8
	5	8	0	5	10	8
	6	2	0	3	4	1
array f		7	9	5	4	7

Assume that $T = \{4, 5\}$, that is, only facilities 4 and 5 are activated;
then $Z(T) = (4 + 7) + (0 + 1 + 0 + 8 + 8 + 1) = 29$.
Assume that $T = \{1, 3, 5\}$, that is, only facilities 1, 4, and 5 are activated;
then $Z(T) = (4 + 1 + 7) + (1 + 2 + 1 + 2 + 5 + 1) = 24$. □

Heuristic methods: greedy algorithm, local search algorithm

• Greedy algorithm

Below, we will apply the *greedy algorithm* to solve our problem and see that the greedy algorithm is a special case of a *local search algorithm*.

Regarding *hard* problems (see Appendix 1): heuristic methods are methods that determine a feasible solution to a problem in a relatively simple manner, generally without providing guarantees regarding the "quality" of the feasible solution found, that is, its distance from the optimal solution. However, heuristic methods are based on common-sense considerations that can offer partial guarantees regarding the "quality" of the feasible solution found, as we will see below in the case of the local search algorithm.

The greedy algorithm is defined (for our problem) as

Greedy algorithm

Initialization: set: $i := 1$; $T_0 := \varnothing$; $Z(T_0) := \infty$.

i-th iteration

$u_i = \text{argmin}\ \{Z(T_{i-1} \cup \{u\}) : u \in M \setminus T_{i-1}\}$;

if $Z(T_{i-1} \cup \{u_i\}) \geq Z(T_{i-1})$, then STOP [T_{i-1} is the *greedy solution*];

otherwise set $T_i := T_{i-1} \cup \{u_i\}$;

if $T_i = M$, then STOP [T_{i-1} is the *greedy solution*];

otherwise set $i := i + 1$ and proceed to i-th iteration. □

Solved Exercise

A certain number of facilities must be activated, chosen from among m potential facilities, to supply n customers. Activating potential facility j incurs a cost of f_j. Each facility is assumed to have unlimited supply capacity. Each customer will be connected to exactly one facility; specifically, connecting customer j to facility j incurs a cost of c_{ij}. The problem is to select which facilities to activate so as to minimize the total cost of activation and customer assignment.
Calculate a "greedy" solution for this problem based on the following instance:

matrix c_{ij}

		FACILITIES				
		1	2	3	4	5
	1	0	1	7	0	4
	2	1	0	3	0	3
	3	3	7	1	4	2
CUSTOMERS	4	1	7	4	0	7
	5	1	0	3	2	4
	6	4	4	1	4	3
	7	7	1	2	1	0

array f	7	9	5	4	7

Solution

Inizialization

$i = 1$; $T_0 = \varnothing$; $Z(T_0) = \infty$.

Iteration 1

$Z(\{1\})$	$Z(\{2\})$	$Z(\{3\})$	$Z(\{4\})$	$Z(\{5\})$
24	29	26	15	30

For example, $Z(\{1\}) = (0 + 1 + 3 + 1 + 1 + 4 + 7) + 7 = 24$.

The minimum is $Z(\{4\}) = 15$;
$15 < \infty \Rightarrow i = 2$, $T_1 = \{4\}$.

Iteration 2

$Z(\{4,1\})$	$Z(\{4,2\})$	$Z(\{4,3\})$	$Z(\{4,5\})$
20	22	14	18

For example, $Z(\{4,1\}) = (0 + 0 + 3 + 0 + 1 + 4 + 1) + (7 + 4) = 20$.

The minimum is $Z(\{4,3\}) = 14$;
$14 < 15 \Rightarrow i = 3$, $T_2 = \{4,3\}$.

Iteration 3

$Z(\{4,3,1\})$	$Z(\{4,3,2\})$	$Z(\{4,3,5\})$
22	21	20

For example, $Z(\{4,3,1\}) = (0 + 0 + 1 + 0 + 1 + 1 + 1) + (7 + 5 + 4) = 22$.

The minimum is $Z(\{4,3,5\}) = 20$;
$20 \geq 14 \Rightarrow$ STOP

The "greedy" solution is $T_2 = \{4,3\}$, con $Z(\{4,3\}) = 14$. □

• Local search algorithm

The greedy algorithm is part of the so-called "local search heuristics."

Definition* (Combinatorial Optimization problem***):** Given a finite set E, a family Φ of subsets of E, and a function $\omega : \Phi \to \mathbb{R}$, find **min** $\{\omega(F) : F \in \Phi\}$. □

Observation

A 0-1 ILP problem (i.e., one with binary decision variables, such as the Knapsack problem) is a combinatorial optimization problem (whereas a combinatorial optimization problem is not necessarily a 0-1 ILP problem, for instance, if the function ω mentioned above is non-linear). Thus, everything we are about to discuss regarding combinatorial optimization problems applies to 0-1 ILP problems as well. □

A general approach to finding a feasible solution for a combinatorial optimization problem is as follows: for each element $F \in \Phi$, a *neighborhood of F* [within Φ] is denoted by con $\mathcal{N}(F)$ and is defined as a set of elements in Φ that are, so to speak, "close" to F; this yields the collection $\{ \mathcal{N}(F) : F \in \Phi\}$, known as the *neighborhood system* for the elements of Φ.

A solution $F^* \in \Phi$ [to the problem] is called a *local optimum* if $\omega(F^*) \leq \omega(F)$ for all $F \in \mathcal{N}(F^*)$.

An optimal solution [to the problem] is called a *global optimum*.

Thus, given a neighborhood system, the objective may be set as finding a local optimum; in other words, if finding an optimal solution to the problem proves particularly difficult [including from the decision-maker's perspective], one might "settle" for finding a local optimum. A *local search algorithm* is, therefore, an algorithm that finds a local optimum.

Local search algorithm

Initialization: set $i := 1$; choose $F_0 \in \Phi$.

i-th iteration

$F_i = \text{argmin } \{\omega(F) : F \in \mathcal{N}(F_{i-1})\}$;

if $\omega(F_i) \geq \omega(F_{i-1})$, then STOP [$F_{i-1}$ is a local optimum];

otherwise, set $i := i + 1$ and proceed to the i-th iteration. □

The "quality" of a local search algorithm depends on the choice of $F_0 \in \Phi$ and the definition of the neighborhood system. Achieving good results generally requires experience. In any case, the decision-maker typically faces a trade-off between the quality of the local optimum and computational complexity.

Observation

We now verify that the greedy algorithm is indeed a local search algorithm; to do so, it suffices to check that it is based on the following neighborhood system, defined as follows for each $T \subseteq M$:

$\mathcal{N}(T) = \{T \cup \{u\} : u \in M \setminus T\}$. □

Appendix 1: Computational complexity and decision support

Note 1: As we have already seen – albeit in a simplified manner sufficient for our purposes, while referring the reader to Computer Science texts for a formal treatment – we briefly outline some concepts regarding computational complexity below. Mathematical Programming problems are classified (as far as possible) into *easy* and *hard* categories based on the computational complexity of the best known solution algorithm (defined in advance) for solving a generic instance of the problem: *Easy* problems, also known as "polynomial" problems, include Linear Programming (LP) problems; for instance, the project scheduling problem is easy. Furthermore, certain problems not initially modeled as LP can ultimately be reduced to LP (and are thus *easy*) through results concerning Totally Unimodular (TUM) matrices; examples include the Assignment, Transportation, Minimum Cost Path, and Maximum Flow problems. Regarding optimization problems that can be formulated as LP problems (which can, of course, be solved to optimality using standard LP methods, such as the Simplex Algorithm), "exact methods" are often introduced or studied to find an optimal solution without relying on standard LP techniques; this is because having fast methods available is beneficial, particularly for large problem instances.
Hard problems, also known as "NP-hard" problems, include Integer Linear Programming (ILP) problems; for instance, the facility location problem is hard. Regarding optimization problems that can be formulated as ILP problems (which can, of course, be solved to optimality using standard ILP methods, such as the Branch and Bound Algorithm), "heuristic methods" are often introduced or studied to find a solution that is presumably good, though not necessarily optimal, without relying on standard ILP techniques; this is because a standard method for ILP can be slow, even for small problem instances; consequently, the decision-maker (if they so choose) can employ a "heuristic method" which, while guaranteeing a presumably good, though not necessarily optimal solution, has the advantage of speed.

Note 2: As previously noted, the primary purpose of modeling and solving an optimization problem as a Mathematical Programming problem is to provide "decision support"; indeed, the resulting model is often merely an approximation of reality. In particular, the solution obtained can offer guidance both on how to proceed regarding the decision directly linked to the problem and on how to handle decisions indirectly related to it (for instance, we have seen that *shadow prices*, in the context of production involving the combination of resources, identify directions and conditions for the potential expansion of that production).

Appendix 2: Example of solving an LP problem using Excel

Let us look at how to solve a Linear Programming (LP) problem using Excel. We will revisit an exercise we have already seen.

A winery wishes to produce two types of wine: table wine and dessert wine. The profit the company derives from producing one unit of table wine is 3, while the profit from producing one unit of dessert wine is 7. Production requires a specific combination of two types of grapes: let us call them Type A and Type B. Producing 1 unit of table wine requires 3 units of Type A grapes and 2 units of Type B grapes. Producing 1 unit of dessert wine requires 1 unit of Type A grapes and 4 units of Type B grapes. Finally, the company has 1,000 units of Type A grapes and 400 units of Type B grapes available. The problem is to determine the quantities of table wine and dessert wine to produce in order to maximize total profit.

Decision variables: x_1, x_2

x_1 = quantity of table wine produced

x_2 = quantity of dessert wine produced

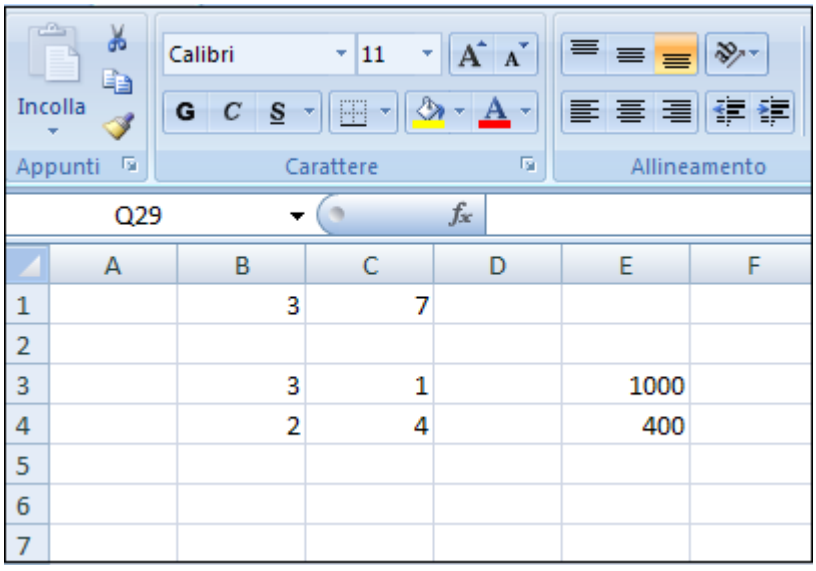
(1) max $3x_1 + 7x_2$

$3x_1 + x_2 \leq 1000$

$2x_1 + 4x_2 \leq 400$

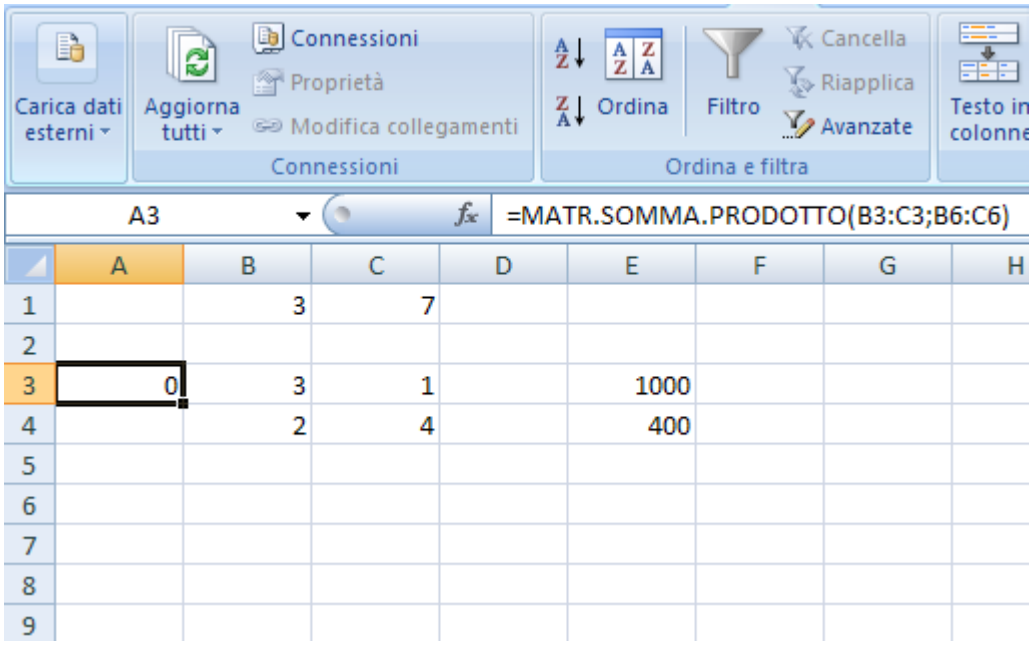
$x_1, x_2 \geq 0$

We enter the problem data into Excel as follows, also assigning cells B6 and C6 to the variables.



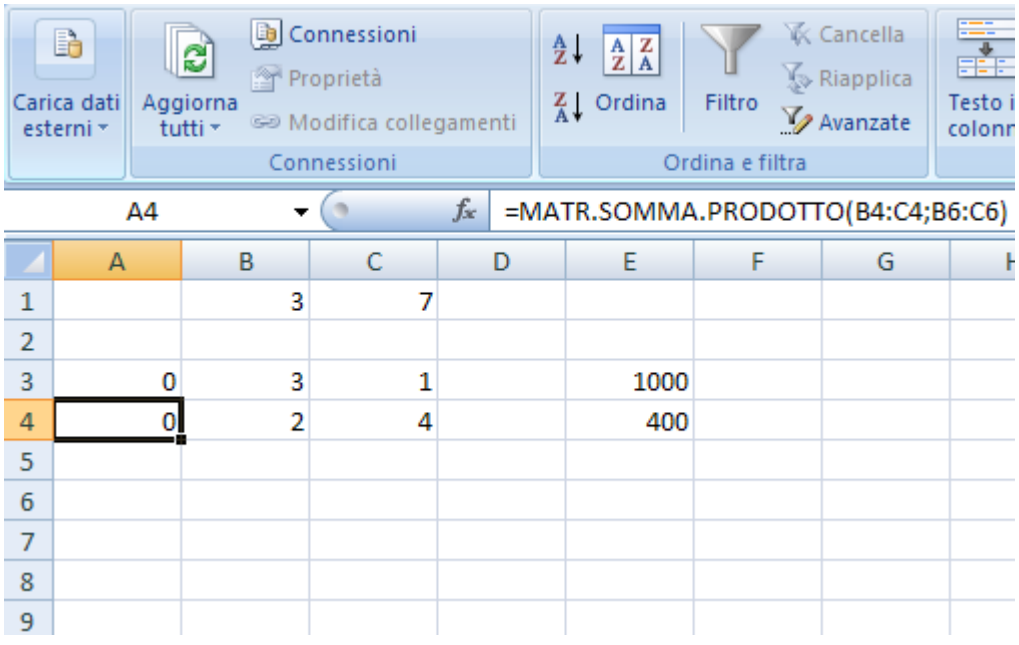
	A	B	C	D	E	F
1		3	7			
2						
3		3	1		1000	
4		2	4		400	
5						
6						
7						

In cell A3, we enter the SUMPRODUCT array (B3:C3; B6:C6), that is, the value $3x_1 + x_2$ corresponding to the first inequality.



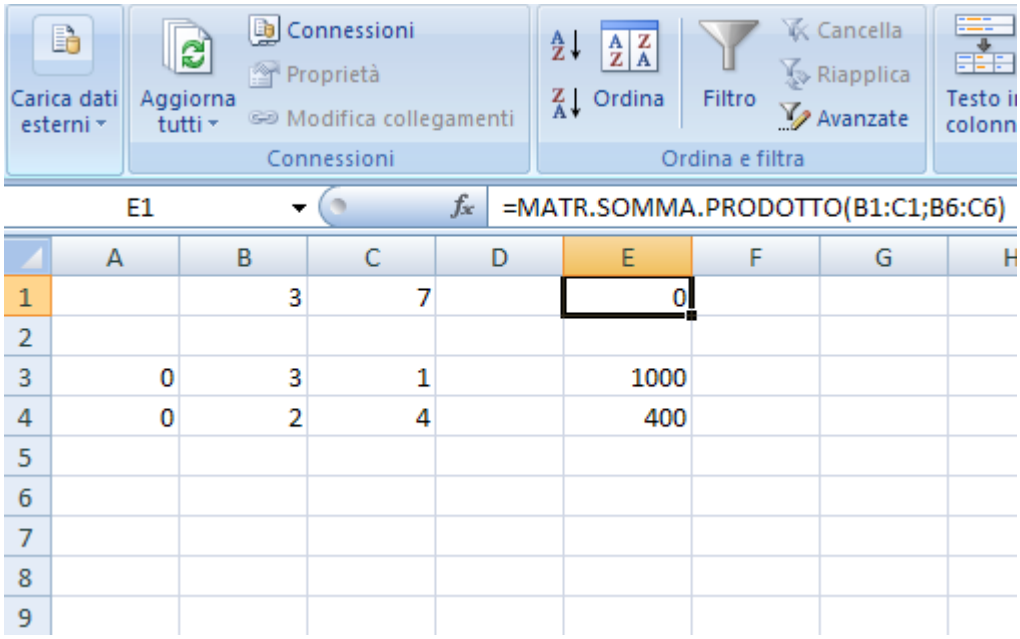
	A	B	C	D	E	F	G	H
1		3	7					
2								
3	0	3	1		1000			
4		2	4		400			
5								
6								
7								
8								
9								

In cell A4, we proceed in a similar manner, for the value $2x_1 + 4x_2$ corresponding to the second inequality.



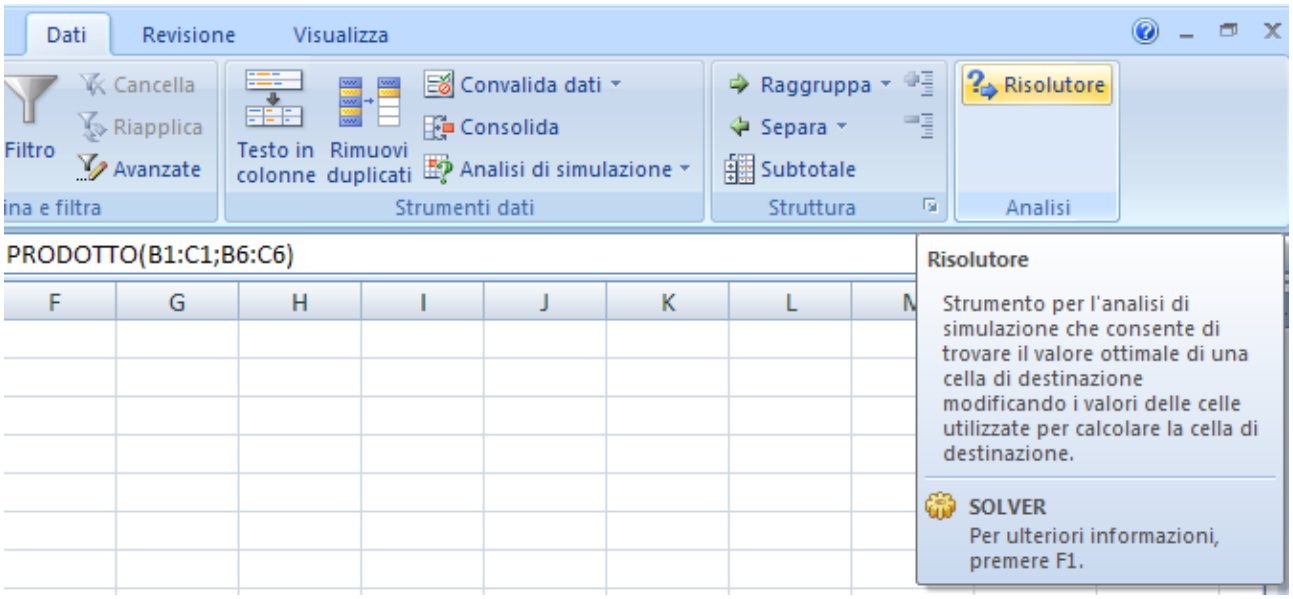
	A	B	C	D	E	F	G
1		3	7				
2							
3	0	3	1		1000		
4	0	2	4		400		
5							
6							
7							
8							
9							

In cell E1, we enter the SUMPRODUCT array (B1:C1; B6:C6), that is, the value $3x_1 + 7x_2$ corresponding to the objective function.

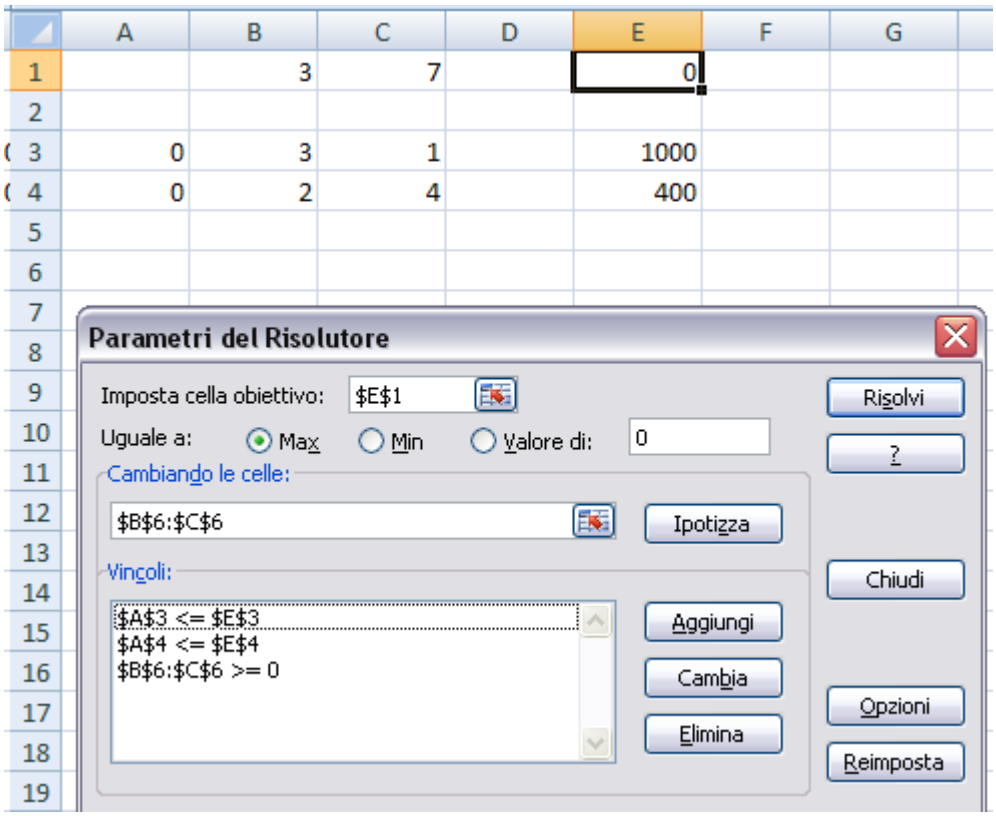


	A	B	C	D	E	F	G	H
1		3	7		0			
2								
3	0	3	1		1000			
4	0	2	4		400			
5								
6								
7								
8								
9								

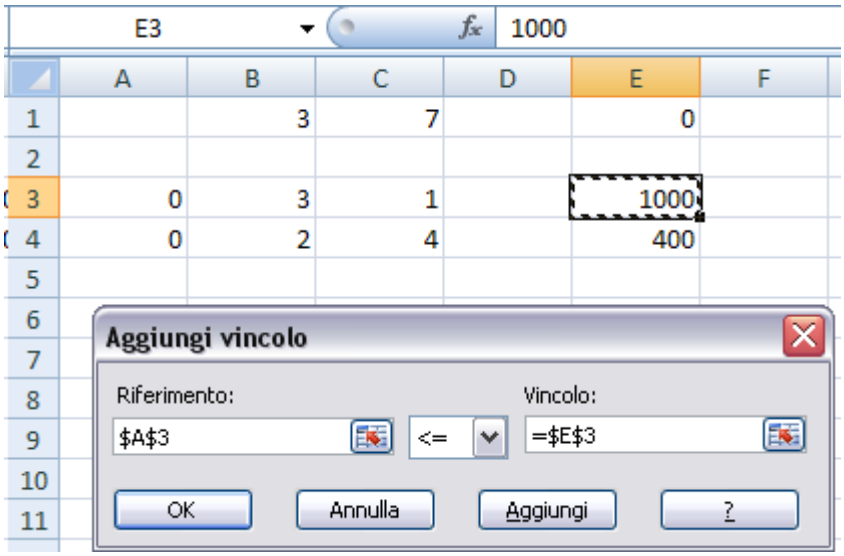
We can now use a tool provided by Excel: the Solver.



By clicking on it, we can enter the problem parameters:



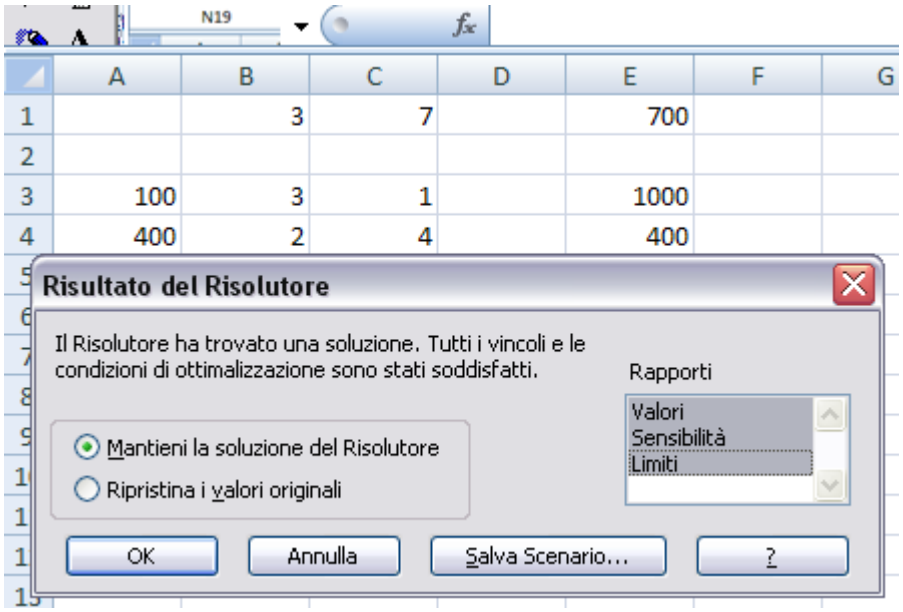
In the "Set Objective" field (i.e., the cell representing the objective function's value), enter cell E1; the optimal value of the objective function will appear in this cell. Then, select the "Max" option, since this is a maximization problem. In the "By Changing Variable Cells" field (i.e., the cells representing the variables), enter cells B6 and C6; the optimal values for the variables, specifically, the values for x_1 and x_2, will appear in these cells. Finally, enter all constraints in the "Subject to the Constraints" section using the "Add" button, as illustrated below for the first constraint.



Next, go to “Options” (within the Solver parameters), check the “Assume Linear Model” option, and click OK.

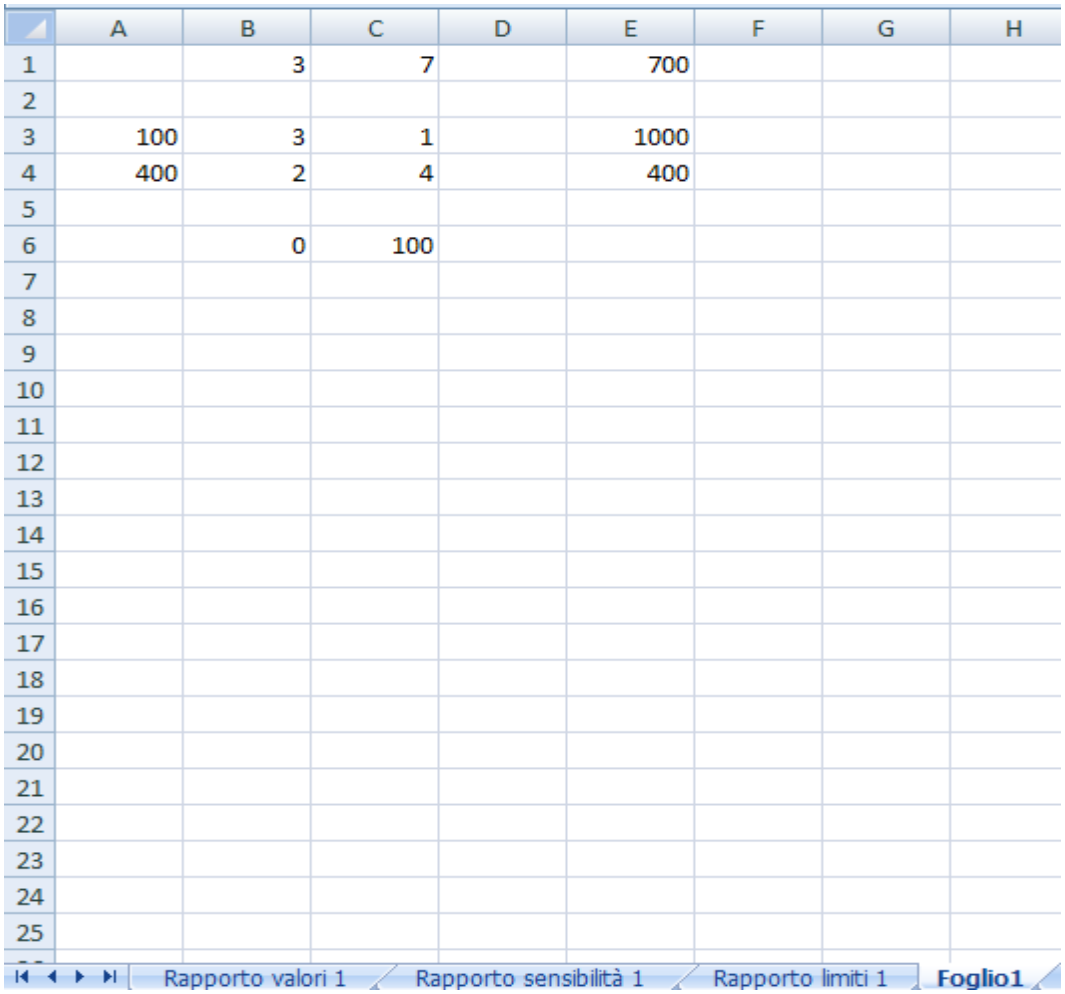
Now we can proceed to calculate the solution by clicking “Solve”.

At this point, we can also request reports on values, sensitivity, and limits (as shown below) by clicking on each of them.



We click OK and finally have the results we were waiting for.

The optimal solution is: $x^*_1 = 0$, $x^*_2 = 100$, with a value of 700.



	A	B	C	D	E	F	G	H
1		3	7		700			
2								
3	100	3	1		1000			
4	400	2	4		400			
5								
6		0	100					
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								
23								
24								
25								

At the bottom, we can click on the various reports to obtain more precise information about the results. Let us look in particular at the "Sensitivity Report": here, we can see the following.

1. The *shadow prices* of the resources, which are the optimal solution to the dual problem:

$$\begin{aligned} \min \quad & 1000\, y_1 + 400\, y_2 \\ & 3y_1 + 2y_2 \geq 3 \\ & y_1 + 4y_2 \geq 7 \\ & y_1, y_2 \geq 0 \end{aligned}$$

that is, $y^*_1 = 0$, $y^*_2 = 1.75$, with a value of 700.

Celle variabili

Cella	Nome	Valore finale	ridotto Costo	oggettivo Coefficiente	consentito Incremento	consentito Decremento
B6		0	-0,5	3	0,5	1E+30
C6		100	0	7	1E+30	1

Vincoli

Cella	Nome	Valore finale	ombra Prezzo	Vincolo a destra	consentito Incremento	consentito Decremento
A3		100	0	1000	1E+30	900
A4		400	1,75	400	3600	400

2. Post-optimality analysis regarding the objective function coefficients; the optimal basic solution will remain the same provided that: the unit selling price of table wine increases by at most 0.5, and the unit selling price of dessert wine decreases by at most 1.

Celle variabili

Cella	Nome	Valore finale	ridotto Costo	oggettivo Coefficiente	consentito Incremento	consentito Decremento
B6	Tavola	0	-0,5	3	0,5	1E+30
C6	Dess.	100	0	7	1E+30	0,999999999

Vincoli

Cella	Nome	Valore finale	ombra Prezzo	Vincolo a destra	consentito Incremento	consentito Decremento
A3	uva A	100	0	1000	1E+30	900
A4	uva B	400	1,75	400	3600	400

3. Post-optimality analysis regarding the right-hand side values; the optimal basic solution will remain unchanged provided that: the availability of grape A decreases by at most 900, and the availability of grape B increases by at most 3600 and decreases by at most 400.

Celle variabili

Cella	Nome	Valore finale	ridotto Costo	oggettivo Coefficiente	consentito Incremento	consentito Decremento
B6	Tavola	0	-0,5	3	0,5	1E+30
C6	Dess.	100	0	7	1E+30	0,999999999

Vincoli

Cella	Nome	Valore finale	ombra Prezzo	Vincolo a destra	consentito Incremento	consentito Decremento
A3	uva A	100	0	1000	1E+30	900
A4	uva B	400	1,75	400	3600	400

Consistent with the example of the economic interpretation of duality, we present a possible partial interpretation of these data.

Let us consider grape resource B: its shadow price is 1.75. Then:

• it is profitable to purchase grape B at a unit price $p < 1.75$; this would guarantee an increase in profit (i.e., the value of the optimal solution) equal to $1.75 - p$ per unit of grape B; however, this applies only to a quantity not exceeding 3,600 units, since the optimal basic solution changes once this value is exceeded;

• it is profitable to sell grape B at a unit price $p > 1.75$; this would ensure an increase in profit (i.e., the value of the optimal solution) equal to $p - 1.75$ per unit of grape B; however, this applies only to a quantity not exceeding 400 units, since the optimal basic solution changes once this value is exceeded. □

Course pack for “Operations research and logistics” (9 e.c.t.s.) 2026/2027

This course pack is freely adapted from the following books by Prof. Matteo Fischetti and Prof. Antonio Sassano, respectively:

[Fischetti]: M. Fischetti, *Lezioni di Ricerca Operativa*, Ed. Libreria Progetto Padova (2014).
[Sassano]: A. Sassano, *Modelli e Algoritmi della Ricerca Operativa*, Ed. Franco Angeli (1999).

Module 2

Module 2 of the "Operations Research and Logistics" course serves as a potential extension of Module 1, building upon the material previously covered.

As previously discussed, Operations Research deals with modeling and solving optimization problems, that is, problems where a decision-maker must select an action from a set of feasible actions to maximize a specific utility, by framing them – if possible – as Mathematical Programming problems. Specifically, we have examined two fundamental classes of Mathematical Programming problems: Linear Programming (LP) and Integer Linear Programming (ILP). These nevertheless appear to serve as a benchmark for many practical optimization problems.

Regarding logistics, we cite below the definition provided by the Italian Logistics Association (AILOG): it is *the set of organizational, managerial, and strategic activities governing the flow of materials and related information within a company, spanning from their origins at suppliers to the delivery of finished products to customers and subsequent after-sales service*.

In Module 2, we will examine certain optimization problems, within potential logistics contexts, and model them as Linear (Integer) Programming problems, in accordance with Note 2 below.

During Module 1, we have already encountered optimization problems (and their corresponding models) that could fall within the scope of logistics; examples include production involving the assembly or disassembly of resources, transportation, the minimum-cost path problem, production planning (i.e., inventory management), and facility location.

In Module 2, we will cover the following problems:

A. Some examples: vehicle loading; load balancing between vehicles; minimizing the number of vehicles; distribution network (re-definition); service network (re-definition); supply network (re-definition); a trip in highway

B. Minimum Spanning Tree Problem (MST)

C. Transportation Problem: two generalizations

D. Traveling Salesman Problem (TSP)

E. Vehicle Routing Problem (VRP)

The written material for Module 2 consists of: (i) the book by Prof. Matteo Fischetti (see reference [2] in the course program), (ii) this set of notes.
As we have already seen in detail, in Chapter 1, a possible description of the potential utility of Operations Research is provided by the following scheme:

optimization problem (informal) $\rightarrow$ mathematical model (formal) $\rightarrow$ solution

which summarizes the following:
• first step: starting from a given optimization problem (in informal terms) and arriving – if possible – at a mathematical model of it (in formal terms); this step relies on human skill and experience;
• second step: starting from the aforementioned mathematical model and arriving – if possible – at its resolution, that is, the calculation of an "optimal solution" to the given optimization problem; this step is achieved using specialized software.

Specifically: an *optimization problem* is understood as a problem in which a decision-maker must choose, from a set of feasible decisions/solutions, a decision/solution that optimizes a certain utility; a *mathematical model* is understood, in accordance with the book [Fischetti], as a Mathematical Programming problem.

A ***Mathematical Programming*** problem consists of computing

$$\textbf{min}\ \{f(x) : x \in X\}$$

where:
$X \subseteq \mathbb{R}^n$ ($n \in \mathbb{N}$) it is a set called the *set of feasible solutions* and
$f : X \rightarrow \mathbb{R}$ is a function called the objective function.
Note: **min** can be replaced with **max** by changing the sign of the function.

As with Module 1, some hours of Module 2 will be dedicated to the second step using the Excel solver; this aspect, in particular, will carry greater weight in Module 2.

Note 1: As previously noted, the primary purpose of modeling and solving an optimization problem as a Mathematical Programming problem is to provide "decision support"; indeed, the resulting model is often merely an approximation of reality. In particular, the solution obtained can offer guidance both on how to proceed regarding the decision directly linked to the problem and on how to handle decisions indirectly related to it [for instance, we have seen that *shadow prices*, in the context of production involving the combination of resources, identify directions and conditions for the potential expansion of that production].

Note 2: As we have already seen – albeit in a manner sufficient only for our current purpose, while referring the reader to Computer Science texts for a formal treatment – we briefly review below some concepts regarding computational complexity. Mathematical programming problems are classified [as far as possible] as *easy* or *hard* based on the computational complexity of the best solution algorithm [defined in advance] known for solving a generic instance of the problem:
Easy problems, also known as "polynomial problems", include Linear Programming (LP) problems; for instance, the project scheduling problem is easy. Furthermore, certain problems not initially modeled as LP can ultimately be reduced to LP (and are thus *easy*) through results concerning Totally Unimodular (TUM) matrices; examples include the Assignment, Transportation, Minimum Cost Path, and Maximum Flow problems. Regarding optimization problems that can be formulated as LP problems (which can, of course, be solved to optimality using standard LP methods, such as the Simplex Algorithm), "exact methods" are often introduced or studied to find an optimal solution without relying on standard LP techniques; this is because having fast methods available is beneficial, particularly for large problem instances.
Hard problems, also known as "NP-hard" problems", include Integer Linear Programming (ILP) problems; for instance, the facility location problem is hard. Regarding optimization problems that can be formulated as ILP problems (which can, of course, be solved to optimality using standard ILP methods, such as the Branch and Bound Algorithm), "heuristic methods" are often introduced or studied to find a solution that is presumably good, though not necessarily optimal, without relying on standard ILP techniques; this is because a standard method for ILP can be slow, even for small problem instances; consequently, the decision-maker (if they so choose) can employ a "heuristic method" which, while guaranteeing a presumably good—though not necessarily optimal solution, has the advantage of speed.

A. Some examples

Below, we look at seven optimization problems – set in possible logistics contexts – that can be modeled as Linear (Integer) Programming problems; in particular, during Module 1, we have already seen how to model similar situations in other contexts.

A1. Vehicle loading

A company has a vehicle available to deliver certain items; the vehicle has a capacity b in terms of *size* (which may refer to weight, volume, or another metric). The company needs to deliver n items, but their total size exceeds capacity b; therefore, the company must select which items to deliver during the vehicle's first trip. Specifically, for $i = 1, \ldots, n$, each item i has a *size* a_i (defined as above) and a *utility* u_i (which may refer to the item's value or the utility to the company of delivering it).
The problem is to select the items to deliver (i.e., to load onto the vehicle), subject to the size constraint, so as to maximize the total utility of the selected items.

Decision variables:
x_i for $i = 1, \ldots, n$;
$x_i = 1$ if item i is selected;
$x_i = 0$ if item i is not selected.

Model: max $u_1x_1 + u_2x_2 + \ldots + u_nx_n$

$a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b$

$x_i \in \{0, 1\}$ for $i = 1, \ldots, n$

Exercise: Solve the following instance of the problem using Excel.

$\boldsymbol{n = 7}$; $\boldsymbol{b} = 70$

	1	**2**	**3**	**4**	**5**	**6**	**7**
$\boldsymbol{a_i}$	12	8	16	19	24	10	9
$\boldsymbol{u_i}$	7	5	20	25	30	14	10

A2. Load balancing between vehicles

A company has two vehicles, C_1 and C_2, available to deliver n items. For $i = 1, \ldots, n$, each item i has a *size* a_i (which may refer to weight, volume, or another metric). Assume that both vehicles have sufficient capacity to accommodate the load.
The problem is to select which items to load onto C_1 and C_2, respectively, so as to balance their total loads as much as possible.

Decision variables:
x_i for $i = 1, \ldots, n$
$x_i = 1$ if item i is loaded onto C_1
$x_i = 0$ if item i is not loaded onto C_1 (i.e., it is loaded onto C_2)

Model: $\min \sum_{i=1}^{n} a_i x_i$

$$\sum_{i=1}^{n} a_i x_i \geq \sum_{i=1}^{n} a_i \ / 2$$

$x_i \in \{0, 1\}$ for $i = 1, \ldots, n$

We next consider a generalization to the case of m vehicles.

A company has m vehicles, let us call them $C_1, \ldots, C_m$, available to deliver n items. For each item i (where $i = 1, \ldots, n$), there is an associated *size* a_i (which may refer to weight, volume, or another metric). Assume that the vehicles have sufficient capacity to accommodate the load.
The problem is to select which items to load onto $C_1, \ldots, C_m$ so as to balance their respective total loads as much as possible.

Decision variables:
x_{ij} for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$
$x_{ij} = 1$ if item i is loaded onto C_j
$x_{ij} = 0$ if item i is not loaded onto C_j
y_{max} = total size of the vehicle with the maximum overall size
y_{min} = total size of the vehicle with the minimum overall size

Model: min $y_{max} - y_{min}$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \text{for } i = 1, \ldots, n \quad \text{(each item is loaded)}$$

$$\sum_{i=1}^{n} a_i x_{ij} \leq y_{max} \qquad \text{for } j = 1, \ldots, m$$

$$\sum_{i=1}^{n} a_i x_{ij} \geq y_{min} \qquad \text{for } j = 1, \ldots, m$$

$$x_{ij} \in \{0, 1\} \qquad \text{for } i = 1, \ldots, n$$

$$y_{max} \geq 0$$

$$y_{min} \geq 0$$

Exercise: Solve the following instance of the problem using Excel.

$\boldsymbol{n} = 7$; $\boldsymbol{m} = 3$

	1	**2**	**3**	**4**	**5**	**6**	**7**
$\boldsymbol{a_i}$	12	8	16	19	24	10	9

A3. Minimizing the number of vehicles

A company has m vehicles available, labeled 1, …, m, to deliver n items. Each item i (where i = 1, …, n) has a *size* a_i (which may refer to weight, volume, or another metric). For each vehicle j (where j = 1, …, m), there is a capacity K_j in terms of size.
The problem is to determine which trucks to use (for this delivery), subject to the size constraints, so as to minimize the total number of vehicles used.

Decision variables:
x_{ij} for i = 1, …, n and for j = 1, …, m
x_{ij} = 1 if item i is loaded onto C_j
x_{ij} = 0 if item i is not loaded onto C_j
y_j for j = 1, …, m;
y_j = 1 if C_j is used, namely, if some item is loaded onto C_j
y_j = 0 if C_j is not used, namely, if no item is loaded onto C_j

Model: min $\sum_{j=1}^{m} y_j$

$\sum_{j=1}^{m} x_{ij} = 1$ for i = 1, …, n (each item is loaded)

$\sum_{i=1}^{n} a_i x_{ij} \le K_j$ for j = 1, …, m

$\sum_{i=1}^{n} x_{ij} \le M y_j$ for j = 1, …, m

(where M is a very large scalar)

$x_{ij} \in \{0, 1\}$ for i = 1, …, n and for j = 1, …, m

$y_j \in \{0, 1\}$ for j = 1, …, m

Exercise: Solve the following instance of the problem using Excel.

$\boldsymbol{n}$ = 7 ; $\boldsymbol{m}$ = 3

	1	**2**	**3**	**4**	**5**	**6**	**7**
$\boldsymbol{a_i}$	12	8	16	19	24	10	9

	1	**2**	**3**
$\boldsymbol{K_j}$	50	50	50

A4. Distribution network (re-definition)

A company takes over an existing distribution network for a specific good, consisting of m warehouses supplying n stores; specifically, each store is supplied by a single warehouse.

Parameters:
:: $M = \{1, \ldots, m\}$ = set of warehouses
:: $N = \{1, \ldots, n\}$ = set of stores
:: d_j = monthly capacity of warehouse j
:: r_i = monthly demand of store i
:: C_j = cost of confirming warehouse j (over the fixed time horizon)
:: c_{ij} = cost of supplying store i from warehouse j (over the fixed time horizon)

Problem: The company wishes to redefine its network of warehouses, specifically, to explore the possibility of reducing the number of warehouses so that only a subset remains active.
In particular: regarding the costs of maintaining warehouses and the costs of replenishing stores from those warehouses, the company considers a fixed time horizon [e.g., three and a half years]; regarding warehouse capacities and store demand, the company considers a one-month time horizon.

Decision variables:
x_{ij} for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$
x_{ij} = 1 if store i is supplied by warehouse j
x_{ij} = 0 otherwise
y_j for $j = 1, \ldots, m$
y_j = 1 if warehouse j is to be confirmed
y_j = 0 otherwise

Model: min $\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{j=1}^{m} C_j y_j$

$\sum_{j=1}^{m} x_{ij} = 1$ for $i = 1, \ldots, n$ (each store is supplied by a warehouse)

$\sum_{i=1}^{n} r_i x_{ij} \leq d_j$ for $j = 1, \ldots, m$

$\sum_{i=1}^{n} x_{ij} \leq M y_j$ for $j = 1, \ldots, m$

(where M is a very large scalar)

$x_{ij} \in \{0, 1\}$ for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$
$y_j \in \{0, 1\}$ for $j = 1, \ldots, m$

Exercise: Solve the following instance of the problem using Excel.

$n = 4$; $m = 3$

c_{ij}	**1**	**2**	**3**
1	2	3	4
2	3	5	3
3	4	4	2
4	5	1	3

	1	**2**	**3**
C_j	15	28	25
d_j	120	250	200

	1	**2**	**3**	**4**
r_i	70	40	100	70

A5. Service network (re-definition)

A region wishes to redefine the departments within its hospital network following resource cuts. For the sake of simplicity, let us focus on a single type of department, for example, Orthopedics.

Parameters:
:: the Region sets a time horizon, for example, one year;
:: $D = \{D_1, \ldots, D_n\}$ = set of [indicative] districts into which the Region is divided;
:: $R = \{R_1, \ldots, R_m\}$ = set of Orthopedics departments in the Region;
:: d_i = number of (expected) patients estimated to require admission to one of the departments in R (i.e., an Orthopedics department) from district D_i (for $i = 1, \ldots, n$) over one year;
:: r_j = maximum number of (expected) patients estimated to be admissible to department R_j (per $j = 1, \ldots, m$) over one year;
:: C_j = "fixed cost" = estimated cost solely for maintaining department R_j (for $j = 1, \ldots, m$) over one year;
:: c_{ij} = "marginal cost" = estimated cost of admitting a patient from district D_i (for $i = 1, \ldots, n$) to department R_j (per $j = 1, \ldots, m$) over one year [this cost can be understood, for example, as $c_j + q_{ij}$, where: c_j is the estimated cost incurred by the Region to admit a generic patient to department R_j, and q_{ij} is the estimated cost incurred by the patient from district D_i to be admitted to department R_j].

Problem: The Region wishes to determine which departments to retain from the set $\{R_1, \ldots, R_m\}$, subject to the constraint that patient admissions be managed so as to guarantee each (expected) patient a place in one of the departments in R, while minimizing the sum of "fixed costs" and "marginal costs."

Decision variables:
x_{ij} for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$
x_{ij} = number of (expected) patients from district D_i admitted to ward R_j
y_j for $j = 1, \ldots, m$
$y_j = 1$ if the R_j department is to be confirmed
$y_j = 0$ otherwise

Modello: $\min \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{j=1}^{m} C_j y_j$

$$\sum_{j=1}^{m} x_{ij} \geq d_i \qquad \text{for } i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} \leq r_j \qquad \text{for } j = 1, \ldots, m$$

$$\sum_{i=1}^{n} x_{ij} \leq M\, y_j \qquad \text{for } j = 1, \ldots, m$$

(where M is a very large scalar)

$x_{ij} \geq 0$ for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$

$y_j \in \{0, 1\}$ for $j = 1, \ldots, m$

Exercise: Solve the following instance of the problem using Excel; specifically, consider the case where $n = 4$ and $m = 4$, imagining, for example, that the districts are the provinces of Abruzzo and the units are the corresponding units located in the respective provincial capitals.

$\boldsymbol{n} = 4$; $\boldsymbol{m} = 4$

c_{ij}	**1**	**2**	**3**	**4**
1	1	3	5	7
2	3	1	4	5
3	5	4	1	2
4	7	5	2	1

	1	**2**	**3**	**4**
C_j	4000	5200	3800	7000
r_j	320	220	300	450

	1	**2**	**3**	**4**
d_i	120	350	200	400

A6. Supply network (re-definition)

A Middle Eastern state wishes to redefine its gas supply in order to cover a supply deficit caused by "force majeure", over a time horizon of n months, taking into account that it can purchase gas from m (potential) suppliers.

Parameters:
:: the State sets a time horizon of n months;
:: $M = \{M_1, \ldots, M_n\}$ = set of months in the time horizon;
:: $F = \{F_1, \ldots, F_m\}$ = set of (potential) suppliers;
:: d_i = quantity of gas required by the State in month M_i (for $i = 1, \ldots, n$);
:: b_{ij} = maximum quantity of gas that can be purchased in month M_i (for $i = 1, \ldots, n$) from supplier F_j (for $j = 1, \ldots, m$) [e.g., for a supplier requiring the construction of a pipeline, this maximum quantity may be zero during the initial months];
:: C_j = "fixed cost" = estimated fixed cost incurred if any non-zero quantity of gas is purchased from supplier F_j (for $j = 1, \ldots, m$) during the time horizon [e.g., for pipeline construction];
:: c_{ij} = "marginal cost" = estimated marginal cost for purchasing one unit of gas in month M_i (for $i = 1, \ldots, n$) from supplier F_j (for $j = 1, \ldots, m$).

Problem: The State wishes to determine the quantities of gas to be purchased each month from each supplier, subject to necessity constraints, in order to minimize total costs.

Decision variables:
x_{ij} for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$
x_{ij} = units of gas to be purchased in month M_i (for $i = 1, \ldots, n$) from supplier F_j (for $j = 1, \ldots, m$)
y_j for $j = 1, \ldots, m$
$y_j = 1$ if the (potential) supplier F_j is selected as the supplier
$y_j = 0$ otherwise

Model: min $\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{j=1}^{m} C_j y_j$

$\sum_{j=1}^{m} x_{ij} \geq d_i$ for $i = 1, \ldots, n$

$x_{ij} \leq b_{ij}$ for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$

$\sum_{i=1}^{n} x_{ij} \leq M y_j$ for $j = 1, \ldots, m$

(where M is a very large scalar)

$x_{ij} \geq 0$ for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$

$y_j \in \{0, 1\}$ for $j = 1, \ldots, m$

Note: the model in example A6 is (almost) identical to the model in example A5.

Exercise: Solve the following instance of the problem using Excel.

$n = 5$; $m = 4$

c_{ij}	**1**	**2**	**3**	**4**
1	5	5	3	4
2	5	5	3	4
3	7	6	3	4
4	7	6	3	4
5	7	7	3	4

	1	**2**	**3**	**4**
C_j	5000	8000	14000	7000

	1	**2**	**3**	**4**	**5**
d_i	300	420	500	550	570

b_{ij}	**1**	**2**	**3**	**4**
1	500	200	0	200
2	500	200	0	200
3	500	200	700	200
4	500	200	700	200
5	500	200	700	200

A7. A trip in highway

This example is loosely based on the presentation at https://slideplayer.it/slide/935618/
by Prof. Claudio Arbib (consulting this reference is recommended).

A person who regularly drives from Pescara to Rome via the highway wants to minimize fuel consumption while adhering to a maximum travel time.
Observing that fuel consumption depends on the road gradient and the car's speed, the driver decides to divide the route (from tollbooth to tollbooth) into n segments (each with a uniform gradient) and, for simplicity, to consider only m possible speeds (e.g., 70 km/h, 80 km/h, ..., 130 km/h). Finally, the following parameters are calculated and set:
:: let c_{ij} be the fuel consumption/cost to travel segment i (for $i = 1, \ldots, n$) at speed j (for $j = 1, \ldots, m$);
:: let t_{ij} be the time taken to travel segment i (for $i = 1, \ldots, n$) at speed j (for $j = 1, \ldots, m$);
:: let T be the set maximum travel time.
The problem is to determine the speed to use for each segment, subject to the maximum travel time constraint, so as to minimize the total fuel consumption/cost.

Decision variables:
x_{ij} for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$
$x_{ij} = 1$ if the distance i is traveled at speed j
$x_{ij} = 0$ otherwise

Model: min $\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \text{for } i = 1, \ldots, n$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} t_{ij}x_{ij} \le T$$

$$x_{ij} \in \{0, 1\} \quad \text{for } i = 1, \ldots, n \text{ and for } j = 1, \ldots, m$$

Exercise: Solve the following instance of the problem using Excel.

$n = 5$; $m = 4$; $T = 100$

c_{ij}	**100 Km/h**	**110 Km/h**	**120 Km/h**	**130 Km/h**
1	5	6	7	8
2	6	7	8	9
3	8	8	9	11
4	4	5	6	7
5	5	6	7	8

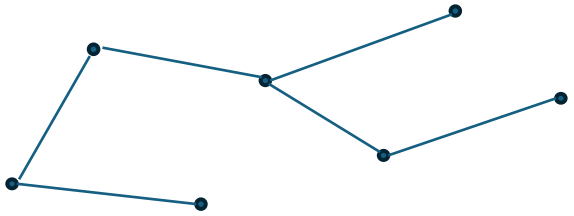
t_{ij}	**100 Km/h**	**110 Km/h**	**120 Km/h**	**130 Km/h**
1	24	22	20	18
2	12	11	10	9
3	37	33	30	27
4	19	17	15	13
5	30	27	25	23

B. Minimum Spanning Tree Problem (MST)

These notes are freely adapted from Professor Matteo Fischetti's book (look up "Minimum-cost trees: Efficient algorithms, Kruskal's algorithm" in the index).

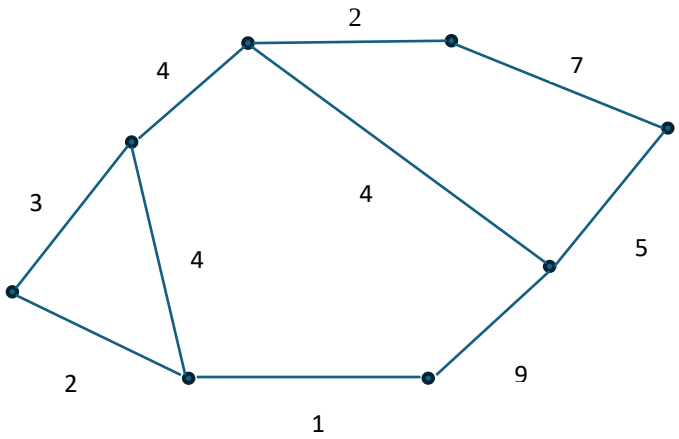
Recall that a *tree* is a connected graph that contains no cycles.
In particular, the following theorem holds.

***Theorem* B.1:** A tree with n vertices has $n - 1$ edges. □



Problem definition: Let $G = (V, E)$ be a graph and let $c : E \to \mathbb{R}$ be a *cost* (or *weight*, or *length*) function that assigns a cost c_e to each $e \in E$; specifically, for any $T \subseteq E$, we denote the cost of the subset T by $c(T) = \sum_{e \in T} c_e$. The problem is to determine a subgraph $G_T(V, T)$ of G, with $T \subseteq E$, such that $G_T(V, T)$ is a tree with minimum cost $c(T)$. □

Note that $G = (V, E)$ and $G_T(V, T)$ share the same set of vertices, namely, the set V, which means that $G_T(V, T)$ is a subgraph of G comprising all the vertices of G and only a specific subset T of G's edges that forms a tree; such a tree is called a "spanning tree of G." The problem, therefore, consists of finding a "spanning tree of G" with minimum cost.



Notation: for every $S \subseteq V$, we write $E(S) = E \cap (S \times S)$; in other words, $E(S)$ is the set of edges of G whose endpoints are vertices in S.

ILP model

Decision variables
x_e , for any $e \in E$
$x_e = 1$, if e will be part of the minimum-cost spanning tree for G
$x_e = 0$, altrimenti

Model min $\sum_{e \in E} c_e x_e$

$\sum_{e \in E} x_e = n - 1$
$\sum_{e \in E(S)} x_e \leq |S| - 1$, foe every $S \subseteq V$
$x_e \geq 0$, for any $e \in E$
$x_e \leq 1$, for any $e \in E$
x_e integer , for any $e \in E$

It can be shown that the polyhedron associated with this formulation is integral [even if the matrix is not TUM], making it possible to eliminate the constraints "x_e integer, for any $e \in E$" from the formulation [as we saw in the first part of the course]; however, this formulation involves an exponential number of constraints, given that V has an exponential number of subsets S.

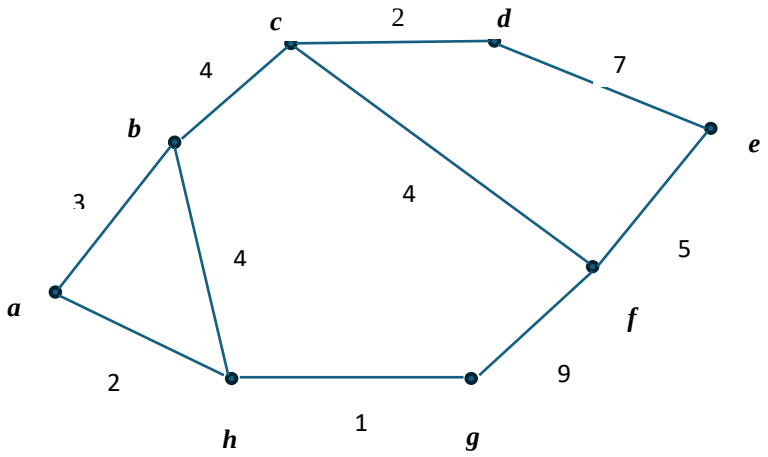
***Theorem* B.2:** Given a graph $G = (V, E)$, let $T' \subset T \subseteq V$, where $G_T(V, T)$ is an optimal solution to the SST problem. Then, for every $e \in E$ such that
$e = \arg\min \{ c_e : e \in E \setminus T' ; T' \cup \{e\}$ does not form a cycle in $G\}$,
it holds that $T' \cup \{e\} \subseteq T^*$, where $G_{T^*}(V, T^*)$ is an optimal solution to the SST problem. □

Kruskal's algorithm
Input: graph $G = (V, E)$, $|V| = n$, $c : E \to \mathbb{R}$
Output: subgraph $G_T(V, T)$ of G that is an optimal solution to the MST problem.

Step 0) $T := \{\varnothing\}$.
Step 1) Sort the edges of G by increasing cost; let them be $\{e_1, e_2, \ldots, e_n\}$ with $c_{e1} \leq c_{e2} \leq \ldots \leq c_{en}$.
Step 2) For i = 1, …, m:
:: if $T \cup \{e_i\}$ does not form a cycle, then $T := T \cup \{e_i\}$
:: if $|T| = n - 1$, then STOP [$G_T(V, T)$ is an optimal solution to the SST problem].

Solved Exercise

Solve the SST problem for the instance shown in the figure using Kruskal's algorithm.



Solution

The graph G shown in the figure has $n = 8$ vertices.

Step 0) $T := \{\varnothing\}$.

Step 1) Sort the edges of G by increasing cost; let them be $\{e_1, e_2, \ldots, e_n\}$ with $c_{e1} \leq c_{e2} \leq \ldots \leq c_{en}$.

Let them be {(***g***,***h***), (***a***,***h***), (***c***,***d***), (***a***,***b***), (***b***,***h***), (***b***,***c***), (***c***,***f***), (***e***,***f***), (***d***,***e***), (***f***,***g***)}

Step 2)

$T \cup \{(\boldsymbol{g},\boldsymbol{h})\}$ does not form a cycle, then $T := T \cup \{(\boldsymbol{g},\boldsymbol{h})\}$

$T \cup \{(\boldsymbol{a},\boldsymbol{h})\}$ does not form a cycle, then $T := T \cup \{(\boldsymbol{a},\boldsymbol{h})\}$

$T \cup \{(\boldsymbol{c},\boldsymbol{d})\}$ does not form a cycle, then $T := T \cup \{(\boldsymbol{c},\boldsymbol{d})\}$

$T \cup \{(\boldsymbol{a},\boldsymbol{b})\}$ does not form a cycle, then $T := T \cup \{(\boldsymbol{a},\boldsymbol{b})\}$

$T \cup \{(\boldsymbol{b},\boldsymbol{h})\}$ forms a cycle, formed by (***b***,***h***), (***a***,***b***), (***a***,***h***)

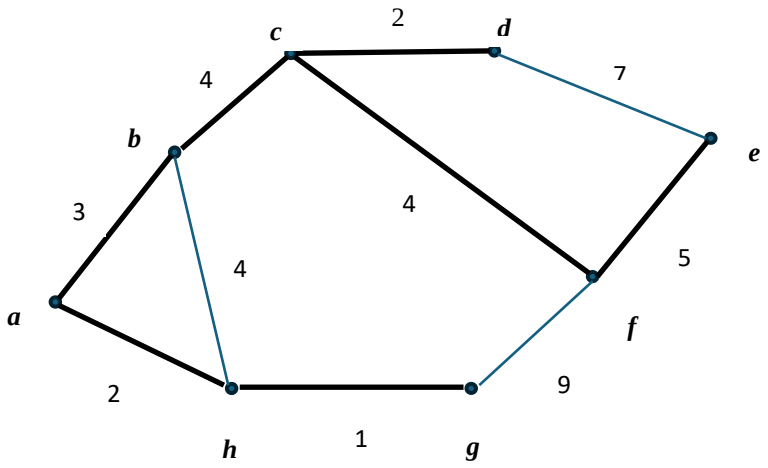
$T \cup \{(\boldsymbol{b},\boldsymbol{c})\}$ does not form a cycle, then $T := T \cup \{(\boldsymbol{b},\boldsymbol{c})\}$

$T \cup \{(\boldsymbol{b},\boldsymbol{c})\}$ does not form a cycle, then $T := T \cup \{(\boldsymbol{b},\boldsymbol{c})\}$

$T \cup \{(\boldsymbol{c},\boldsymbol{f})\}$ does not form a cycle, then $T := T \cup \{(\boldsymbol{c},\boldsymbol{f})\}$

$T \cup \{(\boldsymbol{e},\boldsymbol{f})\}$ does not form a cycle, then $T := T \cup \{(\boldsymbol{e},\boldsymbol{f})\}$

$|T| = n - 1$, then STOP [$G_T(V, T)$ is an optimal solution to the SST problem]. □



Concluding remarks on applications

The SST problem appears to have numerous practical applications.

A classic example involves designing an electrical grid for a group of housing complexes (or, more generally, locations) that must all be supplied by a power plant; the power plant and the housing complexes can be viewed as the vertices of a graph, with edges representing the possibility (and associated cost) of a direct connection between the corresponding vertices. Thus, the problem of designing a minimum-cost electrical grid is precisely the SST problem.

Similarly, other network design applications can be envisioned, such as planning an initial highway network or a high-speed railway network.

C. Transportation Problem: two generalizations

We revisit the transportation problem, which we already encountered in the Operations Research course, along with two generalizations; notably, the transportation problem is *easy* [as we have already seen], and these two generalizations remain *easy* problems.

Transportation problem

There are n depots and m outlets. Each depot A_i , for $i \in \{1, \ldots, n\}$, produces $a_i \geq 0$ units of a certain good, and each outlet B_j , for $j \in \{1, \ldots, m\}$, requires at least $b_j \geq 0$ units of the same good. Let $c_{ij} \geq 0$ be the cost of transporting one unit of the good from A_i to B_j.
See Figure 1 for $n = 2$ and $m = 3$.

The problem is to plan transportation so as to minimize the total cost.

Decision variables:

x_{ij} = quantity of goods transported from the A_i depot to the B_j outlet

Model: min $\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$

$\sum_{j=1}^{m} x_{ij} \leq a_i$ for $i = 1, \ldots, n$ (availability constraints)

$\sum_{i=1}^{n} x_{ij} \geq b_j$ for $j = 1, \ldots, m$ (request constraints)

$x_{ij} \geq 0$ for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$

x_{ij} integer for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$ (if necessary)

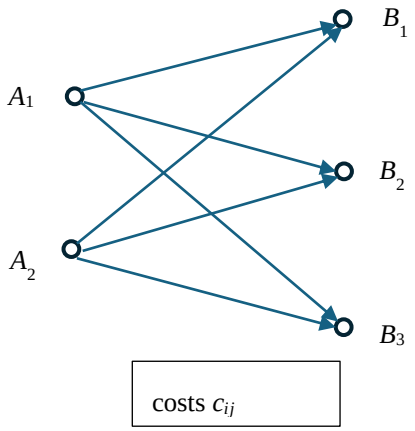


Figure 1

Below, we present two generalizations of the transportation problem:
:: the first is the "Minimum-Cost Flow" problem, which lends itself to many practical applications; both in its direct form, as a generalization of the transportation problem where the connection between depots and outlets is not necessarily direct but consists of a network, and in its more abstract form, such as in waste collection and disposal processes;
:: the second is a multi-period generalization of the transportation problem; while essentially a theoretical exercise, it can have plausible practical applications, given that transportation costs are often set in advance and frequently depend on the time variable (such as airline ticket prices).

C1. Minimum-Cost Flow (Min-Cost-Flow)

A network $N = (\sigma, \tau, V, E, w, c)$ is a directed graph (V, E) together with a *source* $\sigma \in V$ having an in-degree of 0 [i.e., no edge enters node σ], a *sink* $\tau \in V$ having an out-degree of 0 [i.e., no edge leaves node τ], an "edge-capacity" function $w : E \to \mathbb{R}.$, and an "edge-cost" function $c : E \to \mathbb{R}.$.
A *flow f* in *N* is a vector in $\mathbb{R}^{|E|}$ [with a component $f(u, v)$ for each $(u, v) \in E$] such that:

(i) $0 \leq f(u, v) \leq w(u, v)$ for any $(u, v) \in E$

(ii) $\sum_{(u,v)\in E} f(u,v) = \sum_{(v,u)\in E} f(v,u)$ for any $v \in V \setminus \{\sigma, \tau\}$.

The *value* of *f* is the quantity: $\sum_{(\sigma,v)\in E} f(\sigma, v)$ (where σ is the *source*)

The *cost* of *f* is the quantity: $\sum_{(u,v)\in E} c(u,v)\, f(u,v)$

Given a network $N = (\sigma, \tau, V, E, w, c)$ and a flow value v_f, the *Min-Cost Flow problem* with respect to (N, v_f) is to calculate a flow *f* in *N* with value v_f that has minimum cost.

Below, we show that the *transportation problem* can be represented as a *Min-Cost Flow problem*, as follows, with reference to a generic instance

The network $N = (\sigma, \tau, V, E, w, c)$ is defined as follows:

• construct a directed graph $G = (V, E)$ where
$V = \{A_i : i = 1, \ldots, n\} \cup \{B_j : j = 1, \ldots, m\}$;

$E = \{(A_i, B_j) : i = 1, \ldots, n\ ; j = 1, \ldots, m\}$;

• add a vertex σ and the arcs $\{(\sigma, A_i) : i = 1, \ldots, n\}$;

• add a vertex τ and the arcs $\{(B_j, \tau) : j = 1, \ldots, m\}$;

• the elements of V and E are updated accordingly;

• the "edge-capacities" function $w : E \to \mathbb{R}$ is defined as follows:

$w(\sigma, A_i) = a_i$, for $i = 1, \ldots, n$;
$w(B_j, \tau) = b_j$, for $j = 1, \ldots, m$;
$w(e) = \infty$, for the remaining edges $e \in E$;

• the "edge-costs" function $c : E \to \mathbb{R}$ is defined as follows:

$c(A_i, B_j) = c_{ij}$, for $i = 1, \ldots, n$, for $j = 1, \ldots, m$
$c(e) = 0$, for the remaining edges $e \in E$;

Then the *transportation problem* can then be solved as a *Min-Cost-Flow problem* with respect to the pair (N, v_f), where N is the network defined above and $v_f = \sum_{j=1}^{m} b_j$.

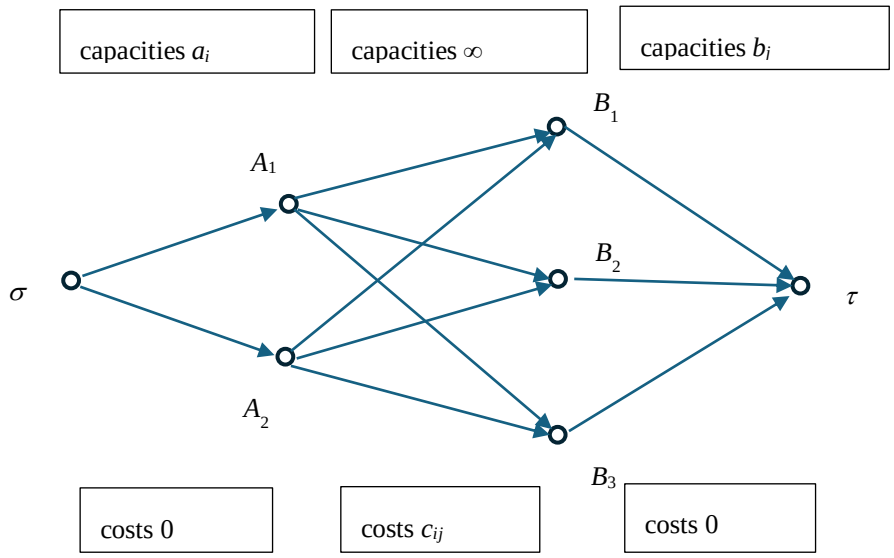


Figure 2

C2. A multi-period generalization of the transportation problem

Let *problem P* be the following multi-period generalization of the *transportation problem*:

There are r depots, s outlets, and t time periods (e.g., weeks) forming a time horizon (e.g., a month). We assume there are t copies of the transportation problem, one for each time period, linked by their

sequence within the time horizon. Each depot A_{ik}, for $i \in \{1, \ldots, n\}$ and for $k \in \{1, \ldots, t\}$, produces $a_{ik} \geq 0$ units of a certain good. Each outlet B_{jk}, for $j \in \{1, \ldots, m\}$ and for $k \in \{1, \ldots, t\}$, requires at least $b_{jk} \geq 0$ units of the same good. Let $c_{ijk} \geq 0$ be the cost of transporting one unit of the good from F_{ik} to O_{jk} during the k-th period. Let $h_{ik} \geq 0$ (let $h'_{jk} \geq 0$) be the cost of storing one unit of the good at F_{ik} (in O_{jk}) at the end of the k-th period.
See Figure 3 for $n = 2$, $m = 3$, $t = 3$.

The problem is to plan transportation over a given time horizon so as to minimize the total cost of transportation and storage.

Decision variables:
x_{ijk} = quantity of the good transported from F_{ik} a O_{jk} (in the k-th period)
z_{ik} = quantity of good stored in F_{ik} (at the end of the k-th period);
z'_{ik} = quantity of good stored in O_{jk} (at the end of the k-th period).

A *solution* to problem P is determined by the values of
x_{ijk}, z_{ik}, z'_{ik} for $i = 1, \ldots, r$, for $j = 1, \ldots, s$, for $k = 1, \ldots, t$.

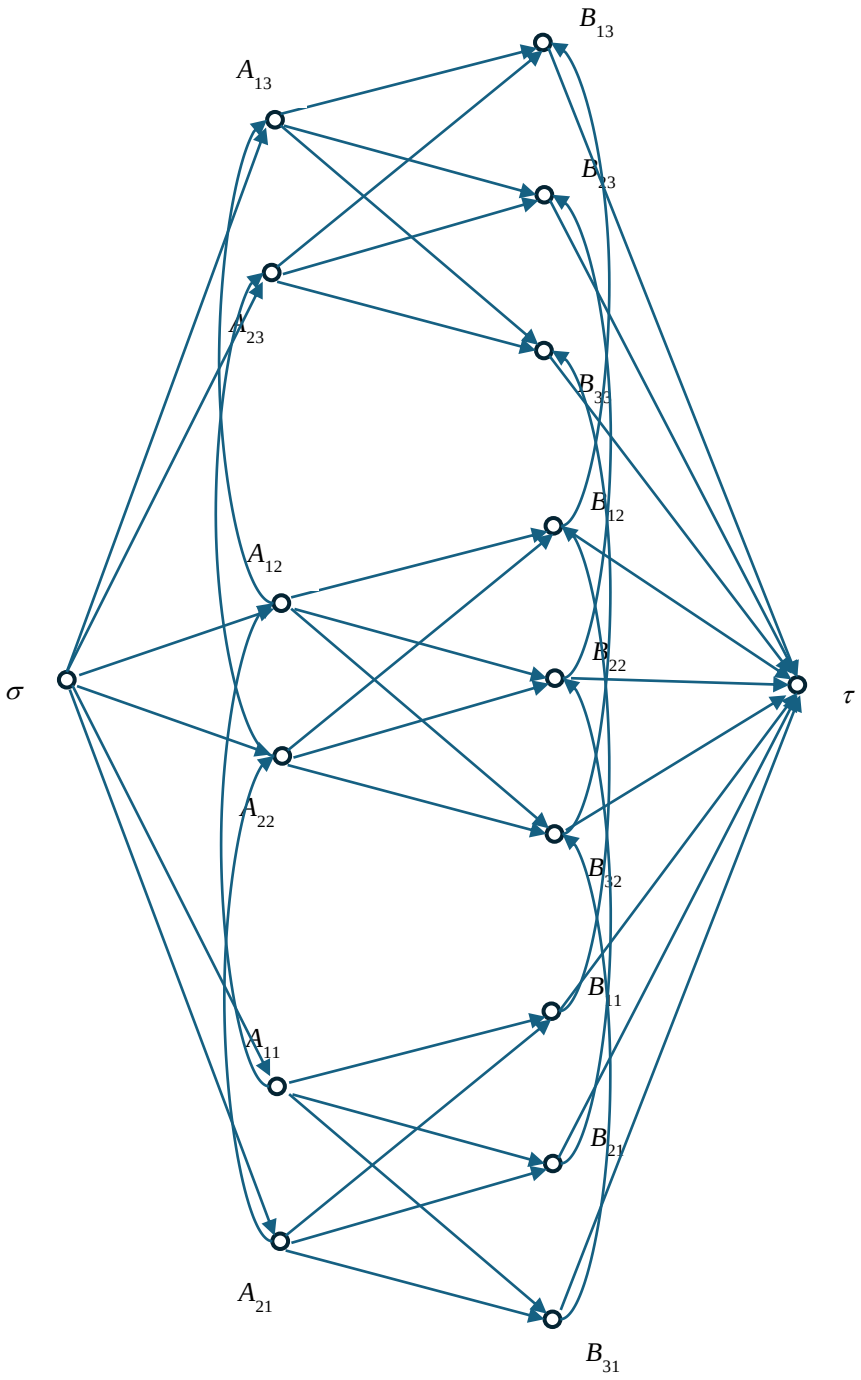

B_{13}
A_{13}
B_{23}
A_{23}
B_{33}
B_{12}
A_{12}
σ
B_{22}
τ
A_{22}
B_{32}
B_{11}
A_{11}
B_{21}
A_{21}
B_{31}

Figure 3

A Min-Cost Flow model for problem *P*

Below, we attempt to model *problem P* as a *Min-Cost Flow problem* on the following network. This network is constructed by taking into account that *problem P* comprises *t* copies of the transportation problem, specifically by adding a source node, a *sink* node, and arcs connecting consecutive periods.

- Construct t (disjoint) directed graphs $G_1, \ldots, G_t$ such that:

$G_k = (V_k, E_k)$, for $k = 1, \ldots, t$, where

$V_k = \{A_{ik} : i = 1, \ldots, r\} \cup \{B_{jk} : j = 1, \ldots, s\}$;

$E_k = \{(A_{ik}, B_{jk}) : i = 1, \ldots, r\ ; j = 1, \ldots, s\}$;

- add the vertex σ and the edges $\{(\sigma, A_{ik}) : i = 1, \ldots, r\ ; k = 1, \ldots, t\ \}$;

- add the vertex τ and the arcs $\{(B_{jk}, \tau) : j = 1, \ldots, s\ ; k = 1, \ldots, t\ \}$;

- add the arches $\{(A_{ik}, A_{ik+1}) : i = 1, \ldots, r\ ; k = 1, \ldots, t\text{-}1\}$;

- add the arches $\{(B_{jk}, B_{jk+1}) : j = 1, \ldots, s\ ; k = 1, \ldots, t\text{-}1\}$;

- the elements of V and E are defined as above;

- the "edge-capacities" function $w : E \to \mathbb{R}$ is defined as follows:

$w(\sigma, A_{ik}) = a_{ik}$, for $i = 1, \ldots, r$, for $k = 1, \ldots, t$;
$w(B_{jk}, \tau) = b_{jk}$, for $j = 1, \ldots, s$, for $k = 1, \ldots, t$;
$w(e) = \infty$, for the remaining edges $e \in E$;

- the "edge-costs" function $c : E \to \mathbb{R}$ is defined as follows:

$c(A_{ik}, B_{jk}) = c_{ijk}$, for $i = 1, \ldots, r$, for $j = 1, \ldots, s$, for $k = 1, \ldots, t$;
$c(A_{ik}, A_{ik+1}) = h_{ik}$, for $i = 1, \ldots, r$, for $k = 1, \ldots, t\text{-}1$;
$c(B_{jk}, B_{jk+1}) = h'_{ik}$, for $j = 1, \ldots, s$, for $k = 1, \ldots, t\text{-}1$;
$c(e) = 0$, for the remaining edges $e \in E$.

Then *problem P* can then be solved as a *Min-Cost-Flow problem* with respect to the pair (N, v_f), where N is the network defined above and $v_f = \sum_{j=1}^{s} \sum_{k=1}^{t} b_{jk}$.

D. Traveling Salesman Problem (TSP)

These notes are loosely based on the book by Prof. Antonio Sassano (see reference [4] in the syllabus) and the resource http://groups.di.unipi.it/optimize/Courses/RO2IG/aa1415/8-tsp_vrp.pdf by Prof. Laura Galli (consulting these is recommended).

Regarding this problem, various definitions (of varying generality) exist in the literature; below, we present what is perhaps the most general definition possible.

Recall that a *path* in a graph G is a sequence of consecutive edges e_1, ..., e_k of G, that is, edges of the form $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3)$, ..., $e_k = (v_k, v_{k+1})$; such a path c*onnects* v_1 to v_{k+1}; in particular, if $v_1 \equiv v_{k+1}$, it is called a *cycle*. Note that, according to this definition, a path (as well as a cycle) may pass through the same vertex multiple times.

Definition of the TSP: Let $G = (V, E)$ be a directed graph such that a directed path exists between every pair of vertices, and let $c_{ij} \geq 0$ be a cost assignment for each arc (i, j) of G. The problem is to determine a directed cycle of minimum cost [where the cost of a cycle is the sum of the costs of its constituent arcs] that visits all vertices of G. □

This problem is NP-hard, that is, *hard* (see Note 2).

In the Introduction, we saw that determining the minimum-cost path between two vertices of a graph is an *easy* problem. Thus, bearing in mind that a graph is said to be *complete* if it contains all possible edges [i.e., for every $i, j \in V$, $(i, j) \in E$], the TSP is often redefined as follows, with each weight c^*_{ij} representing the cost of a minimum-cost path between vertices i and j in the original graph G:

Definition (alternative) of the TSP: Let $G^* = (V, E^*)$ be a complete directed graph, and let $c^*_{ij} \geq 0$ be a cost assignment for each arc (i, j) of G^* . The problem is to determine a directed cycle of minimum cost [where the cost of a cycle is the sum of the costs of its constituent arcs] that visits all the vertices of G. □

In the following, we will examine two Integer Linear Programming models for the TSP [which yield an exact solution to the problem] and three heuristic methods for the TSP [which yield a heuristic solution to the problem].

ILP models for the TSP

Below, we present two ILP models for the TSP. Specifically, we assume that for each of these methods, the input is a TSP instance consistent with the alternative definition, that is, we assume that for each method:

Input: a complete directed graph $G = (V, E)$, with $V = \{v_1, \ldots, v_n\}$ and $c_{ij} \geq 0$ for every $(i, j) \in E$.
Output: a cycle in G that passes through all the vertices of G.
Thus, by the definition of the input, we are looking for a cycle that passes through each vertex of G exactly once.

Model 1

Decision variables:
x_{ij} for any $(i, j) \in E$
$x_{ij} = 1$ if the arc (i, j) belongs to the cycle
$x_{ij} = 0$ otherwise

Model: $\min \sum_{(i,j)\in E} c_{ij} x_{ij}$

$$\sum_{(i,j)\in E} x_{ij} = 1 \quad \text{for any } i \in V \qquad (1)$$

$$\sum_{(j,i)\in E} x_{ji} = 1 \quad \text{for any } i \in V \qquad (2)$$

$$\sum_{(i,j)\in E: i\in S, j\notin S} x_{ij} \geq 1 \quad \text{for any } S \subseteq V, \text{ with } S \neq \emptyset, V \qquad (*)$$

$$x_{ij} \in \{0, 1\} \quad \text{for any } (i, j) \in E$$

Correctness: the correctness of the model is quite evident; indeed, constraint groups (1) and (2) ensure that, for every vertex, there is exactly one incoming arc and exactly one outgoing arc (with respect to the cycle being sought), while constraint group (*) ensures that the resulting solution is a single cycle rather than a collection of multiple disjoint cycles.

Note: the group (*) of constraints consists of an exponential number of constraints, specifically, $2^{n-1} - 2$ constraints; this poses a drawback if one were to implement Model 1 using a solver. Model 2, introduced below, eliminates this drawback.

Model 2

Decision variables:
x_{ij} for any $(i, j) \in E$
$x_{ij} = 1$ if the arc (i, j) belongs to the cycle
$x_{ij} = 0$ otherwise
u_i for any $i \in V$
$u_i = k$ if i is the k-th vertex visited in the cycle

Model: min $\sum_{(i,j)\in E} c_{ij} x_{ij}$

$\sum_{(i,j)\in E} x_{ij} = 1$ for any $i \in V$ (1)

$\sum_{(j,i)\in E} x_{ji} = 1$ for any $i \in V$ (2)

$n\, x_{ij} + u_i - u_j \leq n - 1$ for any $(i, j) \in E$, with $\boldsymbol{j \neq 1}$ (**)

$\boldsymbol{u_1 = 1}$

$2 \leq u_i \leq n$ for any $i \in V$, with $\boldsymbol{i \neq 1}$

$x_{ij} \in \{0, 1\}$ for any $(i, j) \in E$

Correctness: the correctness of the model is not immediately obvious (we shall see it below); constraint groups (1) and (2) ensure that, for every vertex, there is exactly one incoming arc and exactly one outgoing arc (with respect to the cycle being sought); the problem remains of ensuring that the solution found is not a collection of multiple disjoint cycles but rather a single cycle; we shall see below that constraint group (**) resolves this issue:
:: as a preliminary step, we observe that:
if $x_{ij} = 0$, then the corresponding constraint in (**) becomes $u_i - u_j \leq n - 1$, i.e., it becomes a constraint that is always satisfied [considering the other constraints on the variables u_i];
if $x_{ij} = 1$, then the corresponding constraint in (**) becomes $u_i - u_j \leq -1$, i.e., $u_j \geq u_i + 1$;
:: on the one hand, if the solution is a single cycle formed by nodes 1, 2, …, n in that sequence (without loss of generality), that is, defined by $x_{12} = 1$, $x_{23} = 1$, …, $x_{n1} = 1$, then the values $u_1 = 1$, $u_2 = 2$, …, $u_n = n$ guarantee that the solution is feasible [indeed, for each such variable, the corresponding constraint in (**) is satisfied];
:: on the other hand, if the solution is not a single cycle [i.e., it is a collection of disjoint cycles], then there exists a cycle C within the solution that does not contain vertex 1; that is, cycle C is defined by the variables $x_{ij} = 1$ for every arc (i, j) in C; consequently, the solution is not feasible [indeed, for each such variable, one obtains the corresponding inequalities $u_j \geq u_i + 1$, which, when taken as a system, yield an incompatibility, i.e., a contradiction, given the constraints on the variables u_i and the fact that vertex 1 does not belong to cycle C].

Note: the group (**) of constraints consists of a polynomial number of constraints, specifically, at most 2^n; this resolves the drawback observed with Model 1 and makes Model 2 the preferred choice for implementing a TSP model on a solver—despite the fact that it introduces n additional variables compared to Model 1 (to visualize this, one could, for instance, set $n = 10$ and simulate, in the abstract, an implementation of both models).

Observation – Relaxation:

We observe that a *relaxation* of Model 1 and Model 2 is given by the following problem:

Modello: min $\sum_{(i,j)\in E} c_{ij} x_{ij}$

$\sum_{(i,j)\in E} x_{ij} = 1$ for any $i \in V$ (1)

$\sum_{(j,i)\in E} x_{ji} = 1$ for any $i \in V$ (2)

$x_{ij} \in \{0, 1\}$ for any $(i, j) \in E$

which corresponds to an Assignment problem, that is, an *easy* problem [see Note 2].
The optimal solution to this *relaxation*, let us call it *S*, provides a *lower bound* on the optimal solution [with respect to the cost of *S*]; furthermore, if *S* consists of a single cycle, then *S* is also the optimal solution to the TSP; if S does not consist of a single cycle, it can serve as a starting point for finding a (good) feasible TSP solution within a heuristic method. □

Heuristic methods for the TSP

Below, we examine three heuristic methods for the TSP. These methods are progressively more elaborate. Specifically, for each of these methods, we assume the input is a TSP instance consistent with the alternative definition, that is, we assume the following for each method:

Input: a complete directed graph $G = (V, E)$, with $V = \{v_1, \ldots, v_n\}$, and with $c_{ij} \geq 0$ for any $(i, j) \in E$.
Output: a cycle in *G* that visits all vertices of *G*.
Thus, based on the input definition, we seek a cycle that visits each vertex of *G* exactly once.

• Nearest-neighbor method

One starts at an arbitrary vertex and marks it (meaning the fact that the vertex has been visited is recorded); at each iteration, one proceeds to the nearest vertex [specifically, a nearest vertex, as there may be multiple vertices at the same distance] among the unmarked ones; finally, the process stops when all vertices have been marked.

begin
--- choose any vertex of G, say a, and set $\pi[1] := a$.
--- **for** $i = 1, \ldots, n-1$ **do**
--- --- choose a vertex $v \in V \setminus \{\pi[1], \ldots, \pi[i]\}$ such that $c_{\pi[i]v}$ is minimum and set $\pi[i+1] := v$
end

• Gradual insertion method

One begins with an arbitrary cycle in G consisting of three vertices of G; then, the following procedure is iterated: insert a new vertex into the cycle (for example, a vertex that minimizes the insertion cost, as we shall see) to obtain a new cycle containing one additional vertex from G; finally, the process stops when all vertices have been inserted.

begin
--- let C be any cycle of G that passes through three vertices $\pi[1]$, $\pi[2]$, $\pi[3]$ of G;
--- **for** $i = 3, \ldots, n-1$ **do**
--- **begin**
--- --- sia $(c_{\pi[\boldsymbol{j}]\boldsymbol{w}} + c_{\boldsymbol{w}\pi[\boldsymbol{j+1}]}) - c_{\pi[\boldsymbol{j}]\pi[\boldsymbol{j+1}]} =$
$= \mathit{argmin} \{ (c_{\pi[j]v} + c_{v\pi[j+1]}) - c_{\pi[j]\pi[j+1]} : v \in V \setminus \{\pi[1], \ldots, \pi[i]\} , j \in \{1, \ldots, i\} \text{ (mod } i)\}$
--- --- set $\pi[\boldsymbol{j+1}] := \boldsymbol{w}$
(i.e., insert vertex $\boldsymbol{w}$ into the cycle as the $(\boldsymbol{j}+\mathbf{1})$-th vertex)
--- --- set $\pi[j+2] := \pi[j+1]$, $\pi[j+3] := \pi[j+2]$, … , $\pi[i+1] := \pi[i]$
(i.e., redefine the new cycle with $i+1$ vertices)
--- **end**
end

We note that this method can nonetheless be "customized", for example, by fixing a specific choice of the initial cycle or by setting an alternative insertion criterion.

• 2-Opt Method

Let $G = (V, A)$ be a directed graph; assume that, for every arc $(i, j) \in A$, one has $c(i, j) = c(j, i)$. Initially, consider any cycle H of G, with arcs $E(H) = \{(u_1, u_2), \ldots, (u_{n-1}, u_n), (u_n, u_1)\}$, that visits all vertices of G; then, iterate the following procedure (see Figure 4):

> if there exist $(u_h, u_{h+1}), (u_i, u_{i+1}) \in H$, with no shared endpoints, such that $c(u_h, u_{h+1}) + c(u_i, u_{i+1}) > c(u_h, u_i) + c(u_{h+1}, u_{i+1})$, then *replace* the edges (u_h, u_{h+1}) and (u_i, u_{i+1}) in H with the edges (u_h, u_i) and (u_{h+1}, u_{i+1}), and then *reverse* the direction of the edges along the path $\{u_{h+1}, \ldots, u_i\}$ in H [note that, ultimately, H remains a cycle passing through all vertices of G but with a lower cost],

as long as such a procedure is possible [i.e., as long as they exist].

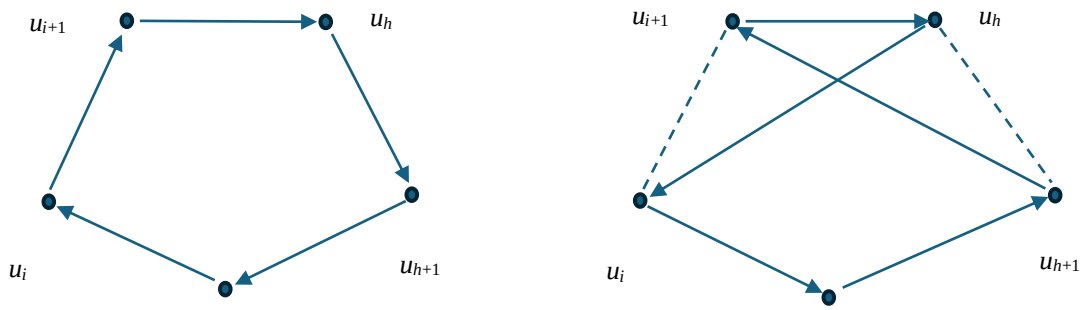


Figura 4

Observation – Relaxation:

Consistent with the previous observation regarding the relaxation of Model 1 and Model 2, we present below a heuristic method known as the "Patch Algorithm," without going into detail (merely to provide an idea):

Step 1. Calculate the TSP relaxation as indicated in the previous remark; let $F = \{C_1, \ldots, C_p\}$ be the family of directed cycles thus obtained; if F contains a single cycle, then STOP [this cycle is an optimal solution for the TSP].

Step 2. For each pair of cycles C_i , $C_j \in F$, calculate the *cost increment* γ_{ij} corresponding to *merging* C_i and C_j in the most cost-effective way possible.

Step 3. *Merge* the two cycles C_i and C_j corresponding to the minimum value of γ_{ij}. Update F.

Step 4. If F contains a single cycle, then STOP [this cycle is a heuristic solution for the TSP]. Otherwise, return to Step 2. □

E. Vehicle Routing Problem (VRP)

These notes are freely adapted from the book by Prof. Antonio Sassano (see reference [4] in the syllabus) and from the resource http://groups.di.unipi.it/optimize/Courses/RO2IG/aa1415/8-tsp_vrp.pdf by Prof. Laura Galli (consulting these references is recommended).

Let $G = (V, E)$ be a complete directed graph, where $V = \{v_1, \ldots, v_n\}$, and let $c_{ij} \geq 0$ be a cost assignment for each arc (i, j) of G [cf. the alternative definition of the TSP, i.e., c_{ij} represents the cost of the shortest directed path from vertex i to vertex j].

Vertex v_1 represents a *depot* (warehouse or starting point) from which shipments of a specific good originate, while each vertex in the set $V \setminus \{v_1\}$ represents a *point of sale*.
For each point of sale $v_i \in V \setminus \{v_1\}$, the *demand* d_i for a certain quantity of the good is known. A fleet of *m vehicles* is available to carry out deliveries; each vehicle has the (same) *maximum capacity Q*, expressed in the same unit of measurement as the demand.

Basic definition of the VRP: The problem involves assigning each sales outlet to a single vehicle, subject to the constraint that the total demand of the outlets assigned to each vehicle does not exceed the maximum capacity *Q*, with the aim of minimizing the total distance traveled by the vehicles to complete all deliveries to their assigned outlets (starting from and returning to the depot). □

This problem is NP-hard, that is, *hard* (see Note 2).

A *feasible solution* for the VRP is identified by a partition $\{V_1, \ldots, V_m\}$ of $V \setminus \{v_1\}$ [i.e., the set of customer locations] such that, for each $h \in \{1, \ldots, m\}$ [i.e., for each vehicle h], the sum of the demands of the locations in V_h does not exceed the maximum capacity Q; it is represented by a family of m directed cycles $\{C_1, \ldots, C_m\}$ such that, for each $h \in \{1, \ldots, m\}$, the cycle C_h passes through all vertices in V_h and through vertex v_1 [i.e., the depot].

ILP Model for the VRP

Model

Let $V = \{v_1, \ldots, v_n\} = \{1, \ldots, n\}$.

Decision variables:
x_{ij} for any $(i, j) \in E$
$x_{ij} = 1$ if the arc (i, j) belongs to the family of m cycles
$x_{ij} = 0$ otherwise
u_i for any $i \in V$
u_i = loading the vehicle before visiting the vertex i

Model: $\min \sum_{(i,j)\in E} c_{ij} x_{ij}$

$$\sum_{(i,j)\in E} x_{ij} = 1 \qquad \text{for any } i \in V \setminus \{1\} \tag{1}$$

$$\sum_{(j,i)\in E} x_{ji} = 1 \qquad \text{for any } i \in V \setminus \{1\} \tag{2}$$

$$\sum_{(1,j)\in E} x_{1j} = m \tag{3}$$

$$u_j - u_i + Q\,x_{ij} \leq Q - d_i \qquad \text{for any } (i,j) \in E, \text{ with } j \neq 1 \tag{***}$$

$$d_i \leq u_i \leq Q \qquad \text{for any } i \in V, \text{ with } i \neq 1$$

$$x_{ij} \in \{0, 1\} \qquad \text{for any } (i,j) \in E$$

Correctness: the validity of the model is not immediately apparent (we shall examine it below); constraint groups (1) and (2) ensure that, for every vertex other than the depot, there is—with respect to the cycles being sought—exactly one incoming arc and exactly one outgoing arc; meanwhile, constraint (3) ensures that exactly *m* vehicles will depart from and return to the depot (based on the preceding constraint groups); the problem remains of ensuring that the resulting solution is feasible; this can be verified by applying the same considerations used for constraint group (**) to constraint group (***) and taking the remaining constraints into account.

Note: the group (***) of constraints consists of a polynomial number of constraints, specifically, at most $|V|^2$ constraints; this, just as in the case of the group of constraints (**), ensures that the total number of cons

Variants: Naturally, there can be many variants of the (basic) VRP; we list only a few below to reflect real-world scenarios. However, such variants can generally be incorporated into the aforementioned ILP model for the (basic) VRP:
:: *number of vehicles*: this can be fixed a priori or serve as a decision variable;
:: *vehicle type*: vehicles may or may not all have the same maximum capacity;
:: *number of depots*: there may be more than one;
:: *time constraints*: there may be time constraints for making deliveries;
:: *objective functions*: various objectives may be pursued, often conflicting and difficult to represent via a single objective function, though the literature proposes functions that account for factors such as distance traveled, the number of vehicles used, total service cost, etc.

Heuristic methods for the VRP

There are many heuristic methods for the VRP. However, we do not detail them here – except in general terms, as shown below – but instead refer the reader to heuristic methods for the TSP to gain an idea of the types of heuristics available.

The heuristic methods for the VRP proposed in the literature can be grouped into three categories.

Heuristic methods for the VRP proposed in the literature can be grouped into three categories.

:: constructive methods: a feasible solution is built incrementally; vertices not yet assigned to a vehicle are inserted into one of the sets according to a "greedy" criterion.

:: Two-phase methods: the first phase establishes the partition $\{V_1, \ldots, V_m\}$, while the second phase determines the routes $\{C_1, \ldots, C_m\}$.

:: Improvement methods or local search methods: a feasible solution is found first, and an attempt is then made to improve it by exploring a neighborhood until a local optimum is reached.