Chapter 2
Graphs and Optimization

2.1 Graph Theory

The matching problem is an important combinatorial problem defined on a *graph*. Graphs provide very often a pictorial representation of the mathematical structure underlying combinatorial optimization problems. On the other hand, graph theory is by itself rich of elegant results that can give us useful insights on many combinatorial and physical problems. For these reasons, we present here a very short introduction to the basic definitions and results of graph theory. We will refer mostly to the standard textbook of Diestel [3].

2.1.1 Fundamental Definitions

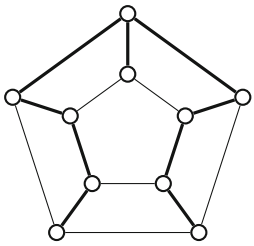
A graph $\mathtt{G} = \mathrm{Graph}(\mathcal{V}; \mathcal{E})$ is a pair of sets $(\mathcal{V}, \mathcal{E})$ such that $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The elements of $\mathcal{V}$ are called *vertices* (or *nodes*), whilst the elements of $\mathcal{E}$ are usually called *edges*. We will denote by[1] $V = |\mathcal{V}|$ the number of elements of $\mathcal{V}$ (V is sometimes called *order* of $\mathtt{G}$) and we will suppose always that $V \in \mathbb{N}$ is finite, unless otherwise specified. Moreover, given a vertex v and an edge e, we say that v is *incident* with e if $v \in e$. In this case e is an edge *at* v and we will write $e \to v$. We will call $\mathcal{E}(v)$ the set of edges at v and $|\mathcal{E}(v)|$ the *degree* of v. We say that $u, v \in \mathcal{V}$ are *adjacent* if $(u, v) \in \mathcal{E}$; we will denote by ∂v the set of adjacent vertices to v. We define the *complete graph* $\mathtt{K}_V$ as the graph with V vertices in which each vertex is adjacent to all the others.

Two graphs $\mathtt{G} = \mathrm{Graph}(\mathcal{V}; \mathcal{E})$ and $\mathtt{G}' = \mathrm{Graph}(\mathcal{V}'; \mathcal{E}')$ are *isomorphic*, or $\mathtt{G} \sim \mathtt{G}'$, if a bijection $\varphi \colon \mathcal{V} \to \mathcal{V}'$ exists such that $(u, v) \in \mathcal{E} \Leftrightarrow (\varphi(u), \varphi(v)) \in \mathcal{E}'$. Finally, given two graphs $\mathtt{G} = \mathrm{Graph}(\mathcal{V}; \mathcal{E})$ and $\mathtt{G}' = \mathrm{Graph}(\mathcal{V}'; \mathcal{E}')$, if $\mathcal{V} \subseteq \mathcal{V}'$ and $\mathcal{E} \subseteq \mathcal{E}'$,

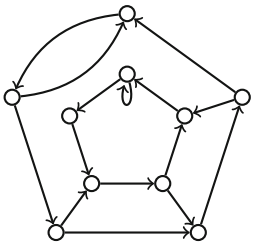
[1] In the present work, given a set A of $N \in \mathbb{N}$ elements, we will use the notation $|A| = N$ for the cardinality of the set.

© Springer International Publishing AG 2017

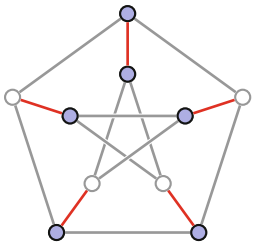
G. Sicuro, *The Euclidean Matching Problem*, Springer Theses,
DOI 10.1007/978-3-319-46577-7_2



(a) Example, in bold, of spanning subgraph (in this case, a tree) of a given graph.



(b) Directed graph with a loop.



(c) The Petersen's graph with a vertex cover (in blue) and an edge cover (in red).

Fig. 2.1 Three examples of graphs

than we say that G is a *subgraph* of G' and G' is a *supergraph* of G: in symbols, $\mathsf{G} \subseteq \mathsf{G}'$. We say that G is a *spanning subgraph* of G' if and only if $\mathcal{V} = \mathcal{V}'$, see Fig. 2.1a. A *directed graph* (or *digraph*) is a graph in which we assign an *initial vertex* and a *terminal vertex* to each edge in the edge set, see Fig. 2.1b. In digraphs edges are *ordered* couples of vertices. In particular, if the vertex u and the vertex v are respectively the initial vertex and the terminal vertex of the edge (u, v), we write

$$\vec{e} = \overrightarrow{(u, v)}.$$

In a digraph, an edge in which the initial vertex and terminal vertex coincide is called *loop*.

A *vertex cover* of G is a subset of $\mathcal{V}$ such that any edge of G has at least one endpoint in it. The *vertex covering number* $c_V(\mathsf{G})$ is the smallest possible size of a vertex cover of G. Similarly, an *edge cover* of G is a subset of $\mathcal{E}$ such that any vertex of G is the endpoint of at least one element in it. The *edge covering number* $c_E(\mathsf{G})$ is the smallest possible size of an edge cover of G (see Fig. 2.1c).

Given two graphs $\mathsf{G} = \text{Graph}(\mathcal{V}; \mathcal{E})$ and $\mathsf{G}' = \text{Graph}(\mathcal{V}; \mathcal{E}')$ with the same vertex set $\mathcal{V}$, we define

$$\mathsf{G} \triangle \mathsf{G}' := \text{Graph}(\mathcal{V}_{\mathsf{G}\triangle\mathsf{G}'}; \mathcal{E} \triangle \mathcal{E}'), \tag{2.1.1}$$

where

$$\mathcal{E} \triangle \mathcal{E}' := \left(\mathcal{E} \cup \mathcal{E}'\right) \setminus \left(\mathcal{E} \cap \mathcal{E}'\right)$$

is the symmetric difference between the two edge sets and $\mathcal{V}_{\mathsf{G}\triangle\mathsf{G}'}$ is the set of the vertices that are ends for the edges in $\mathcal{E} \triangle \mathcal{E}'$.

A certain graph $\mathsf{G} = \text{Graph}(\mathcal{V}; \mathcal{E})$ can be represented also by a $V \times E$ matrix, called *incidence matrix*, $\mathsf{B} := (b_{ij})_{ij}$ in such a way that

$$b_{ij} = \begin{cases} 1 & \text{if } v_i \in e_j \\ 0 & \text{otherwise.} \end{cases} \tag{2.1.2}$$

Similarly, we introduce the more commonly used $V \times V$ *adjacency matrix* $\mathsf{A} := (a_{ij})_{ij}$, such that

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{2.1.3}$$

An undirected graph has a symmetric adjacency matrix, and, therefore, A has a real spectrum, called *spectrum of* $\mathtt{G}$. A *weighted* graph is a graph in which we associate a certain function $w\colon \mathcal{E} \to \mathbb{R}$ to the graph itself. Given an edge $e \in \mathcal{E}$, we say that $w(e)$ is the *weight* of e. For a given weighted graph $\mathtt{G}$, we can introduce the *weighted adjacency matrix* as $\mathsf{W} := (w(e_{ij})a_{ij})_{ij}$.

2.1.2 Paths, Forests and Multipartite Graphs

A *path* $\mathtt{P} = \mathrm{Graph}(\mathcal{V}_\mathtt{P}; \mathcal{E}_\mathtt{P})$ in a graph $\mathtt{G}$ is a particular subgraph $\mathtt{P} \subseteq \mathtt{G}$ such that $\mathcal{V}_\mathtt{P} = \{v_0, \ldots, v_k\}$ is a set of k distinct vertices and its edge set is given by

$$\mathcal{E}_\mathtt{P} = \{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)\}.$$

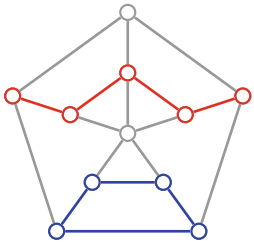
We say that the path links v_0 and v_k and has length k. A graph $\mathtt{G}$ is *connected* if, for any couple of vertices $v, u \in \mathcal{V}$, there exists a path in $\mathtt{G}$ linking them. The length $\delta(v, u)$ of the shortest path linking two vertices u and v of $\mathtt{G}$ is called *distance* of u and v on $\mathtt{G}$. The diameter of a graph $\mathrm{diam}(\mathtt{G})$ is given by

$$\mathrm{diam}(\mathtt{G}) := \max_{u,v} \delta(u, v). \tag{2.1.4}$$

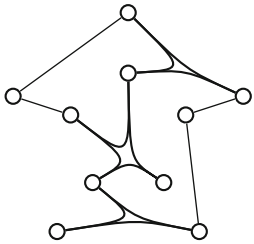
If the graph $\mathtt{G}$ is not connected then, by definition, $\mathrm{diam}(\mathtt{G}) = +\infty$. Any graph can be expressed as union of maximal connected subgraphs, called *components*, and a connected graph is a graph with only one component. Given a connected graph $\mathtt{G} = \mathrm{Graph}(\mathcal{V}; \mathcal{E})$, the subset $\mathcal{X} \subset \mathcal{V} \cup \mathcal{E}$ is said to be a *separating set* if $\mathtt{G}' = \mathrm{Graph}(\mathcal{V} \setminus \mathcal{X}; \mathcal{E} \setminus \mathcal{X})$ is not connected. If $\mathcal{X}$ contains only a single vertex, the vertex is said a *cutvertex*. Similarly if $\mathcal{X}$ contains one edge, we say that the edge is a *bridge*. A graph is called *separable* if it is not connected or has a cutvertex. In particular, we will call *biconnected* a connected graph with no cutvertices.

For $k \geq 3$, a *cycle* $\mathtt{C} = \mathrm{Graph}(\mathcal{V}_\mathtt{C}; \mathcal{E}_\mathtt{C})$ in a graph $\mathtt{G}$ is a subgraph $\mathtt{C} \subseteq \mathtt{G}$ such that $\mathcal{V}_\mathtt{C} = \{v_0, \ldots, v_{k-1}\}$ is a set of k distinct vertices and

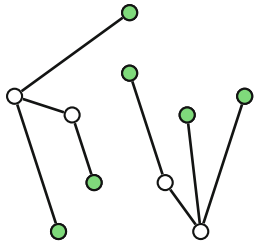
$$\mathcal{E}_\mathtt{C} = \{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-2}, v_{k-1}), (v_{k-1}, v_0)\}, \tag{2.1.5}$$



(a) Path (in red) and cycle (in blue) in a graph.



(b) Example of hypergraph with two 3-hyperedges and one 4-hyperedge.



(c) Forest composed by two trees. Leaves are colored in green.

Fig. 2.2 Examples of paths, cycles, separable graphs, trees and forests

see Fig. 2.2a. We say that such a cycle has length k. The minimum length of a cycle contained in a certain graph $\mathtt{G}$ is called *girth* of $\mathtt{G}$, whilst the maximum length is called *circumference* of $\mathtt{G}$.

A *Hamiltonian path* in a graph is a path traversing all vertices of the graph exactly once. A Hamiltonian path that is a cycle is called *Hamiltonian cycle*. Similarly, an *Eulerian path* in a graph is a path traversing all edges of the graph exactly once, whereas an Eulerian path that is a cycle is called *Eulerian cycle*. We call *Eulerian graph* a graph containing an Eulerian cycle.

Clearly, not all graphs contain cycles: an acyclic graph is called *forest*, and a connected forest is called *tree*. A non-trivial forest (i.e., a forest with $|\mathcal{E}| \neq 0$) has always 1-degree vertices, called *leaves*, see Fig. 2.2c. Given a tree, sometimes a specific vertex is considered special and called *root* of the tree. With reference to the root, we define the *height* of a vertex as the distance of the vertex itself from the root.

A graph $\mathtt{G} = \mathrm{Graph}(\mathcal{V}; \mathcal{E})$ is called q-*partite* (or, less precisely, multipartite) if we can partition $\mathcal{V}$ into q subsets (or *classes*),

$$\mathcal{V} = \bigcup_{i=1}^{q} \mathcal{V}_i, \quad \mathcal{V}_i \cap \mathcal{V}_j = \emptyset \text{ for } i \neq j,$$

in such a way that every edge in $\mathcal{E}$ connects vertices in different classes. We will denote such a graph as $\mathtt{G} = \mathrm{Graph}(\mathcal{V}_1, \ldots, \mathcal{V}_q; \mathcal{E})$. A q-partite graph is called *complete* if, given two vertices in two different classes, there exists an edge connecting them. We will denote the complete q-partite graph by $\mathtt{K}_{V_1,\ldots,V_q}$, $V_i := |\mathcal{V}_i|$. If $q = 2$ a multipartite graph is called *bipartite*. Bipartite graphs have the characterizing property of having no odd cycles.

2.1.3 Euler's Formula and Planar Graphs

Let us consider now the set $\mathcal{S}_{\mathtt{G}}$ of the *spanning* subgraphs of a given graph $\mathtt{G} = \mathrm{Graph}(\mathcal{V}; \mathcal{E})$. The set $\mathcal{S}_{\mathtt{G}}$ contains the set $\mathcal{S}^{\mathrm{E}}_{\mathtt{G}}$ of the *Eulerian subgraphs*. This space has the peculiar property of being closed under the symmetric difference operation $\triangle$. In other words, if $\mathtt{G}_1 \in \mathcal{S}^{\mathrm{E}}_{\mathtt{G}}$ and $\mathtt{G}_2 \in \mathcal{S}^{\mathrm{E}}_{\mathtt{G}}$, then $\mathtt{G}_1 \triangle \mathtt{G}_2 \in \mathcal{S}^{\mathrm{E}}_{\mathtt{G}}$. The dimension of $\mathcal{S}^{\mathrm{E}}_{\mathtt{G}}$ respect to the operation $\triangle$ is called *cyclomatic number* L of the graph $\mathtt{G}$. Indeed, L is the number of cycles in $\mathtt{G}$ that cannot be obtained by other subgraphs through symmetric difference. These cycles are called *independent cycles* and play the role of a "basis" in the space of Eulerian subgraphs. Let us call $\mathcal{L}(\mathtt{G})$ the set of independent cycles. If a graph $\mathtt{G}$ has κ components, then the following general Euler's formula can be proved:

$$V + L = E + \kappa. \tag{2.1.6}$$

The relation above is particularly easy to apply for *planar graphs*. Planar graphs are graph that can be embedded in (i.e., drawn on) the surface of a sphere in such a way that no edge crossing appears. If such embedding is considered, we can immediately recognize F cycles on the sphere, called *faces*. Each face is characterized by a simple property, i.e., it divides the sphere into two regions in such a way that in one of them there is no paths having both endpoints on the face itself. The cyclomatic number is then recovered as $L = F - 1$.

2.1.4 Hypergraphs

In an *hypergraph* $\mathtt{G} = \mathrm{HyperGr}(\mathcal{V}; \mathcal{E})$ with vertex set $\mathcal{V}$, an edge $e \in \mathcal{E}$ can connect more than two vertices, see Fig. 2.2b. If the edge e connects k vertices, we say that e is a k-hyperedge and we write $|e| = k$. We say that a *walk* on the hypergraph is an ordered sequence $(v_0, e_1, v_1, \ldots, e_p, v_p)$, $\{v_i\}_{i=0,\ldots,p} \in \mathcal{V}$, $\{e_i\}_{i=1,\ldots,p} \in \mathcal{E}$. A *path* is a walk in which all vertices and all edges are distinct respectively. A *cycle* is a path having $p \geq 2$ and $v_0 = v_p$. An hypergraph is *connected* if, given two distinct vertices $v, u \in \mathcal{V}$, there exists a walk connecting them. Remarkably, the following generalization of the Euler's formula holds for an hypergraph with κ connected components [1]

$$\sum_{e \in \mathcal{E}} |e| - E - V + \kappa \geq 0, \tag{2.1.7}$$

where, as usual, $E = |\mathcal{E}|$ and $V = |\mathcal{V}|$.

2.1.5 Matchings on Graphs

We can finally introduce the main concept of this dissertation. Given a graph $\mathtt{G} = \mathrm{Graph}(\mathcal{V}; \mathcal{E})$, we say that $\mathcal{E}_\mathtt{M} \subseteq \mathcal{E}$ is a *matching* of *size* $|\mathcal{E}_\mathtt{M}|$ in $\mathtt{G}$ if, given two edges in $\mathcal{E}_\mathtt{M}$, they have no vertex in common, see Fig. 2.3. The size of the largest matching (*maximum matching*) in $\mathtt{G}$, $m(\mathtt{G})$, is called *matching number* of $\mathtt{G}$. Defining now $\mathcal{V}_\mathtt{M} := \{v \in \mathcal{V} : \exists e \in \mathcal{E}_\mathtt{M} \text{ such that } e \to v\}$, if $\mathcal{V}_\mathtt{M} = \mathcal{V}$ we say that the matching is *perfect*. In the following, we will use the notation $\mathtt{M} = \mathrm{Graph}(\mathcal{V}_\mathtt{M}; \mathcal{E}_\mathtt{M})$ to denote the sugraph of $\mathtt{G}$ induced by the matching set, and we will call it matching as well. The following fundamental result, proved in 1931 by Dénes Kőnig, relates the matching number with the vertex covering number in a bipartite graph.

Theorem 2.1.1 (Kőnig's minimax theorem) *Let* $\mathtt{G}$ *be a bipartite graph. Then*

$$c_V(\mathtt{G}) = m(\mathtt{G}). \tag{2.1.8}$$

In 1935, Philip Hall proposed an equivalent formulation of the theorem above.

Theorem 2.1.2 (Hall's theorem) *A bipartite graph* $\mathtt{G}$ *contains a matching if and only if*

$$\left| \bigcup_{x \in \mathcal{X}} \partial x \right| \geq |\mathcal{X}|, \quad \forall \mathcal{X} \subseteq \mathcal{V}_1. \tag{2.1.9}$$

We will denote by $\partial_\mathtt{G} \mathcal{X} := \bigcup_{x \in \mathcal{X}} \partial x$, where the subscript $\mathtt{G}$ stress the fact that we refer to the topology of $\mathtt{G}$.

Corollary 2.1.3 (Marriage theorem) *Let* $\mathtt{G} = \mathrm{Graph}(\mathcal{V}_1, \mathcal{V}_2; \mathcal{E})$ *be a bipartite graph. Then* $\mathtt{G}$ *admits a perfect matching if and only if the property in Eq.* (2.1.9) *holds and moreover*

$$|\mathcal{V}_1| = |\mathcal{V}_2|. \tag{2.1.10}$$

If $|\mathcal{V}_1| = |\mathcal{V}_2| = V$, numbering separately the vertices of each class in the graph, a perfect matching can be expressed as a permutation of V elements. For a proof of

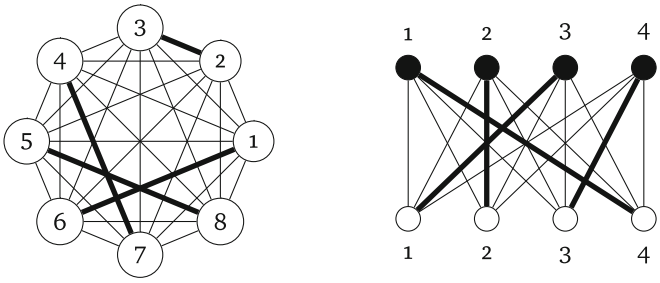


Fig. 2.3 *On the left*, complete graph $\mathtt{K}_8$ and an example of perfect matching on it. *On the right*, complete bipartite graph $\mathtt{K}_{4,4}$ and an example of perfect bipartite matching on it

the previous statements, see [9]. Finally, given a matching $\mathtt{M}$ on $\mathtt{G}$ and a path $\mathtt{P} \subset \mathtt{G}$, we say that the path $\mathtt{P}$ is $\mathtt{M}$-*alternating* if the edges of $\mathtt{P}$ are alternately in and not in $\mathtt{M}$.

2.2 Optimization Problems

In the previous Section we have discussed some fundamental definitions about graphs. Graphs provide very often a mathematical representation of combinatorial problems and, in particular, many combinatorial problems can be described in terms of concepts introduced in graph theory. To proceed further, let us first introduce the definition of *optimization problem* [2]. An *instance of an optimization problem* is a couple of mathematical elements, i.e., a *space of feasible solutions* $\mathcal{F} \neq \emptyset$ and a *cost function*

$$\mathcal{C}\colon \mathcal{F} \to \mathbb{R}. \tag{2.2.1}$$

The target is to find the *globally optimal solution*, i.e. an element $x_o \in \mathcal{F}$ such that

$$\mathcal{C}[x_o] = \min_{x \in \mathcal{F}} \mathcal{C}[x]. \tag{2.2.2}$$

The set of all instances of a certain optimization problem is, by definition, the optimization problem itself. Observe that the cardinality of $\mathcal{F}$ was not specified and the existence of x_o is, *a priori*, not guaranteed.

Linear Optimization Problems To exemplify the given definitions, let us briefly discuss here the important class of linear optimization problems [2]. A *linear optimization problem* can be stated as follows: let us consider an $n \times m$ matrix of integers $\mathsf{A} = (a_{ij})_{ij}$, a vector of m integers $\mathbf{b} \in \mathbb{Z}^m$ and a vector of n integers $\mathbf{c} \in \mathbb{Z}^n$. We want to find a vector $\mathbf{X} = (X_i)_{i=1,\ldots,n} \in (\mathbb{R}^+)^n$ such that

$$z := \mathbf{c}^\mathsf{T} \cdot \mathbf{X} = \min_{\mathbf{x} \in \mathcal{F}} \mathbf{c}^\mathsf{T} \cdot \mathbf{x} \tag{2.2.3}$$

on a certain non-empty space $\mathcal{F}$ of feasible solutions, defined as follows

$$\mathcal{F} := \left\{ \mathbf{x} = (x_1, \ldots, x_n) \in (\mathbb{R}^+)^n \colon \mathsf{A} \cdot \mathbf{x} = \mathbf{b} \right\} \neq \emptyset. \tag{2.2.4}$$

The linear optimization problem, when stated in the previous form, is said to be in a *standard form*. To have a solution of our optimization problem it is necessary that $n - \text{ rank } \left[\mathsf{A}\right] > 0$ (otherwise no point satisfies all the constraints). We require therefore for simplicity that $\text{ rank } \left[\mathsf{A}\right] = m < n$. In the space $\mathbb{R}^{n-m}$, the constraint condition $\mathsf{A} \cdot \mathbf{x} = \mathbf{b}$, with the additional constraints $x_i \geq 0 \; \forall i = 1, \ldots, n$, delimits an $(n - m)$-dimensional convex polytope (see Fig. 2.4).

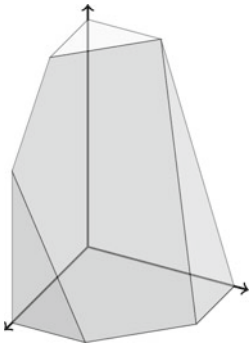


Fig. 2.4 An example of 3-dimensional convex polytope

2.2.1 Combinatorial Optimization

Combinatorial optimization deals with optimization problems in which the cardinality of $\mathcal{F}$ is *finite* for all instances, $|\mathcal{F}| \in \mathbb{N}$. In this case the problem has *always* at least one solution. However, in many cases the number of feasible solutions is extremely large, and a brute-force approach is computationally impracticable. To better exemplify these aspects, let us discuss some relevant (classical) combinatorial optimization problems in more details.

THE TRAVELLING SALESMAN PROBLEM In the *Travelling Salesman Problem* (TSP) a complete graph $\mathtt{K}_N = \text{Graph}(\mathcal{V}; \mathcal{E})$ is given with a weight function $w\colon \mathcal{E} \to \mathbb{R}^+$, in such a way that a weight $w(e) \in \mathbb{R}^+$ is associated to each edge e of the graph. The space $\mathcal{F}$ is given by all possible closed paths $\mathtt{h} = \text{Graph}(\mathcal{V}_\mathtt{h}; \mathcal{E}_\mathtt{h})$ passing only once through each vertex. In other words, the space of feasible solutions in the TSP is the set of Hamiltonian cycles on $\mathtt{K}_N$. The cost function is given by

$$\mathcal{C}^{\text{TSP}}[\mathtt{h}] := \sum_{e \in \mathcal{E}_\mathtt{h}} w(e). \tag{2.2.5}$$

A similar formulation of the problem can be given on a generic (connected) graph $\mathtt{G}$. Observe that, working on $\mathtt{K}_N$, $|\mathcal{F}| = \frac{(N-1)!}{2}$. Therefore the direct inspection of the solution by computing all possible values of the cost function requires a huge amount of steps even for relatively small values of N. In a variation of this problem, the *Chinese Postman Problem*, the set $\mathcal{F}$ of feasible solutions is given by the set of all Eulerian cycles of the considered graph.

THE GRAPH q-COLORING PROBLEM The *graph q-coloring problem* (q-COL) is a problem defined on a graph $\mathtt{G} = \text{Graph}(\mathcal{V}; \mathcal{E})$, $V := |\mathcal{V}|$. We want to assign to each vertex $v \in \mathcal{V}$ a number (a "color") $q_v \in \{1, \ldots, q\}$, $q \in \mathbb{N}$, in such a way that the cost function

$$\mathcal{C}^{q-\text{COL}}[\mathbf{q}] := \sum_{(v,u) \in \mathcal{E}} \delta_{q_v, q_u} \tag{2.2.6}$$

is minimized. The set $\mathcal{F}$ is therefore given by all sets $\mathbf{q} = \{q_v\}_{v\in\mathcal{V}}$ such that $q_v \in \{1, \dots, q\}$. The number of feasible solutions is therefore q^V.

THE k-SAT PROBLEM The k-SAT problem is defined on an hypergraph $\mathtt{G} = \text{HyperGr}(\mathcal{V}; \mathcal{E})$, $|\mathcal{V}| = V$, such that $|e| = k\ \forall e \in \mathcal{E}$. We assign a quantity $J_e^v \in \{-1, 1\}$ to each edge e at $v \in \mathcal{V}$, depending on the edge and on the vertex. We search for the set $\boldsymbol{\sigma} = \{\sigma_v\}_{v\in\mathcal{V}} \in \mathcal{F}$, $\sigma_v \in \{-1, 1\}$ such that

$$\mathcal{C}^{k-\text{SAT}}[\boldsymbol{\sigma}] = \sum_{e\in\mathcal{E}} \prod_{v\in e} \frac{1 - J_e^v \sigma_v}{2} \tag{2.2.7}$$

is minimized. Again, in this case $|\mathcal{F}| = 2^V$ is exponential in the size V of the problem. Observe also that the cost function above, Eq. (2.2.7), reminds immediately a Hamiltonian function for a spin system on a graph. This analogy will become extremely relevant in the next Chapter.

MATCHING PROBLEMS Let us now consider a complete weighted graph $\mathtt{K}_{2N} = \text{Graph}(\mathcal{V}; \mathcal{E})$ with weight function $w\colon \mathcal{E} \to \mathbb{R}^+$. In the (monopartite) matching problem we want to find a perfect matching $\mathtt{M} \subseteq \mathtt{G}$ such that the cost functional

$$\mathcal{C}^{\text{M}}[\mathtt{M}] := \frac{1}{|\mathcal{E}_{\mathtt{M}}|} \sum_{e\in\mathcal{E}_{\mathtt{M}}} w(e) \tag{2.2.8}$$

is minimized. If $w(e) = 1\ \forall e \in \mathcal{E}$ the problem is sometimes called *cardinality matching problem*.

The *k-assignment problem* is formulated on the complete weighted bipartite graph $\mathtt{K}_{N,M}$. In this problem we ask for an optimal matching in the graph $\mathtt{K}_{N,M}$ of cardinality $k \le \min\{N, M\}$. In particular, the *bipartite matching problem*, or simply *assignment problem*, is the N-assignment problem on the complete bipartite graph $\mathtt{K}_{N,N}$. Observe that, in assignment problems, the matching can be represented as a permutation of N elements $\sigma \in \mathcal{P}_N$, $\mathcal{P}_N$ set of all permutations of N elements. Indeed, given the complete bipartite graph $\mathtt{K}_{N,N} = \text{Graph}(\mathcal{V}, \mathcal{U}; \mathcal{E})$, we can number the vertices as $\mathcal{V} = \{v_1, \dots, v_N\}$ and $\mathcal{U} = \{u_1, \dots, u_N\}$, and assume that $w\colon (v_i, u_j) \mapsto w_{ij}$. The optimal cost associated to the optimal matching $\mathtt{M}_o$ can be expressed therefore as

$$\mathcal{C}^{\text{M}}[\mathtt{M}_o] = \min_{\sigma\in\mathcal{P}_N} \frac{1}{N} \sum_{i=1}^{N} w_{i\,\sigma(i)}. \tag{2.2.9}$$

In the assignment problem there are $N!$ possible solutions. However, we will show that, from the algorithmic point of view, this problem belongs to the class of "simple" combinatorial problems and can be solved with quite fast algorithms.

POLYNOMIAL AND NON-DETERMINISTIC POLYNOMIAL ALGORITHMIC CLASSES

Given an instance of an optimization problem, the main target is often to find the optimal solution. However many different kinds of questions can be asked about an

optimization problem. For example, we may wonder if, for a given constant c, the set

$$S_c = \{x \in \mathcal{F} \colon \mathcal{C}[x] \leq c\} \tag{2.2.10}$$

is empty or not. This type of problem is called *decision problem*. In the theory of computational complexity [10, 12], each optimization problem is classified according to the running time (number of computational operations) and memory required to evaluate the decision problem or to find its solution. In particular, the class of *non-deterministic polynomial problems* NP is the set of problems such that, given a feasible solution x and a constant c, it is easy to evaluate if $x \in S_c$ or not. Here "easy" means that the check can be performed by a certain algorithm in a number of computational operations that is polynomial in the size of the input. An algorithm is *polynomial* if the running time is bounded from above by a certain polynomial in the size of the input and *superpolynomial* if such a bound does not exist. We say that a certain optimization problem belongs to the class $\mathtt{P} \subseteq \mathtt{NP}$, or that it is a *polynomial-time problem*, if there exists a polynomial algorithm that solves it. It is still unknown whether $\mathtt{P} = \mathtt{NP}$ or $\mathtt{P} \neq \mathtt{NP}$. In NP it is possible to identify a special set of problems, called NP-complete problems. Every problem in NP can be mapped in an NP-complete problem with, at most, an additional polynomial computational overhead. It follows that, if a NP-complete problem is found to be in P, it would follow that $\mathtt{P} = \mathtt{NP}$.

Among the problems discussed above, TSP, q-COL with $q > 2$ and k-SAT with $k > 2$, are NP-complete problems. The assignment problem, instead, belongs to the P computational complexity class. Indeed, we will show below that a fast algorithm is available for its solution.

2.2.2 Algorithms for Assignment

We discuss now two algorithms for the solution of the assignment problem. *Dantzig's algorithm*, called also *simplex method*, is a general algorithm for the solution of linear optimization problems. The assignment problem can be indeed seen as a linear optimization problem, as we will show below, and therefore the simplex method can be applied to it. The *Hungarian algorithm*, on the other hand, is the classical algorithm for the solution of the assignment problem: its computational complexity is polynomial and therefore the assignment problem is in P. Another very important algorithm, derived from the *cavity method*, will be discussed in the next Chapter.

Here we consider the assignment problem on a weighted complete bipartite graph

$$\mathtt{K}_{N,N} = \mathrm{Graph}(\mathcal{V}, \mathcal{U}; \mathcal{E}), \quad \mathcal{V} = \{v_i\}_{i=1,\ldots,N}, \quad \mathcal{U} = \{u_i\}_{i=1,\ldots,N}. \tag{2.2.11}$$

The weight function is such that

$$w\colon (v_i, u_j) \mapsto w_{ij} \in \mathbb{R}^+. \tag{2.2.12}$$

A matching can be represented by a $N \times N$ matrix $\mathsf{M} = (m_{ij})_{ij}$ such that

$$m_{ij} = \begin{cases} 1 & \text{if } (v_i, u_j) \in \mathcal{E}_{\mathrm{M}}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.2.13}$$

We can therefore identify the set of matchings on the graph $\mathtt{K}_{N,N}$ with the space of $N \times N$ matrices

$$\mathcal{F} = \left\{ \mathsf{M} = \left(m_{ij}\right)_{\substack{i=1,\dots,N \\ j=1,\dots,N}} \middle| \, m_{ij} \in \{0,1\} \text{ and } \sum_{i=1}^{N} m_{ij} = \sum_{j=1}^{N} m_{ij} = 1 \ \forall i, j \right\}. \tag{2.2.14}$$

The matching cost for a given $\mathsf{M} \in \mathcal{F}$ is defined as

$$\mathcal{C}^{\mathrm{M}}[\mathsf{M}] := \frac{1}{N} \sum_{i,j} w_{ij} m_{ij}. \tag{2.2.15}$$

It is evident that this is completely equivalent to Eq. (2.2.8). M_o is the optimal matching if and only if

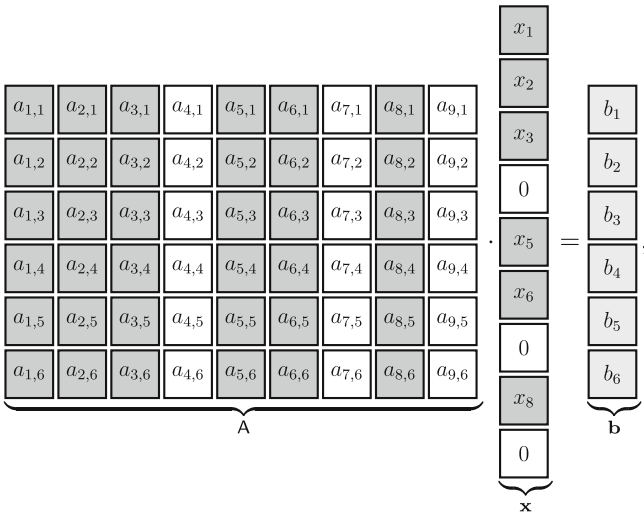
$$\mathcal{C}^{\mathrm{M}}[\mathsf{M}_o] := \min_{\mathsf{M} \in \mathcal{F}} \frac{1}{N} \sum_{i,j} w_{ij} m_{ij}. \tag{2.2.16}$$

THE SIMPLEX METHOD

G.B. Dantzig formulated the celebrated simplex algorithm for the algorithmic solution of linear optimization problems [2, 13]. Here we will sketch his approach, skipping some details that the reader can find properly treated in the cited references. As anticipated above, in a linear optimization problem we search for a vector $\mathbf{X} = (X_i)_{i=1,\dots,n} \in (\mathbb{R}^+)^n$ such that $\mathbf{c}^\intercal \cdot \mathbf{X} = \min_{\mathbf{x} \in \mathcal{F}} \mathbf{c}^\intercal \cdot \mathbf{x}$, where $\mathcal{F}$ is the space of vectors $\mathbf{x}$ of m real positive elements satisfying the constraint $\mathsf{A} \cdot \mathbf{x} = \mathbf{b}$. Here A is a matrix of $n \times m$ integers, $n > m$, $\mathbf{b}$ is a vector of m integers and $\mathbf{c}$ is a vector of n integers. We suppose that $\operatorname{rank}\left[\mathsf{A}\right] = m$. We can select therefore m linearly independent columns in A, let us call them $\mathsf{B} := \{\mathbf{a}_{i_k}\}_{k=1,\dots,m}$, such that the system

$$\mathsf{A} \cdot \mathbf{x} = \mathbf{b}, \qquad x_j = 0 \text{ if } j \neq i_k \, \forall k = 1, \dots m \tag{2.2.17}$$

has a unique solution. The problem is now in the form



(2.2.18)

where we have highlighted the submatrix B. If the solution $\mathbf{x}$ of the previous problem has $x_i \geq 0$, it is called *basic feasible solution* (BFS). Remarkably, it can be proved that the optimal solution that we are searching for, *is* a BFS. BFSs have an easy geometrical meaning. Indeed, let us consider the polytope associated to the matrix A. It can be proved that

$$\mathbf{x} \in \mathcal{F} \text{ and } \mathbf{x} \text{ is BFS} \Leftrightarrow \mathbf{x} \text{ is a vertex of the polytope.} \tag{2.2.19}$$

For any instance of a linear programming problem there is an optimal BFS. By the previous operation, let us suppose that a certain BFS $\mathbf{x}^*$ is known and that this BFS corresponds, for the sake of simplicity, to the set of columns $\{\mathbf{a}_i\}_{i=1,\ldots,m}$ of A, in such a way that $\mathbf{x}^* = (x_1^*, \ldots, x_m^*, 0, \ldots, 0)$. This solution can be the starting pointing of our search for the optimal solution. From what we said, indeed, an optimal solution can be found among the vertices of the polytope. The simplex method therefore is such that *we move from one vertex to another* through proper pivoting operations, until the optimal vertex is found.

The first step is to write down the *simplex tableaux*:

$$\begin{cases} \mathsf{A} \cdot \mathbf{x} = \mathbf{b} \\ -\mathbf{c}^\intercal \cdot \mathbf{x} + z = 0 \end{cases} \Rightarrow \left(\begin{array}{cc|c} \mathsf{A} & 0 & \mathbf{b} \\ -\mathbf{c}^\intercal & 1 & 0. \end{array}\right) \tag{2.2.20}$$

The structure of the matrix is therefore (in the pictures, $m = 6$ and $n = 9$)

$$\begin{array}{|cccccccccc|c|}
\hline
a_{1,1} & a_{2,1} & a_{3,1} & a_{4,1} & a_{5,1} & a_{6,1} & a_{7,1} & a_{8,1} & a_{9,1} & 0 & b_1 \\
a_{1,2} & a_{2,2} & a_{3,2} & a_{4,2} & a_{5,2} & a_{6,2} & a_{7,2} & a_{8,2} & a_{9,2} & 0 & b_2 \\
a_{1,3} & a_{2,3} & a_{3,3} & a_{4,3} & a_{5,3} & a_{6,3} & a_{7,3} & a_{8,3} & a_{9,3} & 0 & b_3 \\
a_{1,4} & a_{2,4} & a_{3,4} & a_{4,4} & a_{5,4} & a_{6,4} & a_{7,4} & a_{8,4} & a_{9,4} & 0 & b_4 \\
a_{1,5} & a_{2,5} & a_{3,5} & a_{4,5} & a_{5,5} & a_{6,5} & a_{7,5} & a_{8,5} & a_{9,5} & 0 & b_5 \\
a_{1,6} & a_{2,6} & a_{3,6} & a_{4,6} & a_{5,6} & a_{6,6} & a_{7,6} & a_{8,6} & a_{9,6} & 0 & b_6 \\
-c_1 & -c_2 & -c_3 & -c_4 & -c_5 & -c_6 & -c_7 & -c_8 & -c_9 & 1 & 0 \\
\hline
\end{array} \tag{2.2.21}$$

The first m columns of A correspond to the submatrix B associated to the solution $\mathbf{x}^*$ discussed above, that is our starting point. After a sequence of row operation we can transform the simplex tableaux in the following way

$$\begin{array}{|cccccccccc|c|}
\hline
1 & 0 & 0 & 0 & 0 & 0 & \tilde{a}_{7,1} & \tilde{a}_{8,1} & \tilde{a}_{9,1} & 0 & x_1^* \\
0 & 1 & 0 & 0 & 0 & 0 & \tilde{a}_{7,2} & \tilde{a}_{8,2} & \tilde{a}_{9,2} & 0 & x_2^* \\
0 & 0 & 1 & 0 & 0 & 0 & \tilde{a}_{7,3} & \tilde{a}_{8,3} & \tilde{a}_{9,3} & 0 & x_3^* \\
0 & 0 & 0 & 1 & 0 & 0 & \tilde{a}_{7,4} & \tilde{a}_{8,4} & \tilde{a}_{9,4} & 0 & x_4^* \\
0 & 0 & 0 & 0 & 1 & 0 & \tilde{a}_{7,5} & \tilde{a}_{8,5} & \tilde{a}_{9,5} & 0 & x_5^* \\
0 & 0 & 0 & 0 & 0 & 1 & \tilde{a}_{7,6} & \tilde{a}_{8,6} & \tilde{a}_{9,6} & 0 & x_6^* \\
0 & 0 & 0 & 0 & 0 & 0 & -\tilde{c}_7 & -\tilde{c}_8 & -\tilde{c}_9 & 1 & z_{\mathsf{B}} \\
\hline
\end{array} \tag{2.2.22}$$

where z_{B} is the value of z on the current BFS. The solution $\mathbf{x}^*$ appears in the last column, being x_i^* for $i = 1, \ldots, m$. If all the entries of $\{\tilde{c}_i\}_{i=m+1,\ldots,n}$ are positive, then the current BFS is optimal and z_{B} is the optimal cost. Indeed, if we consider a solution in the form $\mathbf{x}^{**} = (x_1^*, \ldots, x_m^*, x_{m+1}, 0, \ldots, 0)$, we will have for it $z = z_{\mathsf{B}} + \tilde{c}_{m+1} x_{m+1}^* > z_{\mathsf{B}}$.

If this is not the case, we have to proceed further. We choose a non-zero pivot element $\tilde{a}_{rc} \neq 0$ in the simplex tableaux, and we multiply the corresponding row for $\tilde{a}_{rc}^{-1}$. Proper multiples of the new row are added to the remaining rows of $\tilde{\mathsf{A}}$ in such a way that the c-th column of the new matrix has 1 in correspondence of the position (r, c) and zero otherwise. The chosen variable is a *new* basic variable and it is called *entering variable*. It substitute the old r-th basic variable, called now *leaving variable*. We switch the r-th column with the current c-th column to obtain a new simplex tableaux in the form (2.2.22). Due to the fact that the value of z must be minimized, the entering variable is chosen in a column c in such a way that $\tilde{c}_c < 0$

(non-zero values in the direction of the new selected component decrease z). The condition that the new solution must be *feasible* determines a criterion for the row: it can be shown that this condition implies that, being c the chosen pivot column, the row r must be such that $\tilde{a}_{rc}^{-1} x_r^*$ is minimum among all rows r. The iteration of this sequence of steps leads to the optimal solution exploring the BFSs. Note that the method requires as starting point a BFS.

Finally, observe that if we have constraints expressed in terms of inequalities, e.g.,

$$\sum_{j=1}^{n} a_{ij} x_j < b_i, \tag{2.2.23}$$

we can introduce a new *slack variable* $s_i \geq 0$ for each inequality and write it as $\sum_{j=1}^{n} a_{ij} x_j + s_i = b_i$. Obviously we can introduce a "dependence" of z from the new variables in a trivial way as $z = \sum_{i=1}^{n} c_i x_i + 0 \cdot \sum_j s_j$.

The simplex method is, practically, very efficient. However, Klee and Minty [7] proved that there exist linear problems for which, in the worst case, the convergence time of the simplex algorithm is exponentially large in the size of the input.

Application to the assignment problem

The simplex algorithm can be applied to the assignment problem quite straightforwardly [13]. Given a matching problem with a cost function as in Eq. (2.2.15), let us consider the following space of feasible solutions,

$$\mathcal{F} = \left\{ \mathbf{x} \in \left(\mathbb{R}^+\right)^{N^2} | \mathsf{A} \cdot \mathbf{x} = \mathbf{1} \right\}. \tag{2.2.24}$$

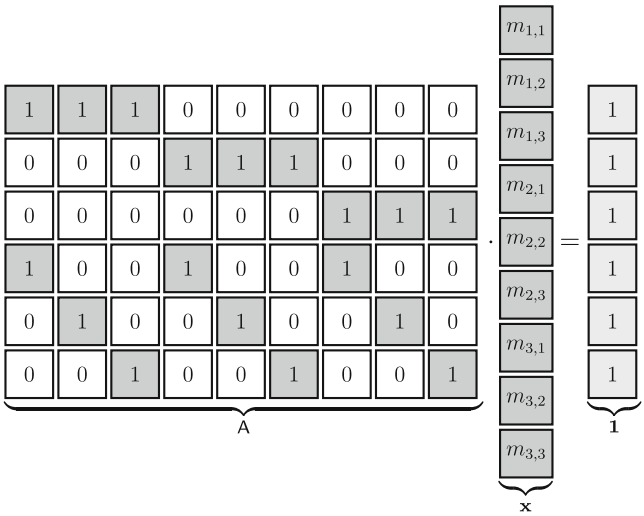
Here $\mathbf{1}$ is a vector of $2N$ elements all equal to 1 and $\mathsf{A} = (a_{ij})_{ij}$ is a $2N \times N^2$ matrix such that

$$a_{ij} = \begin{cases} 1 & \text{if } 1 \leq i \leq N \text{ and } (i-1)N < j \leq iN, \\ 1 & \text{if } N+1 \leq i \leq 2N \text{ and } j - i + N \mod N = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.2.25}$$

We search for the vector $\mathbf{x}_o$ such that

$$\mathbf{c}^\mathsf{T} \cdot \mathbf{x}_o = \min_{\mathbf{x} \in \mathcal{F}} \mathbf{c}^\mathsf{T} \cdot \mathbf{x}. \tag{2.2.26}$$

Here $\mathbf{c} = (c_i)_i$ is a N^2 dimensional column vector such that $w_{ij} = c_{(i-1)N+j}$ for $i = 1, \ldots, N$ and $j = 1, \ldots, N$. The vector $\mathbf{x} = (x_i)_i$ can be easily identified with a $N \times N$ matching matrix $\mathsf{M} = (m_{ij})_{ij}$, putting $m_{ij} \equiv x_{(i-1)N+j}$. With these identifications, the application of the simplex method is straightforward. For $N = 3$, for example, the constraint has therefore the form



(2.2.27)

THE HUNGARIAN ALGORITHM

We analyze now in some details the *Hungarian algorithm* [6] for the solution of the assignment problem on the complete graph[2] $\mathtt{K}_{N,N}$. This algorithm was proposed by Harold W. Kuhn in 1955 [8] and it has polynomial computational complexity, proving that the assignment problem belongs to the $\mathtt{P}$ computational complexity class. Indeed, the Hungarian algorithm, in the version of Munkres [11], has time complexity $O(N^4)$ in the size N of the input. Dinic and Kronrod [4] and Edmonds and Karp [5] were later able to decrease the computational complexity to $O(N^3)$. Remarkably, the Hungarian algorithm is deeply connected with the more general theory of the cavity method. The algorithm is named in honor of two Hungarian mathematicians, Dénes Kőnig and Jenő Egerváry, that proved the fundamental theoretical results behind the algorithm elaborated by Kuhn.

As above, we consider the complete bipartite graph $\mathtt{K}_{N,N} = \mathrm{Graph}(\mathcal{V}, \mathcal{U}; \mathcal{E})$, $\mathcal{V} = \{v_i\}_{i=1,\ldots,N}, \mathcal{U} = \{u_j\}_{j=1,\ldots,N}$. The target is to find the matching matrix M_0 such that the matching cost

$$\mathcal{C}^{\mathrm{M}}[\mathsf{M}] := \frac{1}{N} \sum_{i,j} w_{ij} m_{ij}, \quad w_{ij} \geq 0 \; \forall i, j, \tag{2.2.28}$$

is minimized. The matrix M_o minimizing the cost above is also a solution for the matching problem associated with the shifted cost

[2] More general polynomial algorithms are available to solve the matching problem on weighted graph $\mathtt{K}_{N,M}$, $N \neq M$.

$$\mathcal{C}^{\mathrm{M}}_{h_0}[\mathsf{M}] := \frac{1}{N}\sum_{i,j} w_{ij}m_{ij} + h_0. \tag{2.2.29}$$

In particular, the cost function in Eq. (2.2.29) is invariant under the *gauge transformation*

$$w_{ij} \mapsto w_{ij} - \lambda_i - \mu_j, \quad h_0 \mapsto h_0 + \frac{1}{N}\sum_{i=1}^{N}\lambda_i + \frac{1}{N}\sum_{j=1}^{N}\mu_j. \tag{2.2.30}$$

Gauge transformations work directly on the $N \times N$ weight matrix $\mathsf{W} = (w_{ij})_{ij}$, that becomes the only object to work on for the solution. The two column vectors of real values $\boldsymbol{\lambda} = (\lambda_i)_{i=1,\dots,N}$ and $\boldsymbol{\mu} = (\mu_i)_{i=1,\dots,N}$ identify the gauge transformation. We say that the gauge is *proper* if

$$w_{ij} \mapsto w_{ij} - \lambda_i - \mu_j \geq 0, \quad \forall i, j. \tag{2.2.31}$$

It is said in this case that the set $\{\lambda_1, \dots, \lambda_N, \mu_1, \dots, \mu_N\}$ is a *feasible node-weighting*. The following theorem about proper gauges holds [9].

Theorem 2.2.1 *(*Egerváry's theorem*) There exists a proper gauge* $(\boldsymbol{\lambda}^o, \boldsymbol{\mu}^o)$ *such that the cost of the optimal assignment* M_o *is given by*

$$\mathcal{C}^{\mathrm{M}}[\mathsf{M}_o] = \frac{\sum_{i=1}^{N}\lambda_i^o + \sum_{j=1}^{N}\mu_j^o}{N}. \tag{2.2.32}$$

Moreover, this value is maximal among all possible proper gauges, i.e.,

$$\mathcal{C}^{\mathrm{M}}[\mathsf{M}_o] = \max_{(\boldsymbol{\lambda},\boldsymbol{\mu})\ proper} \frac{\sum_{i=1}^{N}\lambda_i + \sum_{j=1}^{N}\mu_j}{N}. \tag{2.2.33}$$

Let us suppose that a proper gauge transformation has been performed on our matrix

$$\mathsf{W} \xrightarrow[\boldsymbol{\mu}]{\boldsymbol{\lambda}} \tilde{\mathsf{W}} = \mathsf{W} - \boldsymbol{\mu} \otimes \mathbf{1}^{\intercal} - \mathbf{1} \otimes \boldsymbol{\lambda}^{\intercal}, \tag{2.2.34}$$

where $\mathbf{1} = (1)_{i=1,\dots,N}$ is an N-dimensional column vector. We can construct, on the basis of the new weight matrix $\tilde{\mathsf{W}}$, the *equality subgraph* $\mathtt{Z} = \mathrm{Graph}(\mathcal{V}, \mathcal{U}; \mathcal{E}_{\mathtt{Z}}) \subseteq \mathtt{K}_{N,N}$ such that $e \in \mathcal{E}_{\mathtt{Z}} \Leftrightarrow \tilde{w}(e) = 0$, being $\tilde{w}(e)$ the weight associated to the edge e

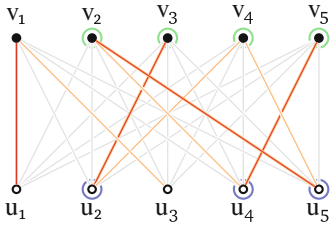


Fig. 2.5 The complete $\mathtt{K}_{5,5}$ with an equality graph $\mathtt{Z}$ (in color) and the corresponding maximum matching $\mathtt{M}$ (in red) on it. An example of set $\mathcal{V}^*$ (green) with corresponding $\partial_{\mathtt{Z}}\mathcal{V}^*$ (blue) is also depicted

by the new weight matrix $\tilde{\mathbf{W}}$. If $\mathtt{Z}$ contains a perfect matching $\mathtt{M}_o \subseteq \mathtt{Z} \subseteq \mathtt{K}_{N,N}$, then $\mathtt{M}_o$ is the optimal matching,[3] having a matching cost given by the Eq. (2.2.32).

If $\mathtt{Z}$ does not contain a perfect matching, we have to perform a new gauge transformation (see Fig. 2.5). We search now in $\mathcal{V}$ for a set of vertices $\mathcal{V}^*$ such that $|\partial_{\mathtt{Z}}\mathcal{V}^*| < |\mathcal{V}^*|$. The existence of such subset is guaranteed by the Kőnig's theorem (see the formulation of Hall). We apply then the following gauge $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$:

$$\boldsymbol{\lambda}^* = (\lambda_i^*)_i : \lambda_i^* = \begin{cases} \gamma & \text{if } v_i \in \mathcal{V}^*, \\ 0 & \text{if } v_i \notin \mathcal{V}^*, \end{cases} \tag{2.2.35}$$

$$\boldsymbol{\mu}^* = (\mu_j^*)_j : \mu_j^* = \begin{cases} -\gamma & \text{if } u_j \in \partial_{\mathtt{Z}}\mathcal{V}^*, \\ 0 & \text{if } u_j \notin \partial_{\mathtt{Z}}\mathcal{V}^*, \end{cases} \tag{2.2.36}$$

where

$$\gamma = \min_{\substack{v_i \in \mathcal{V}^* \\ u_j \notin \partial_{\mathtt{Z}}\mathcal{V}^*}} \tilde{w}_{ij}. \tag{2.2.37}$$

The obtained weight matrix can be associated again to a new equality subgraph, in which we search again for a perfect matching. The algorithm proceeds repeating the last steps (see Fig. 2.6) until a perfect matching is found. The convergence of the algorithm in polynomial time is proved, e.g., in [6].

[3] Observe that different matching solutions are in general possible, but corresponding to the same cost.

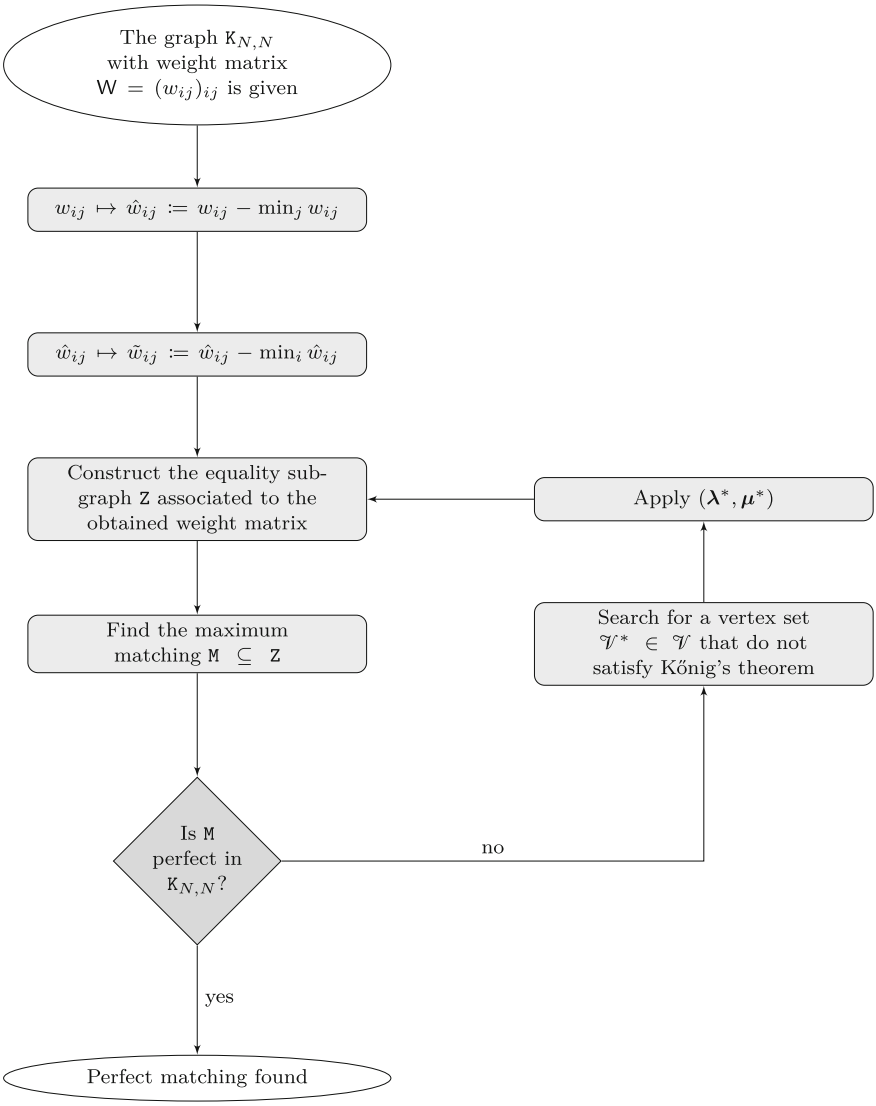


Fig. 2.6 The Hungarian algorithm, with reference to the description given in the main text

References

1. C. Berge, *Graphs and Hypergraphs* (North-Holland Mathematical Library, Amsterdam, 1973)
2. G. Dantzig, M. Thapa, *Linear Programming 1: Introduction*, Springer Series in Operations Research and Financial Engineering (Springer, New York, 1997)
3. R. Diestel, *Graph Theory*, Springer Graduate Texts in Mathematics (GTM) (Springer, Heidelberg, 2012)
4. E. Dinic, M. Kronrod, An algorithm for the solution of the assignment problem. Sov. Math. Dokl. **10**(6), 1324–1326 (1969)
5. J. Edmonds, R.M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM (JACM) **19**(2), 248–264 (1972)
6. D. Jungnickel, T. Schade, *Graphs, Networks and Algorithms* (Springer, Berlin, 2005)
7. V. Klee, G.J. Minty, How good is the simplex algorithm. Technical report, DTIC Document (1970)
8. H.W. Kuhn, The Hungarian method for the assignment problem. Nav. Res. Logist. Q. **2**, 83–97 (1955)
9. L. Lovász, D. Plummer, *Matching Theory*. AMS Chelsea Publishing Series (AMS, Providence, 2009)
10. C. Moore, S. Mertens, *The Nature of Computation* (OUP, Oxford, 2011)
11. J. Munkres, Algorithms for the assignment and transportation problems. J. Soc. Ind. Appl. Math. **5**(1), 32–38 (1957)
12. C. Papadimitriou, *Computational Complexity*, Theoretical computer science (Addison-Wesley, Reading, 1994)
13. C. Papadimitriou, K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover Books on Computer Science Series (Dover Publications, Mineola, 1998)

http://www.springer.com/978-3-319-46576-0

The Euclidean Matching Problem

Sicuro, G.

2017, XIV, 136 p. 50 illus., 6 illus. in color., Hardcover
ISBN: 978-3-319-46576-0