Chapter 1

Introduction

1.1. Introduction

Combinatorial optimization searches for an optimum object in a finite collection of objects. Typically, the collection has a concise representation (like a graph), while the number of objects is huge — more precisely, grows exponentially in the size of the representation (like all matchings or all Hamiltonian circuits). So scanning all objects one by one and selecting the best one is not an option. More efficient methods should be found.

In the 1960s, Edmonds advocated the idea to call a method efficient if its running time is bounded by a polynomial in the size of the representation. Since then, this criterion has won broad acceptance, also because Edmonds found polynomial-time algorithms for several important combinatorial optimization problems (like the matching problem). The class of polynomial-time solvable problems is denoted by P.

Further relief in the landscape of combinatorial optimization was discovered around 1970 when Cook and Karp found out that several other prominent combinatorial optimization problems (including the traveling salesman problem) are the hardest in a large natural class of problems, the class NP. The class NP includes most combinatorial optimization problems. Any problem in NP can be reduced to such 'NP-complete' problems. All NP-complete problems are equivalent in the sense that the polynomial-time solvability of one of them implies the same for all of them.

Almost every combinatorial optimization problem has since been either proved to be polynomial-time solvable or NP-complete — and none of the problems have been proved to be both. This spotlights the big mystery: are the two properties disjoint (equivalently, P$\neq$NP), or do they coincide (P=NP)?

This book focuses on those combinatorial optimization problems that have been proved to be solvable in polynomial time, that is, those that have been proved to belong to P. Next to polynomial-time solvability, we focus on the related polyhedra and min-max relations.

These three aspects have turned out to be closely related, as was shown also by Edmonds. Often a polynomial-time algorithm yields, as a by-product,

a description (in terms of inequalities) of an associated polyhedron. Conversely, an appropriate description of the polyhedron often implies the polynomial-time solvability of the associated optimization problem, by applying linear programming techniques. With the duality theorem of linear programming, polyhedral characterizations yield min-max relations, and vice versa.

So the span of this book can be portrayed alternatively by those combinatorial optimization problems that yield well-described polyhedra and min-max relations. This field of discrete mathematics is called *polyhedral combinatorics*. In the following sections we give some basic, illustrative examples.[1]

1.2. Matchings

Let $G = (V, E)$ be an undirected graph and let $w : E \to \mathbb{R}_+$. For any subset F of E, denote

$$w(F) := \sum_{e \in F} w(e). \tag{1.1}$$

We will call $w(F)$ the *weight* of F.

Suppose that we want to find a *matching* (= set of disjoint edges) M in G with weight $w(M)$ as large as possible. In notation, we want to 'solve'

$$\max\{w(M) \mid M \text{ matching in } G\}. \tag{1.2}$$

We can formulate this problem equivalently as follows. For any matching M, denote the incidence vector of M in $\mathbb{R}^E$ by χ^M; that is,

$$\chi^M(e) := \begin{cases} 1 & \text{if } e \in M, \\ 0 & \text{if } e \notin M, \end{cases} \tag{1.3}$$

for $e \in E$. Considering w as a *vector* in $\mathbb{R}^E$, we have $w(M) = w^{\mathsf{T}}\chi^M$. Hence problem (1.2) can be rewritten as

$$\max\{w^{\mathsf{T}}\chi^M \mid M \text{ matching in } G\}. \tag{1.4}$$

This amounts to maximizing the linear function $w^{\mathsf{T}}x$ over a finite set of vectors. Therefore, the optimum value does not change if we maximize over the *convex hull* of these vectors:

$$\max\{w^{\mathsf{T}}x \mid x \in \text{conv.hull}\{\chi^M \mid M \text{ matching in } G\}\}. \tag{1.5}$$

The set

$$\text{conv.hull}\{\chi^M \mid M \text{ matching in } G\} \tag{1.6}$$

is a polytope in $\mathbb{R}^E$, called the *matching polytope* of G. As it is a polytope, there exist a matrix A and a vector b such that

[1] Terms used but not introduced yet can be found later in this book — consult the Subject Index.

(1.7) $$\text{conv.hull}\{\chi^M \mid M \text{ matching in } G\} = \{x \in \mathbb{R}^E \mid x \geq \mathbf{0}, Ax \leq b\}.$$

Then problem (1.5) is equivalent to

(1.8) $$\max\{w^{\mathsf{T}}x \mid x \geq \mathbf{0}, Ax \leq b\}.$$

In this way we have formulated the original combinatorial problem (1.2) as a *linear programming* problem. This enables us to apply linear programming methods to study the original problem.

The question at this point is, however, how to find the matrix A and the vector b. We know that A and b do exist, but we must know them in order to apply linear programming methods.

If G is bipartite, it turns out that the matching polytope of G is equal to the set of all vectors $x \in \mathbb{R}^E$ satisfying

(1.9) $$\begin{aligned} &x(e) \geq 0 && \text{for } e \in E,\\ &\sum_{e \ni v} x(e) \leq 1 && \text{for } v \in V. \end{aligned}$$

(The sum ranges over all edges e containing v.) That is, for A we can take the $V \times E$ incidence matrix of G and for b the all-one vector $\mathbf{1}$ in $\mathbb{R}^V$.

It is not difficult to show that the matching polytope for bipartite graphs is indeed completely determined by (1.9). First note that the matching polytope is contained in the polytope determined by (1.9), since χ^M satisfies (1.9) for each matching M. To see the reverse inclusion, we note that, if G is bipartite, then the matrix A is *totally unimodular*, i.e., each square submatrix has determinant belonging to $\{0, +1, -1\}$. (This easy fact will be proved in Section 18.2.) The total unimodularity of A implies that the vertices of the polytope determined by (1.9) are *integer* vectors, i.e., belong to $\mathbb{Z}^E$. Now each integer vector satisfying (1.9) must trivially be equal to χ^M for some matching M. Hence, if G is bipartite, the matching polytope is determined by (1.9).

We therefore can apply linear programming techniques to handle problem (1.2). Thus we can find a maximum-weight matching in a bipartite graph in polynomial time, with any polynomial-time linear programming algorithm. Moreover, the duality theorem of linear programming gives

(1.10) $$\begin{aligned} &\max\{w(M) \mid M \text{ matching in } G\} = \max\{w^{\mathsf{T}}x \mid x \geq \mathbf{0}, Ax \leq \mathbf{1}\}\\ &= \min\{y^{\mathsf{T}}\mathbf{1} \mid y \geq \mathbf{0}, y^{\mathsf{T}}A \geq w^{\mathsf{T}}\}. \end{aligned}$$

If we take for w the all-one vector $\mathbf{1}$ in $\mathbb{R}^E$, we can derive from this Kőnig's matching theorem (Kőnig [1931]):

(1.11) the maximum size of a matching in a bipartite graph is equal to the minimum size of a vertex cover,

where a *vertex cover* is a set of vertices intersecting each edge. Indeed, the left-most expression in (1.10) is equal to the maximum size of a matching. The minimum can be seen to be attained by an integer vector y, again by

the total unimodularity of A. This vector y is a $0,1$ vector in $\mathbb{R}^V$, and hence is the incidence vector χ^U of some subset U of V. Then $y^\mathsf{T} A \geq \mathbf{1}^\mathsf{T}$ implies that U is a vertex cover. Therefore, the right-most expression is equal to the minimum size of a vertex cover.

Kőnig's matching theorem (1.11) is an example of a *min-max formula* that can be derived from a polyhedral characterization. Conversely, min-max formulas (in particular in a weighted form) often give polyhedral characterizations.

The polyhedral description together with linear programming duality also gives a *certificate* of optimality of a matching M: to convince your 'boss' that a certain matching M has maximum size, it is possible and sufficient to display a vertex cover of size $|M|$. In other words, it yields a *good characterization* for the maximum-size matching problem in bipartite graphs.

1.3. But what about nonbipartite graphs?

If G is *nonbipartite*, the matching polytope is not determined by (1.9): if C is an odd circuit in G, then the vector $x \in \mathbb{R}^E$ defined by $x(e) := \frac{1}{2}$ if $e \in EC$ and $x(e) := 0$ if $e \notin EC$, satisfies (1.9) but does not belong to the matching polytope of G.

A pioneering and central theorem in polyhedral combinatorics of Edmonds [1965b] gives a complete description of the inequalities needed to describe the matching polytope for arbitrary graphs: one should add to (1.9) the inequalities

$$\sum_{e \subseteq U} x(e) \leq \lfloor \tfrac{1}{2}|U| \rfloor \text{ for each odd-size subset } U \text{ of } V. \tag{1.12}$$

Trivially, the incidence vector χ^M of any matching M satisfies (1.12). So the matching polytope of G is contained in the polytope determined by (1.9) and (1.12). The content of Edmonds' theorem is the converse inclusion. This will be proved in Chapter 25.

In fact, Edmonds designed a polynomial-time algorithm to find a maximum-weight matching in a graph, which gave this polyhedral characterization as a by-product. Conversely, from the characterization one may derive the polynomial-time solvability of the weighted matching problem, with the ellipsoid method. In applying linear programming methods for this, one will be faced with the fact that the system $Ax \leq b$ consists of exponentially many inequalities, since there exist exponentially many odd-size subsets U of V. So in order to solve the problem with linear programming methods, we cannot just list all inequalities. However, the ellipsoid method does not require that all inequalities are listed a priori. It suffices to have a polynomial-time algorithm answering the question:

(1.13) given $x \in \mathbb{R}^E$, does x belong to the matching polytope of G?

Such an algorithm indeed exists, as it has been shown that the inequalities (1.9) and (1.12) can be checked in time bounded by a polynomial in $|V|$, $|E|$, and the size of x. This method obviously should avoid testing all inequalities (1.12) one by one.

Combining the description of the matching polytope with the duality theorem of linear programming gives a min-max formula for the maximum weight of a matching. It again yields a certificate of optimality: if we have a matching M, we can convince our 'boss' that M has maximum weight, by supplying a dual solution y of objective value $w(M)$. So the maximum-weight matching problem has a good characterization — i.e., belongs to NP∩co-NP.

This gives one motivation for studying polyhedral methods. The ellipsoid method proves polynomial-time solvability, it however does not yield a practical method, but rather an incentive to search for a practically efficient algorithm. The polyhedral method can be helpful also in this, e.g., by imitating the simplex method with a constraint generation technique, or by a primal-dual approach.

1.4. Hamiltonian circuits and the traveling salesman problem

As we discussed above, matching is an area where the search for an inequality system determining the corresponding polytope has been successful. This is in contrast with, for instance, Hamiltonian circuits. No full description in terms of inequalities of the convex hull of the incidence vectors of Hamiltonian circuits — the *traveling salesman polytope* — is known. The corresponding optimization problem is the traveling salesman problem: 'find a Hamiltonian circuit of minimum weight', which problem is NP-complete. This implies that, unless NP=co-NP, there exist facet-inducing inequalities for the traveling salesman polytope that have no polynomial-time certificate of validity. Otherwise, linear programming duality would yield a good characterization. So unless NP=co-NP there is no hope for an appropriate characterization of the traveling salesman polytope.

Moreover, unless NP=P, there is no polynomial-time algorithm answering the question

(1.14) given $x \in \mathbb{R}^E$, does x belong to the traveling salesman polytope?

Otherwise, the ellipsoid method would give the polynomial-time solvability of the traveling salesman problem.

Nevertheless, polyhedral combinatorics can be applied to the traveling salesman problem in a positive way. If we include the traveling salesman polytope in a larger polytope (a *relaxation*) over which we *can* optimize in polynomial time, we obtain a polynomial-time computable bound for the traveling salesman problem. The closer the relaxation is to the traveling salesman polytope, the better the bound is. This can be very useful in a

branch-and-bound algorithm. This idea originates from Dantzig, Fulkerson, and Johnson [1954b].

1.5. Historical and further notes

1.5a. Historical sketch on polyhedral combinatorics

The first min-max relations in combinatorial optimization were proved by Dénes Kőnig [1916,1931], on edge-colouring and matchings in bipartite graphs, and by Karl Menger [1927], on disjoint paths in graphs. The matching theorem of Kőnig was extended to the weighted case by Egerváry [1931]. The proofs by Kőnig and Egerváry were in principal algorithmic, and also for Menger's theorem an algorithmic proof was given in the 1930s. The theorem of Egerváry may be seen as polyhedral.

Applying linear programming techniques to combinatorial optimization problems came along with the introduction of linear programming in the 1940s and 1950s. In fact, linear programming forms the hinge in the history of combinatorial optimization. Its initial conception by Kantorovich and Koopmans was motivated by combinatorial applications, in particular in transportation and transshipment.

After the formulation of linear programming as generic problem, and the development in 1947 by Dantzig of the simplex method as a tool, one has tried to attack about all combinatorial optimization problems with linear programming techniques, quite often very successfully. In the 1950s, Dantzig, Ford, Fulkerson, Hoffman, Kuhn, and others studied problems like the transportation, maximum flow, and assignment problems. These problems can be reduced to linear programming by the total unimodularity of the underlying matrix, thus yielding extensions and polyhedral and algorithmic interpretations of the earlier results of Kőnig, Egerváry, and Menger. Kuhn realized that the polyhedral methods of Egerváry for weighted bipartite matching are in fact algorithmic, and yield the efficient 'Hungarian' method for the assignment problem. Dantzig, Fulkerson, and Johnson gave a solution method for the traveling salesman problem, based on linear programming with a rudimentary, combinatorial version of a cutting plane technique.

A considerable extension and deepening, and a major justification, of the field of polyhedral combinatorics was obtained in the 1960s and 1970s by the work and pioneering vision of Jack Edmonds. He characterized basic polytopes like the matching polytope, the arborescence polytope, and the matroid intersection polytope; he introduced (with Giles) the important concept of total dual integrality; and he advocated the interconnections between polyhedra, min-max relations, good characterizations, and efficient algorithms. We give a few quotes in which Edmonds enters into these issues.

In his paper presenting a maximum-size matching algorithm, Edmonds [1965d] gave a polyhedral argument why an algorithm can lead to a min-max theorem:

> It is reasonable to hope for a theorem of this kind because any problem which involves maximizing a linear form by one of a discrete set of non-negative vectors has associated with it a dual problem in the following sense. The discrete set of vectors has a convex hull which is the intersection of a discrete set of half-spaces. The value of the linear form is as large for some vector of the discrete set

> as it is for any other vector in the convex hull. Therefore, the discrete problem is equivalent to an ordinary linear programme whose constraints, together with non-negativity, are given by the half-spaces. The dual (more precisely, a dual) of the discrete problem is the dual of this ordinary linear programme.
> For a class of discrete problems, formulated in a natural way, one may hope then that equivalent linear constraints are pleasant enough though they are not explicit in the discrete formulation.

In another paper (characterizing the matching polytope), Edmonds [1965b] stressed that the number of inequalities is not relevant:

> The results of this paper suggest that, in applying linear programming to a combinatorial problem, the number of relevant inequalities is not important but their combinatorial structure is.

Also in a discussion at the IBM Scientific Computing Symposium on Combinatorial Problems (March 1964 in Yorktown Heights, New York), Edmonds emphasized that the number of facets of a polyhedron is not a measure of the complexity of the associated optimization problem (see Gomory [1966]):

> I do not believe there is any reason for taking as a measure of the algorithmic difficulty of a class of combinatorial extremum problems the number of faces in the associated polyhedra. For example, consider the generalization of the assignment problem from bipartite graphs to arbitrary graphs. Unlike the case of bipartite graphs, the number of faces in the associated polyhedron increases exponentially with the size of the graph. On the other hand, there is an algorithm for this generalized assignment problem which has an upper bound on the work involved just as good as the upper bound for the bipartite assignment problem.

After having received support from H.W. Kuhn and referring to Kuhn's maximum-weight bipartite matching algorithm, Edmonds continued:

> This algorithm depends crucially on what amounts to knowing all the bounding inequalities of the associated convex polyhedron—and, as I said, there are many of them. The point is that the inequalities are known by an easily verifiable characterization rather than by an exhaustive listing—so their number is not important.
> This sort of thing should be expected for a class of extremum problems with a combinatorially special structure. For the traveling salesman problem, the vertices of the associated polyhedron have a simple characterization despite their number—so might the bounding inequalities have a simple characterization despite their number. At least we should hope they have, because finding a really good traveling salesman algorithm is undoubtedly equivalent to finding such a characterization.

So Edmonds was aware of the correlation of good algorithms and polyhedral characterizations, which later got further support by the ellipsoid method.

Also during the 1960s and 1970s, Fulkerson designed the clarifying framework of blocking and antiblocking polyhedra, throwing new light by the classical polarity of vertices and facets of polyhedra on combinatorial min-max relations and enabling, with a theorem of Lehman, the deduction of one polyhedral characterization from another. It stood at the basis of the solution of Berge's perfect graph conjecture in 1972 by Lovász, and it also inspired Seymour to obtain several other basic results in polyhedral combinatorics.

1.5b. Further notes

Raghavan and Thompson [1987] showed that randomized rounding of an optimum fractional solution to a combinatorial optimization problem yields, with high probability, an integer solution with objective value close to the value of the fractional solution (hence at least as close to the optimum value of the combinatorial problem). Related results were presented by Raghavan [1988], Plotkin, Shmoys, and Tardos [1991,1995], and Srinivasan [1995,1999].

Introductions to combinatorial optimization (and more than that) can be found in the books by Lawler [1976b], Papadimitriou and Steiglitz [1982], Sysło, Deo, and Kowalik [1983], Nemhauser and Wolsey [1988], Parker and Rardin [1988], Cook, Cunningham, Pulleyblank, and Schrijver [1998], Mehlhorn and Näher [1999], and Korte and Vygen [2000]. Focusing on applying geometric algorithms in combinatorial optimization are Lovász [1986] and Grötschel, Lovász, and Schrijver [1988]. Bibliographies on combinatorial optimization were given by Kastning [1976], Golden and Magnanti [1977], Hausmann [1978b], von Randow [1982,1985,1990], and O'hEigeartaigh, Lenstra, and Rinnooy Kan [1985].

Survey papers on polyhedral combinatorics and min-max relations were presented by Hoffman [1979], Pulleyblank [1983,1989], Schrijver [1983a,1986a,1987, 1995], and Grötschel [1985], on geometric methods in combinatorial optimization by Grötschel, Lovász, and Schrijver [1984b], and on polytopes and complexity by Papadimitriou [1984].

Chapter 2

General preliminaries

We give general preliminaries on sets, numbers, orders, vectors, matrices, and functions, we discuss how to interpret maxima, minima, and infinity, and we formulate and prove Fekete's lemma.

2.1. Sets

A large part of the sets considered in this book are finite. We often neglect mentioning this when introducing a set. For instance, graphs in this book are finite graphs, except if we explicitly mention otherwise. Similarly for other structures like hypergraphs, matroids, families of sets, etc. Obvious exceptions are the sets of reals, integers, etc.

We call a subset Y of a set X *proper* if $Y \neq X$. Similarly, any other substructure like subgraph, minor, etc. is called *proper* if it is not equal to the structure of which it is a substructure.

A *family* is a set in which elements may occur more than once. More precisely, each element has a *multiplicity* associated. Sometimes, we indicate a family by $(A_1, \dots, A_n)$ or $(A_i \mid i \in I)$.

A *collection* is synonymous with *set*, but is usually used for a set whose elements are sets. Also *class* and *system* are synonyms of set, and are usually used for sets of structures, like a set of graphs, inequalities, or curves.

A set is called *odd* (*even*) if its size is odd (even). We denote for any set X:

$$\begin{aligned} &\mathcal{P}(X) := \text{collection of all subsets of } X,\\ &\mathcal{P}_{\text{odd}}(X) := \text{collection of all odd subsets } Y \text{ of } X,\\ &\mathcal{P}_{\text{even}}(X) := \text{collection of all even subsets } Y \text{ of } X. \end{aligned} \tag{2.1}$$

Odd and even are called *parities*.

We sometimes say that if $s \in U$, then U *covers* s and s *covers* U. A set U is said to *separate* s and t if $s \neq t$ and $|U \cap \{s,t\}| = 1$. Similarly, a set U is said to *separate* sets S and T if $S \cap T = \emptyset$ and $U \cap (S \cup T) \in \{S, T\}$.

We denote the *symmetric difference* of two sets S and T by $S \triangle T$:

$$S \triangle T = (S \setminus T) \cup (T \setminus S). \tag{2.2}$$

We sometimes use the following shorthand notation, where X is a set and y an 'element':

(2.3) $$X + y := X \cup \{y\} \text{ and } X - y := X \setminus \{y\}.$$

We say that sets $S_1, S_2, \ldots, S_k$ are *disjoint* if they are *pairwise* disjoint:

(2.4) $$S_i \cap S_j = \emptyset \text{ for distinct } i, j \in \{1, \ldots, k\}.$$

A *partition* of a set X is a collection of disjoint subsets of X with union X. The elements of the partition are called its *classes*.

As usual:

(2.5) $X \subseteq Y$ means that X is a subset of Y,
$X \subset Y$ means that X is a *proper* subset of Y, that is: $X \subseteq Y$ and $X \neq Y$.

Two sets X, Y are *comparable* if $X \subseteq Y$ or $Y \subseteq X$. A collection of pairwise comparable sets is called a *chain*.

Occasionally, we need the following inequality:

Theorem 2.1. *If T and U are subsets of a set S with $T \not\subseteq U$ and $U \not\subseteq T$, then*

(2.6) $$|T||\overline{T}| + |U||\overline{U}| > |T \cap U||\overline{T \cap U}| + |T \cup U||\overline{T \cup U}|,$$

where $\overline{X} := S \setminus X$ for any $X \subseteq S$.

Proof. Define $\alpha := |T \cap U|$, $\beta := |T \setminus U|$, $\gamma := |U \setminus T|$, and $\delta := |\overline{T \cup U}|$. Then:

(2.7) $$\begin{aligned} |T||\overline{T}| + |U||\overline{U}| &= (\alpha + \beta)(\gamma + \delta) + (\alpha + \gamma)(\beta + \delta) \\ &= 2\alpha\delta + 2\beta\gamma + \alpha\gamma + \beta\delta + \alpha\beta + \gamma\delta \end{aligned}$$

and

(2.8) $$\begin{aligned} |T \cap U||\overline{T \cap U}| + |T \cup U||\overline{T \cup U}| &= \alpha(\beta + \gamma + \delta) + (\alpha + \beta + \gamma)\delta \\ &= 2\alpha\delta + \alpha\gamma + \beta\delta + \alpha\beta + \gamma\delta. \end{aligned}$$

Since $\beta\gamma > 0$, (2.6) follows. ∎

A set U is called an *inclusionwise minimal* set in a collection $\mathcal{C}$ of sets if $U \in \mathcal{C}$ and there is no $T \in \mathcal{C}$ with $T \subset U$. Similarly, U is called an *inclusionwise maximal* set in $\mathcal{C}$ if $U \in \mathcal{C}$ and there is no $T \in \mathcal{C}$ with $T \supset U$.

We sometimes use the term *minimal* for *inclusionwise* minimal, and *minimum* for minimum-*size*. Similarly, we sometimes use *maximal* for *inclusionwise* maximal, and *maximum* for maximum-*size* (or maximum-*value* for flows).

A *metric* on a set V is a function $\mu : V \times V \to \mathbb{R}_+$ such that $\mu(v, v) = 0$, $\mu(u, v) = \mu(v, u)$, and $\mu(u, w) \leq \mu(u, v) + \mu(v, w)$ for all $u, v, w \in V$.

2.2. Orders

A relation $\leq$ on a set X is called a *pre-order* if it is reflexive ($x \leq x$ for all $x \in X$) and transitive ($x \leq y$ and $y \leq z$ implies $x \leq z$). It is a *partial order* if it is moreover anti-symmetric ($x \leq y$ and $y \leq x$ implies $x = y$). The pair $(X, \leq)$ is called a *partially ordered set* if $\leq$ is a partial order.

A partial order $\leq$ is a *linear order* or *total order* if $x \leq y$ or $y \leq x$ for all $x, y \in X$. If $X = \{x_1, \ldots, x_n\}$ and $x_1 < x_2 < \cdots < x_n$, we occasionally refer to the linear order $\leq$ by $x_1, \ldots, x_n$ or $x_1 < \cdots < x_n$. A linear order $\preceq$ is called a *linear extension* of a partial order $\leq$ if $x \leq y$ implies $x \preceq y$.

In a partially ordered set $(X, \leq)$, a *lower ideal* is a subset Y of X such that if $y \in Y$ and $z \leq y$, then $z \in Y$. Similarly, an *upper ideal* is a subset Y of X such that if $y \in Y$ and $z \geq y$, then $z \in Y$. Alternatively, Y is called *down-monotone* if Y is a lower ideal, and *up-monotone* if Y is an upper ideal.

If $(X, \leq)$ is a linearly ordered set, then the *lexicographic order* $\preceq$ on $\bigcup_{k \geq 0} X^k$ is defined by:

$$(v_1, \ldots, v_t) \prec (u_1, \ldots, u_s) \iff \text{the smallest } i \text{ with } v_i \neq u_i \text{ satisfies } v_i < u_i, \tag{2.9}$$

where we set $v_i :=$void if $i > t$, $u_i :=$void if $i > s$, and void$< x$ for all $x \in X$.

2.3. Numbers

$\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ denote the sets of integers, rational numbers, and real numbers, respectively. The subscript $+$ restricts the sets to the nonnegative numbers:

$$\begin{aligned} &\mathbb{Z}_+ := \{x \in \mathbb{Z} \mid x \geq 0\},\ \mathbb{Q}_+ := \{x \in \mathbb{Q} \mid x \geq 0\},\\ &\mathbb{R}_+ := \{x \in \mathbb{R} \mid x \geq 0\}. \end{aligned} \tag{2.10}$$

Further we denote for any $x \in \mathbb{R}$:

$$\begin{aligned} &\lfloor x \rfloor := \text{largest integer } y \text{ satisfying } y \leq x,\\ &\lceil x \rceil := \text{smallest integer } y \text{ satisfying } y \geq x. \end{aligned} \tag{2.11}$$

2.4. Vectors, matrices, and functions

All vectors are assumed to be *column* vectors. The *components* or *entries* of a vector $x = (x_1, \ldots, x_n)^\mathsf{T}$ are $x_1, \ldots, x_n$. The *support* of x is the set of indices i with $x_i \neq 0$. The *size* of a vector x is the sum of its components.

A $0,1$ *vector*, or a $\{0,1\}$*-valued vector*, or a *simple vector*, is a vector with all entries in $\{0,1\}$. An *integer vector* is a vector with all entries integer.

We identify the concept of a *function* $x : V \to \mathbb{R}$ with that of a *vector* x in $\mathbb{R}^V$. Its components are denoted equivalently by $x(v)$ or x_v. An *integer function* is an integer-valued function.

For any $U \subseteq V$, the *incidence vector* of U (in $\mathbb{R}^V$) is the vector χ^U defined by:

$$\chi^U(s) := \begin{cases} 1 & \text{if } s \in U, \\ 0 & \text{if } s \notin U. \end{cases} \tag{2.12}$$

For any $u \in V$ we set

$$\chi^u := \chi^{\{u\}}. \tag{2.13}$$

This is the uth *unit base vector*. Given a vector space $\mathbb{R}^V$ for some set V, the all-one vector is denoted by $\mathbf{1}_V$ or just by $\mathbf{1}$, and the all-zero vector by $\mathbf{0}_V$ or just by $\mathbf{0}$. Similarly, $\mathbf{2}_V$ or $\mathbf{2}$ is the all-two vector. We use $\boldsymbol{\infty}$ for the all-∞ vector.

If $a = (a_1, \ldots, a_n)^\mathsf{T}$ and $b = (b_1, \ldots, b_n)^\mathsf{T}$ are vectors, we write $a \leq b$ if $a_i \leq b_i$ for $i = 1, \ldots, n$, and $a < b$ if $a_i < b_i$ for $i = 1, \ldots, n$.

If A is a matrix and x, b, y, and c are vectors, then when using notation like

$$Ax = b,\ Ax \leq b,\ y^\mathsf{T}A = c^\mathsf{T},\ c^\mathsf{T}x, \tag{2.14}$$

we often implicitly assume compatibility of dimensions.

For any vector $x = (x_1, \ldots, x_n)^\mathsf{T}$:

$$\|x\|_1 := |x_1| + \cdots + |x_n| \text{ and } \|x\|_\infty := \max\{|x_1|, \ldots, |x_n|\}. \tag{2.15}$$

A *hyperplane* in $\mathbb{R}^n$ is a set H with $H = \{x \in \mathbb{R}^n \mid c^\mathsf{T}x = \delta\}$ for some $c \in \mathbb{R}^n$ with $c \neq \mathbf{0}$ and some $\delta \in \mathbb{R}$.

If U and V are sets, then a $U \times V$ *matrix* is a matrix whose rows are indexed by the elements of U and whose columns are indexed by the elements of V. Generally, when using this terminology, the order of the rows or columns is irrelevant. For a $U \times V$ matrix M and $u \in U$, $v \in V$, the entry in position u, v is denoted by $M_{u,v}$. The all-one $U \times V$ matrix is denoted by $J_{U \times V}$, or just by J.

The *tensor product* of vectors $x \in \mathbb{R}^U$ and $y \in \mathbb{R}^V$ is the vector $x \circ y$ in $\mathbb{R}^{U \times V}$ defined by:

$$(x \circ y)_{(u,v)} := x_u y_v \tag{2.16}$$

for $u \in U$ and $v \in V$.

The *tensor product* of a $W \times X$ matrix M and a $Y \times Z$ matrix N (where W, X, Y, Z are sets), is the $(W \times Y) \times (X \times Z)$ matrix $M \circ N$ defined by

$$(M \circ N)_{(w,y),(x,z)} := M_{w,x} N_{y,z} \tag{2.17}$$

for $w \in W$, $x \in X$, $y \in Y$, $z \in Z$.

The $\mathcal{C} \times V$ *incidence matrix* of a collection or family $\mathcal{C}$ of subsets of a set V is the $\mathcal{C} \times V$ matrix M with $M_{C,v} := 1$ if $v \in C$ and $M_{C,v} := 0$ if $v \notin C$ (for $C \in \mathcal{C}$, $v \in V$). Similarly, the $V \times \mathcal{C}$ incidence matrix is the transpose of this matrix.

For any function $w : V \to \mathbb{R}$ and any $U \subseteq V$, we denote

$$w(U) := \sum_{v \in U} w(v). \tag{2.18}$$

If U is a family, we take multiplicities into account (so if v occurs k times in U, $w(v)$ occurs k times in sum (2.18)).

If w is introduced as a 'weight function', then $w(v)$ is called the *weight* of v, and for any $U \subseteq V$, $w(U)$ is called the *weight* of U. Moreover, for any $x : V \to \mathbb{R}$, we call $w^{\mathsf{T}}x$ the *weight* of x. If confusion may arise, we call $w(U)$ and $w^{\mathsf{T}}x$ the *w-weight* of U and x, respectively.

The adjective 'weight' to 'function' has no mathematical meaning, and implies no restriction, but is just introduced to enable referring to $w(v)$ or $w(U)$ as the weight of v or U. Similarly, for 'length function', 'cost function', 'profit function', 'capacity function', 'demand function', etc., leading to the *length*, *cost*, *profit*, *capacity*, *demand*, etc. of elements or of subsets. Obviously, *shortest* and *longest* are synonyms for 'minimum-length' and 'maximum-length'.

A *permutation matrix* is a square $\{0,1\}$ matrix, with exactly one 1 in each row and in each column.

Vectors $x_1, \dots, x_k$ are called *affinely independent* if there do not exist $\lambda_1, \dots, \lambda_k \in \mathbb{R}$ such that $\lambda_1 x_1 + \cdots + \lambda_k x_k = \mathbf{0}$ and $\lambda_1 + \cdots + \lambda_k = 0$ and such that the λ_i are not all equal to 0.

Vectors $x_1, \dots, x_k$ are called *linearly independent* if there do not exist $\lambda_1, \dots, \lambda_k \in \mathbb{R}$ such that $\lambda_1 x_1 + \cdots + \lambda_k x_k = \mathbf{0}$ and such that the λ_i are not all equal to 0. The linear hull of a set X is denoted by lin.hull$X$ or lin.hull(X).

If X and Y are subsets of a linear space L over a field $\mathbb{F}$, $z \in L$, and $\lambda \in \mathbb{F}$, then

$$z + X := \{z + x \mid x \in X\},\ X + Y := \{x + y \mid x \in X, y \in Y\}, \text{ and } \lambda X = \{\lambda x \mid x \in X\}. \tag{2.19}$$

If X and Y are subspaces of L, then

$$X/Y := \{x + Y \mid x \in X\} \tag{2.20}$$

is a *quotient space*, which is again a linear space, with addition and scalar multiplication given by (2.19). The dimension of X/Y is equal to $\dim(X) - \dim(X \cap Y)$.

A function $f : X \to Y$ is called an *injection* or an *injective function* if it is one-to-one: if $x, x' \in X$ and $x \neq x'$, then $f(x) \neq f(x')$. The function f is a *surjection* if it is onto: for each $y \in Y$ there is an $x \in X$ with $f(x) = y$. It is a *bijection* if it is both an injection and a surjection.

For a vector $x = (x_1, \dots, x_n)^{\mathsf{T}} \in \mathbb{R}^n$, we denote

$$\lfloor x \rfloor := (\lfloor x_1 \rfloor, \dots, \lfloor x_n \rfloor)^{\mathsf{T}} \text{ and } \lceil x \rceil := (\lceil x_1 \rceil, \dots, \lceil x_n \rceil)^{\mathsf{T}}. \tag{2.21}$$

If $f, g : X \to \mathbb{R}$ are functions, we say that $f(x)$ is $O(g(x))$, in notation

$$f(x) = O(g(x)) \text{ or } O(f(x)) = O(g(x)), \tag{2.22}$$

if there exists a constant $c \geq 0$ with $f(x) \leq cg(x) + c$ for all $x \in X$. Hence the relation $=$ given in (2.22) is transitive, but not symmetric. We put

$$g(x) = \Omega(f(x)) \tag{2.23}$$

if $f(x) = O(g(x))$.

2.5. Maxima, minima, and infinity

In this book, when speaking of a maximum or minimum, we often implicitly assume that the optimum is finite. If the optimum is not finite, consistency in min-max relations usually can be obtained by setting a minimum over the empty set to $+\infty$, a maximum over a set without upper bound to $+\infty$, a maximum over the empty set to 0 or $-\infty$ (depending on what is the universe), and a minimum over a set without lower bound to $-\infty$. This usually leads to trivial, or earlier proved, statements.

When we speak of making a value infinite, usually large enough will suffice.

If we consider maximizing a function $f(x)$ over $x \in X$, we call any $x \in X$ a *feasible solution*, and any $x \in X$ maximizing $f(x)$ an *optimum solution*. Similarly for minimizing.

2.6. Fekete's lemma

We will need the following result called Fekete's lemma, due to Pólya and Szegő [1925] (motivated by a special case proved by Fekete [1923]):

Theorem 2.2 (Fekete's lemma)**.** *Let $a_1, a_2, \dots$ be a sequence of reals such that $a_{n+m} \geq a_n + a_m$ for all positive $n, m \in \mathbb{Z}$. Then*

$$\lim_{n\to\infty} \frac{a_n}{n} = \sup_n \frac{a_n}{n}. \tag{2.24}$$

Proof. For all $i, j, k \geq 1$ we have $a_{jk+i} \geq ja_k + a_i$, by the inequality prescribed in the theorem. Hence for all fixed $i, k \geq 1$ we have

$$\begin{aligned} \liminf_{j\to\infty} \frac{a_{jk+i}}{jk+i} &\geq \liminf_{j\to\infty} \frac{ja_k + a_i}{jk+i} = \liminf_{j\to\infty} \left(\frac{a_k}{k}\frac{jk}{jk+i} + \frac{a_i}{jk+i}\right) \\ &= \frac{a_k}{k}. \end{aligned} \tag{2.25}$$

As this is true for each i, we have for each fixed $k \geq 1$:

$$\liminf_{n\to\infty} \frac{a_n}{n} = \inf_{i=1,\dots,k} \liminf_{j\to\infty} \frac{a_{jk+i}}{jk+i} \geq \frac{a_k}{k}. \tag{2.26}$$

So

$$\liminf_{n\to\infty} \frac{a_n}{n} \geq \sup_k \frac{a_k}{k}, \tag{2.27}$$

implying (2.24). ∎

We sometimes use the multiplicative version of Fekete's lemma:

Corollary 2.2a. *Let $a_1, a_2, \dots$ be a sequence of positive reals such that $a_{n+m} \geq a_n a_m$ for all positive $n, m \in \mathbb{Z}$. Then*

$$\lim_{n\to\infty} \sqrt[n]{a_n} = \sup_n \sqrt[n]{a_n}. \tag{2.28}$$

Proof. Directly from Theorem 2.2 applied to the sequence $\log a_1, \log a_2, \ldots$. ∎

Chapter 3

Preliminaries on graphs

This chapter is not meant as a rush course in graph theory, but rather as a reference guide and to settle notation and terminology.
To promote readability of the book, nonstandard notation and terminology will be, besides below in this chapter, also explained on the spot in later chapters.

3.1. Undirected graphs

A *graph* or *undirected graph* is a pair $G = (V, E)$, where V is a finite set and E is a family of *unordered* pairs from V. The elements of V are called the *vertices*, sometimes the *nodes* or the *points*. The elements of E are called the *edges*, sometimes the *lines*. We use the following shorthand notation for edges:

$$uv := \{u, v\}. \tag{3.1}$$

We denote

$$\begin{aligned} VG &:= \text{set of vertices of } G, \\ EG &:= \text{family of edges of } G. \end{aligned} \tag{3.2}$$

In running time estimates of algorithms, we denote:

$$n := |VG| \text{ and } m := |EG|. \tag{3.3}$$

In the definition of graph we use the term 'family' rather than 'set', to indicate that the same pair of vertices may occur several times in E. A pair occurring more than once in E is called a *multiple* edge, and the number of times it occurs is called its *multiplicity*. Two edges are called *parallel* if they are represented by the same pair of vertices. A *parallel class* is a maximal set of pairwise parallel edges.

So distinct edges may be represented in E by the same pair of vertices. Nevertheless, we will often speak of 'an edge uv' or even of 'the edge uv', where 'an edge of type uv' would be more correct.

Also *loops* are allowed: edges that are families of the form $\{v, v\}$. Graphs without loops and multiple edges are called *simple*, and graphs without loops are called *loopless*. A vertex v is called a *loopless vertex* if $\{v, v\}$ is not a loop.

An edge uv is said to *connect* u and v. The vertices u and v are called the *ends* of the edge uv. If there exists an edge connecting vertices u and v, then u and v are called *adjacent* or *connected*, and v is called a *neighbour* of u. The edge uv is said to be *incident* with, or to *meet*, or to *cover*, the vertices u and v, and conversely. The edges e and f are said to be *incident*, or to *meet*, or to *intersect*, if they have a vertex in common. Otherwise, they are called *disjoint*.

If $U \subseteq V$ and both ends of an edge e belong to U, then we say that U *spans* e. If at least one end of e belongs to U, then U is said to be *incident with* e. An edge connecting a vertex in a set S and a vertex in a set T is said to *connect* S and T. A set F of edges is said to *cover* a vertex v if v is covered by at least one edge in F, and to *miss* v otherwise.

For a vertex v, we denote:

$$\begin{aligned} &\delta_G(v) := \delta_E(v) := \delta(v) := \text{family of edges incident with } v, \\ &N_G(v) := N_E(v) := N(v) := \text{set of neighbours of } v. \end{aligned} \tag{3.4}$$

Here and below, notation with the subscript deleted is used if the graph is clear from the context. We speak in the definition of $\delta(v)$ of the *family* of edges incident with v, since any loop at v occurs twice in $\delta(v)$.

The *degree* $\deg_G(v)$ of a vertex v is the number of edges incident with v. In notation,

$$\deg_G(v) := \deg_E(v) := \deg(v) := |\delta_G(v)|. \tag{3.5}$$

A vertex of degree 0 is called *isolated*, and a vertex of degree 1 an *end vertex*. A vertex of degree k is called *k-valent*. So isolated vertices are loopless.

We denote

$$\begin{aligned} &\Delta(G) := \text{maximum degree of the vertices of } G, \\ &\delta(G) := \text{minimum degree of the vertices of } G. \end{aligned} \tag{3.6}$$

$\Delta(G)$ and $\delta(G)$ are called the *maximum degree* and *minimum degree* of G, respectively.

If $\Delta(G) = \delta(G)$, that is, if all degrees are equal, G is called *regular*. If all degrees are equal to k, the graph is called *k-regular*. A 3-regular graph is also called a *cubic graph*.

If $G = (V, E)$ and $G' = (V', E')$ are graphs, we denote by $G + G'$ the graph

$$G + G' := (V \cup V', E \cup E') \tag{3.7}$$

where $E \cup E'$ is the union of E and E' as families (taking multiplicities into account).

Complementary, complete, and line graph

The *complementary graph* or *complement* of a graph $G = (V, E)$ is the simple graph with vertex set V and edges all pairs of distinct vertices that are nonadjacent in G. In notation,

(3.8) $\overline{G} :=$ the complementary graph of G.

So if G is simple, then $\overline{\overline{G}} = G$.

A graph G is called *complete* if G is simple and any two distinct vertices are adjacent. In notation,

(3.9) $K_n :=$ complete graph with n vertices.

As K_n is unique up to isomorphism, we often speak of *the* complete graph on n vertices.

The *line graph* of a graph $G = (V, E)$ is the simple graph with vertex set E, where two elements of E are adjacent if and only if they meet. In notation,

(3.10) $L(G) :=$ the line graph of G.

Subgraphs

A graph $G' = (V', E')$ is called a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. If H is a subgraph of G, we say that G *contains* H. If $G' \neq G$, then G' is called a *proper subgraph* of G. If $V' = V$, then G' is called a *spanning subgraph* of G. If E' consists of all edges of G spanned by V', G' is called an *induced subgraph*, or the *subgraph induced by* V'. In notation,

(3.11) $G[V'] :=$ subgraph of G induced by V',
$E[V'] :=$ family of edges spanned by V'.

So $G[V'] = (V', E[V'])$. We further denote for any graph $G = (V, E)$ and for any vertex v, any subset U of V, any edge e, and any subset F of E,

(3.12) $G - v := G[V \setminus \{v\}]$, $G - U := G[V \setminus U]$, $G - e := (V, E \setminus \{e\})$,
$G - F := (V, E \setminus F)$.

We say that these graphs arise from G by *deleting* v, U, e, or F. (We realize that, since an edge e is a set of two vertices, the notation $G - e$ might be ambiguous (if we would consider $U := e$). We expect, however, that the appropriate interpretation will be clear from the context.)

Two subgraphs of G are called *edge-disjoint* if they have no edge in common, and *vertex-disjoint* or *disjoint*, if they have no vertex in common.

In many cases we deal with graphs *up to isomorphism*. For instance, if G and H are graphs, we say that a subgraph G' of G is *an* H *subgraph* if G' is isomorphic to H.

Paths and circuits

A *walk* in an undirected graph $G = (V, E)$ is a sequence

$$P = (v_0, e_1, v_1, \ldots, e_k, v_k), \tag{3.13}$$

where $k \geq 0$, $v_0, v_1, \ldots, v_k$ are vertices, and e_i is an edge connecting v_{i-1} and v_i (for $i = 1, \ldots, k$). If $v_0, v_1, \ldots, v_k$ are all distinct, the walk is called a *path*. (Hence $e_1, \ldots, e_k$ are distinct.)

The vertex v_0 is called the *starting vertex* or *first vertex* of P and the vertex v_k the *end vertex* or *last vertex* of P. Sometimes, both v_0 and v_k are called the *end vertices*, or just the *ends* of P. Similarly, edge e_1 is called the *starting edge* or *first edge* of P, and edge e_k the *end edge* or *last edge* of P. Sometimes, both e_1 and e_k are called the *end edges*.

The walk P is said to *connect* v_0 and v_k, to *run from* v_0 *to* v_k (or *between* v_0 *and* v_k), and to *traverse* $v_0, e_1, v_1, \ldots, e_k, v_k$. The vertices $v_1, \ldots, v_{k-1}$ are called the *internal vertices* of P. For $s, t \in V$, the walk P is called an $s - t$ *walk* if it runs from s to t, and for $S, T \subseteq V$, it is called an $S - T$ *walk* if it runs from a vertex in S to a vertex in T. Similarly, $s - T$ *walks* and $S - t$ *walks* run form s to a vertex in T and from a vertex in S to t, respectively.

The number k is called the *length* of P. (We deviate from this in case a function $l : E \to \mathbb{R}$ has been introduced as a length function. Then the *length* of P is equal to $l(e_1) + \cdots + l(e_k)$.) A walk is called *odd* (*even*, respectively) if its length is odd (even, respectively).

The minimum length of a path connecting u and v is called the *distance* of u and v. The maximum distance over all vertices u, v of G is called the *diameter* of G.

The *reverse walk* P^{-1} of P is the walk obtained from (3.13) by reversing the order of the elements:

$$P^{-1} := (v_k, e_k, v_{k-1}, \ldots, e_1, v_0). \tag{3.14}$$

If $P = (v_0, e_1, v_1, \ldots, e_k, v_k)$ and $Q = (u_0, f_1, u_1, \ldots, f_l, u_l)$ are walks satisfying $u_0 = v_k$, the *concatenation* PQ of P and Q is the walk

$$PQ := (v_0, e_1, v_1, \ldots, e_k, v_k, f_1, u_1, \ldots, f_l, u_l). \tag{3.15}$$

For any walk P, we denote by VP and EP the families of vertices and edges, respectively, occurring in P:

$$VP := \{v_0, v_1, \ldots, v_k\} \text{ and } EP := \{e_1, \ldots, e_k\}. \tag{3.16}$$

A *chord* of P is an edge of G that is not in EP and that connects two vertices of P. The path P is called *chordless* if P has no chords.

If no confusion may arise, we sometimes identify the walk P with the subgraph (VP, EP) of G, or with the set VP of vertices in P, or with the family EP of edges in P. If the graph is simple or if the edges traversed are irrelevant, we indicate the walk just by the sequence of vertices traversed:

$$P = (v_0, v_1, \ldots, v_k) \text{ or } P = v_0, v_1, \ldots, v_k. \tag{3.17}$$

A simple path may be identified by the sequence of edges:

(3.18) $P = (e_1, \dots, e_k)$ or $P = e_1, \dots, e_k$.

We denote

(3.19) $P_n :=$ a path with n vertices,

usually considered as the *graph* (VP_n, EP_n). This graph is unique up to isomorphism.

Two walks P and Q are called *vertex-disjoint* or *disjoint* if VP and VQ are disjoint, *internally vertex-disjoint* or *internally disjoint* if the set of internal vertices of P is disjoint from the set of internal vertices of Q, and *edge-disjoint* if EP and EQ are disjoint.

The walk P in (3.13) is called *closed* if $v_k = v_0$. It is called a *circuit* if $v_k = v_0$, $k \geq 1$, $v_1, \dots, v_k$ are all distinct, and $e_1, \dots, e_k$ are all distinct.

The circuit is also called a *k-circuit*. If $k = 1$, then e_1 must be a loop, and if $k = 2$, e_1 and e_2 are (distinct) parallel edges. If $k = 3$, the circuit is sometimes called a *triangle*.

The above definition of chord of a walk implies that an edge e of G is a *chord* of a circuit C if e connects two vertices in VC but does not belong to EC. A *chordless circuit* is a circuit without chords.

We denote

(3.20) $C_n :=$ a circuit with n edges,

usually considered as the *graph* (VC_n, EC_n). Again, this graph is unique up to isomorphism.

For any graph $G = (V, E)$, a subset F of E is called a *cycle* if each degree of the subgraph (V, F) is even. One may check that for any $F \subseteq E$:

(3.21) F is a cycle $\iff$ F is the symmetric difference of the edge sets of a number of circuits.

Connectivity and components

A graph $G = (V, E)$ is *connected* if for any two vertices u and v there is a path connecting u and v. A maximal connected nonempty subgraph of G is called a *connected component*, or just a *component*, of G. Here 'maximal' is taken with respect to taking subgraphs. Each component is an induced subgraph, and each vertex and each edge of G belong to exactly one component.

We often identify a component K with the set VK of its vertices. Then the components are precisely the equivalence classes of the equivalence relation $\sim$ on V defined by: $u \sim v \iff$ there exists a path connecting u and v.

A component is called *odd* (*even*) if it has an odd (even) number of vertices.

Cuts

Let $G = (V, E)$ be a graph. For any $U \subseteq V$, we denote

(3.22) $\delta_G(U) := \delta_E(U) := \delta(U) :=$ set of edges of G connecting U and $V \setminus U$.

A subset F of E is called a *cut*, if $F = \delta(U)$ for some $U \subseteq V$. In particular, $\emptyset$ is a cut. If $\emptyset \neq U \neq V$, then $\delta(U)$ is called a *nontrivial cut*. (So $\emptyset$ is a nontrivial cut if and only if G is disconnected.) It is important to observe that for any two sets $T, U \subseteq V$:

(3.23) $\delta(T) \triangle \delta(U) = \delta(T \triangle U)$.

Hence the collection of cuts is closed under taking symmetric differences.

If $s \in U$ and $t \notin U$, then $\delta(U)$ is called an $s - t$ *cut*. If $S \subseteq U$ and $T \subseteq V \setminus U$, $\delta(U)$ is called an $S - T$ *cut*. An edge-cut of size k is called a k*-cut*.

A subset F of E is called a *disconnecting edge set* if $G - F$ is disconnected. For $s, t \in V$, if F intersects each $s - t$ path, then F is said to *disconnect* or to *separate* s and t, or to be $s - t$ *disconnecting* or $s - t$ *separating*. For $S, T \subseteq V$, if F intersects each $S - T$ path, then F is said to *disconnect* or to *separate* S and T, or to be $S - T$ *disconnecting* or $S - T$ *separating*.

One may easily check that for all $s, t \in V$:

(3.24) each $s - t$ cut is $s - t$ disconnecting; each inclusionwise minimal $s - t$ disconnecting edge set is an $s - t$ cut.

An edge e of G is called a *bridge* if $\{e\}$ is a cut. A graph having no bridges is called *bridgeless*.

For any subset U of V we denote

(3.25) $d_G(U) := d_E(U) := d(U) := |\delta(U)|$.

Moreover, for subsets U, W of V:

(3.26) $E[U, W] := \{e \in E \mid \exists u \in U, w \in W : e = uw\}$.

The following is straightforward and very useful:

Theorem 3.1. *For all $U, W \subseteq V$:*

(3.27) $d(U) + d(W) = d(U \cap W) + d(U \cup W) + 2|E[U \setminus W, W \setminus U]|$.

Proof. Directly by counting edges. ∎

This in particular gives:

Corollary 3.1a. *For all $U, W \subseteq V$:*

(3.28) $d(U) + d(W) \geq d(U \cap W) + d(U \cup W)$.

Proof. Directly from Theorem 3.1. ∎

A cut of the form $\delta(v)$ for some vertex v is called a *star*.

Neighbours and vertex-cuts

Let $G = (V, E)$ be a graph. For any $U \subseteq V$, we call a vertex v a *neighbour* of U if $v \notin U$ and v has a neighbour in U. We denote

$$N_G(U) := N_E(U) := N(U) := \text{set of neighbours of } U. \tag{3.29}$$

We further denote

$$N^2(v) := N(N(v)) \setminus \{v\}. \tag{3.30}$$

A subset U of V is called a *disconnecting vertex set*, or a *vertex-cut*, if $G - U$ is disconnected. A vertex-cut of size k is called a *k-vertex-cut*. A *cut vertex* is a vertex v of G for which $G - v$ has more components than G has.

For $s, t \in V$, if U intersects each $s - t$ path, then U is said to *disconnect* s and t, or called *$s - t$ disconnecting*. If moreover $s, t \notin U$, then U is said to *separate* s and t, or called *$s - t$ separating*, or an *$s - t$ vertex-cut*. It can be shown that if U is an inclusionwise minimal $s - t$ vertex-cut, then $U = N(K)$ for the component K of $G - U$ that contains s.

For $S, T \subseteq V$, if U intersects each $S - T$ path, then U is said to *disconnect* S and T, or called *$S - T$ disconnecting*. If moreover U is disjoint from $S \cup T$, then U is said to *separate* S and T, or called *$S - T$ separating* or an *$S - T$ vertex-cut*.

A pair of subgraphs (V_1, E_1), (V_2, E_2) of a graph $G = (V, E)$ is called a *separation* if $V_1 \cup V_2 = V$ and $E_1 \cup E_2 = E$. So G has no edge connecting $V_1 \setminus V_2$ and $V_2 \setminus V_1$. Therefore, if these sets are nonempty, $V_1 \cap V_2$ is a vertex-cut of G.

Trees and forests

A graph is called a *forest* if it has no circuits. For any forest (V, E),

$$|E| = |V| - \kappa, \tag{3.31}$$

where κ is the number of components of (V, F). A *tree* is a connected forest. So for any tree (V, E),

$$|E| = |V| - 1. \tag{3.32}$$

Any forest with at least one edge has an end vertex. A connected subgraph of a tree T is called a *subtree* of T.

The notions of forest and tree extend to subsets of edges of a graph $G = (V, E)$ as follows. A subset F of E is called a *forest* if (V, F) is a forest, and a *spanning tree* if (V, F) is a tree. Then for any graph $G = (V, E)$:

$$G \text{ has a spanning tree} \iff G \text{ is connected.} \tag{3.33}$$

For any connected graph $G = (V, E)$ and any $F \subseteq E$:

(3.34) F is a spanning tree $\iff$ F is an inclusionwise maximal forest $\iff$ F is an inclusionwise minimal edge set with (V, F) connected.

Cliques, stable sets, matchings, vertex covers, edge covers

Let $G = (V, E)$ be a graph. A subset C of V is called a *clique* if any two vertices in V are adjacent, a *stable set* if any two vertices in C are nonadjacent, and a *vertex cover* if C intersects each edge of G.

A subset M of E is called a *matching* if any two edges in M are disjoint, an *edge cover* if each vertex of G is covered by at least one edge in M, and a *perfect matching* if it is both a matching and an edge cover. So a perfect matching M satisfies $|M| = \frac{1}{2}|V|$.

We denote and define:

(3.35) $\omega(G) :=$ *clique number* of $G :=$ maximum size of a clique in G,
$\alpha(G) :=$ *stable set number* of $G :=$ maximum size of a stable set in G,
$\tau(G) :=$ *vertex cover number* of $G :=$ minimum size of a vertex cover in G,
$\nu(G) :=$ *matching number* of $G :=$ maximum size of a matching in G,
$\rho(G) :=$ *edge cover number* of $G :=$ minimum size of an edge cover in G.

(We will recall this notation where used.)

Given a matching M in a graph $G = (V, E)$, we will say that a vertex u is *matched to* a vertex v, or u is the *mate* of v, if $uv \in M$. A subset U of V is called *matchable* if the subgraph $G[U]$ of G induced by U has a perfect matching.

Colouring

A *vertex-colouring*, or just a *colouring*, is a partition of V into stable sets. We sometimes consider a colouring as a function $\phi : V \to \{1, \dots, k\}$ such that $\phi^{-1}(i)$ is a stable set for each $i = 1, \dots, k$.

Each of the stable sets in a colouring is called a *colour* of the colouring. The *vertex-colouring number*, or just the *colouring number*, is the minimum number of colours in a vertex-colouring. In notation,

(3.36) $\chi(G) :=$ vertex-colouring number of G.

A graph G is called *k-colourable*, or *k-vertex-colourable*, if $\chi(G) \leq k$, and *k-chromatic* if $\chi(G) = k$. A vertex-colouring is called a *minimum vertex-colouring*, or a *minimum colouring*, if it uses the minimum number of colours.

Similar terminology holds for edge-colouring. An *edge-colouring* is a partition of E into matchings. Each of these matchings is called a *colour* of the edge-colouring. An edge-colouring can also be described by a function $\phi : E \to \{1, \dots, k\}$ such that $\phi^{-1}(i)$ is a matching for each $i = 1, \dots, k$.

The *edge-colouring number* is the minimum number of colours in an edge-colouring. In notation,

(3.37) $\chi'(G) :=$ edge-colouring number of G.

So $\chi'(G) = \chi(L(G))$.

A graph G is called *k-edge-colourable* if $\chi'(G) \leq k$, and *k-edge-chromatic* if $\chi'(G) = k$. An edge-colouring is called a *minimum edge-colouring* if it uses the minimum number of colours.

Bipartite graphs

A graph $G = (V, E)$ is called *bipartite* if $\chi(G) \leq 2$. Equivalently, G is bipartite if V can be partitioned into two sets U and W such that each edge of G connects U and W. We call the sets U and W the *colour classes* of G (although they generally need not be unique).

Bipartite graphs are characterized by:

(3.38) G is bipartite $\iff$ each circuit of G has even length.

A graph $G = (V, E)$ is called a *complete bipartite graph* if G is simple and V can be partitioned into sets U and W such that E consists of all pairs $\{u, w\}$ with $u \in U$ and $w \in W$. If $|U| = m$ and $|W| = n$, the graph is denoted by $K_{m,n}$:

(3.39) $K_{m,n} :=$ the complete bipartite graph with colour classes of size m and n.

The graphs $K_{1,n}$ are called *stars* or (when $n \geq 3$) *claws*.

Hamiltonian and Eulerian graphs

A *Hamiltonian circuit* in a graph G is a circuit C satisfying $VC = VG$. A graph is *Hamiltonian* if it has a Hamiltonian circuit. A *Hamiltonian path* is a path P with $VP = VG$.

A walk P is called *Eulerian* if each edge of G is traversed exactly once by P. A graph G is called *Eulerian* if it has a closed Eulerian walk. The following is usually attributed to Euler [1736] (although he only proved the 'only if' part):

(3.40) a graph $G = (V, E)$ without isolated vertices is Eulerian if and only if G is connected and all degrees of G are even.

Sometimes, we call a graph Eulerian if all degrees are even, ignoring connectivity. This will be clear from the context.

Contraction and minors

Let $G = (V, E)$ be a graph and let $e = uv \in E$. *Contracting* e means deleting e and identifying u and v. We denote (for $F \subseteq E$):

(3.41) $G/e :=$ graph obtained from G by contracting e,
$G/F :=$ graph obtained from G by contracting all edges in F.

The *image* of a vertex v of G in G/F is the vertex of G/F to which v is contracted.

A graph H is called a *minor* of a graph G if H arises from G by a series of deletions and contractions of edges and deletions of vertices. A minor H of G is called a *proper minor* if $H \neq G$. If G and H are graphs, we say that a minor G' of G is *an H minor* of G if G' is isomorphic to H.

Related is the following contraction. Let $G = (V, E)$ be a graph and let $S \subseteq V$. The graph G/S (obtained by *contracting* S) is obtained by identifying all vertices in S to one new vertex, called S, deleting all edges contained in S, and redefining any edge uv with $u \in S$ and $v \notin S$ to Sv.

Homeomorphic graphs

A graph G is called a *subdivision* of a graph H if G arises from H by replacing edges by paths of length at least 1. So it arises from H by iteratively choosing an edge uv, introducing a new vertex w, deleting edge uv, and adding edges uw and wv. If G is a subdivision of H, we call G an *H-subdivision*.

Two graphs G and G' are called *homeomorphic* if there exists a graph H such that both G and G' are subdivisions of H. The graph G is called a *homeomorph* of G' if G and G' are homeomorphic.

Homeomorphism can be described topologically. For any graph $G = (V, E)$, the *topological graph* $|G|$ associated with G is the topological space consisting of V and for each edge e of G a curve $|e|$ connecting the ends of e, such that for any two edges e, f one has $|e| \cap |f| = e \cap f$. Then

(3.42) G and H are homeomorphic graphs $\iff$ $|G|$ and $|H|$ are homeomorphic topological spaces.

Planarity

An *embedding* of a graph G in a topological space S is an embedding (continuous injection) of the topological graph $|G|$ in S. A graph G is called *planar* if it has an embedding in the plane $\mathbb{R}^2$.

Often, when dealing with a planar graph G, we assume that it *is* embedded in the plane $\mathbb{R}^2$. The topological components of $\mathbb{R}^2 \setminus |G|$ are called the *faces* of G. A vertex or edge is said to be *incident* with a face F if it is contained

in the boundary of F. Two faces are called *adjacent* if they are incident with some common edge.

There is a unique *unbounded face*, all other faces are *bounded*. The boundary of the unbounded face is part of $|G|$, and is called the *outer boundary* of G.

Euler's formula states that any connected planar graph $G = (V, E)$, with face collection $\mathcal{F}$, satisfies:

$$|V| + |\mathcal{F}| = |E| + 2. \tag{3.43}$$

Kuratowski [1930] found the following characterization of planar graphs:

Theorem 3.2 (Kuratowski's theorem)**.** *A graph G is planar $\iff$ no subgraph of G is homeomorphic to K_5 or to $K_{3,3}$.*

(See Thomassen [1981b] for three short proofs, and for history and references to other proofs.)

As Wagner [1937a] noticed, the following is an immediate consequence of Kuratowski's theorem (since planarity is maintained under taking minors, and since any graph without K_5 minor has no subgraph homeomorphic to K_5):

$$\text{A graph } G \text{ is planar} \iff G \text{ has no } K_5 \text{ or } K_{3,3} \text{ minor.} \tag{3.44}$$

(In turn, with a little more work, this equivalence can be shown to imply Kuratowski's theorem.)

The *four-colour theorem* of Appel and Haken [1977] and Appel, Haken, and Koch [1977] states that each loopless planar graph is 4-colourable. (Robertson, Sanders, Seymour, and Thomas [1997] gave a shorter proof.)

Tait [1878b] showed that the four-colour theorem is equivalent to: each cubic bridgeless planar graph is 3-edge-colourable. Petersen [1898] gave the example of the now-called *Petersen graph* (Figure 3.1), to show that not every bridgeless cubic graph is 3-edge-colourable. (This graph was also given by Kempe [1886], for a different purpose.)

Wagner's theorem

We will use occasionally an extension of Kuratowski's theorem, proved by Wagner [1937a]. For this we need the notion of a k-sum of graphs.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs and let $k := |V_1 \cap V_2|$. Suppose that $(V_1 \cap V_2, E_1 \cap E_2)$ is a (simple) complete graph. Then the graph

$$(V_1 \cup V_2, E_1 \triangle E_2) \tag{3.45}$$

is called a *k-sum* of G_1 and G_2. We allow multiple edges, so the k-sum might keep edges spanned by $V_1 \cap V_2$.

To formulate Wagner's theorem, we also need the graph denoted by V_8, given in Figure 3.2.

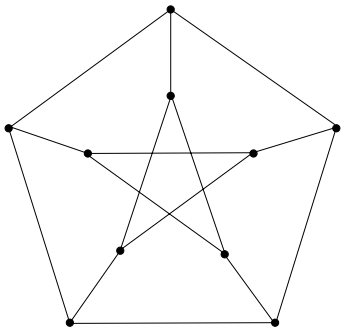

Figure 3.1
The Petersen graph

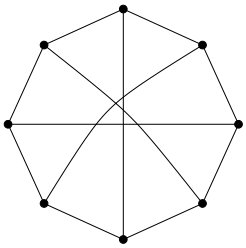

Figure 3.2
V_8

Theorem 3.3 (Wagner's theorem)**.** *A graph G has no K_5 minor $\iff$ G can be obtained from planar graphs and from copies of V_8 by taking* 1*-,* 2*-, and* 3*-sums.*

As Wagner [1937a] pointed out, this theorem implies that the four-colour theorem is equivalent to: each graph without K_5 minor is 4-colourable. This follows from the fact that k-colourability is maintained under taking k'-sums for all $k' \leq k$.

The dual graph

The *dual* G^* of an embedded planar graph $G = (V, E)$ is the graph having as vertex set the set of faces of G and having, for each $e \in E$, an edge e^* connecting the two faces incident with e. Then G^* again is planar, and $(G^*)^*$ is isomorphic to G, if G is connected. For any $C \subseteq E$, C is a circuit in G if and only if $C^* := \{e^* \mid e \in C\}$ is an inclusionwise minimal nonempty cut in

G^*. Moreover, C is a spanning tree in G if and only if $\{e^* \mid e \in E \setminus C\}$ is a spanning tree in G^*.

Series-parallel and outerplanar graphs

A graph is called a *series-parallel graph* if it arises from a forest by repeated replacing edges by parallel edges or by edges in series. It was proved by Duffin [1965] that a graph is series-parallel if and only if it has no K_4 minor.

A graph is called *outerplanar* if it can be embedded in the plane such that each vertex is on the outer boundary. It can be easily derived from Kuratowski's theorem that a graph is outerplanar if and only if it has no K_4 or $K_{2,3}$ minor.

Adjacency and incidence matrix

The *adjacency matrix* of a graph $G = (V, E)$ is the $V \times V$ matrix A with

$$A_{u,v} := \text{number of edges connecting } u \text{ and } v \tag{3.46}$$

for $u, v \in V$.

The *incidence matrix*, or $V \times E$ *incidence matrix*, of G is the $V \times E$ matrix B with

$$B_{v,e} := \begin{cases} 1 & \text{if } v \in e \text{ and } e \text{ is not a loop,} \\ 2 & \text{if } v \in e \text{ and } e \text{ is a loop,} \\ 0 & \text{if } v \notin e, \end{cases} \tag{3.47}$$

for $v \in V$ and $e \in E$. The transpose of B is called the $E \times V$ incidence matrix of G, or just the incidence matrix, if no confusion is expected.

The concepts from graph theory invite to a less formal, and more expressive language, which appeals to the intuition, and whose formalization will be often tedious rather than problematic. Thus we say 'replace the edge uv by two edges in series', which means deleting uv and introducing a new vertex, w say, and new edges uw and wv. Similarly, 'replacing the edge uv by a path' means deleting uv, and introducing new vertices $w_1, \dots, w_k$ say, and new edges uw_1, $w_1w_2, \dots, w_{k-1}w_k$, w_kv.

3.2. Directed graphs

A *directed graph* or *digraph* is a pair $D = (V, A)$ where V is a finite set and A is a family of *ordered* pairs from V. The elements of V are called the *vertices*, sometimes the *nodes* or the *points*. The elements of A are called the *arcs* (sometimes *directed edges*). We denote:

(3.48) $VD :=$ set of vertices of D and $AD :=$ family of arcs of D.

In running time estimates of algorithms we denote:

(3.49) $n := |VD|$ and $m := |AD|$.

Again, the term 'family' is used to indicate that the same pair of vertices may occur several times in A. A pair occurring more than once in A is called a *multiple* arc, and the number of times it occurs is called its *multiplicity*. Two arcs are called *parallel* if they are represented by the same ordered pair of vertices.

Also *loops* are allowed, that is, arcs of the form (v, v). In our discussions, loops in directed graphs will be almost always irrelevant, and it will be clear from the context if they may occur. Directed graphs without loops and multiple arcs are called *simple*, and directed graphs without loops are called *loopless*.

Each directed graph $D = (V, A)$ gives rise to an *underlying undirected graph*, which is the graph $G = (V, E)$ obtained by ignoring the orientation of the arcs:

(3.50) $E := \{\{u, v\} \mid (u, v) \in A\}$.

We often will transfer undirected terminology to the directed case. Where appropriate, the adjective 'undirected' is added to a term if it refers to the underlying undirected graph.

If G is the underlying undirected graph of a directed graph D, we call D an *orientation* of G.

An arc (u, v) is said to *connect* u and v, and to *run from* u *to* v. For an arc $a = (u, v)$, u and v are called the *ends* of a, and u is called the *tail* of a, and v the *head* of a. We say that $a = (u, v)$ *leaves* u and *enters* v. For $U \subseteq V$, an arc $a = (u, v)$ is said to *leave* U if $u \in U$ and $v \notin U$. It is said to *enter* U if $u \notin U$ and $v \in U$.

If there exists an arc connecting vertices u and v, then u and v are called *adjacent* or *connected*. If there exists an arc (u, v), then v is called an *outneighbour* of u, and u is called an *inneighbour* of v.

The arc (u, v) is said to be *incident* with, or to *meet*, or to *cover*, the vertices u and v, and conversely. The arcs a and b are said to be *incident*, or to *meet*, or to *intersect*, if they have a vertex in common. Otherwise, they are called *disjoint*. If $U \subseteq V$ and both ends of an arc a belong to U, then we say that U *spans* a.

For any vertex v, we denote:

(3.51)
$$\begin{aligned}
&\delta^{\mathrm{in}}_D(v) := \delta^{\mathrm{in}}_A(v) := \delta^{\mathrm{in}}(v) := \text{set of arcs entering } v,\\
&\delta^{\mathrm{out}}_D(v) := \delta^{\mathrm{out}}_A(v) := \delta^{\mathrm{out}}(v) := \text{set of arcs leaving } v,\\
&N^{\mathrm{in}}_D(v) := N^{\mathrm{in}}_A(v) := N^{\mathrm{in}}(v) := \text{set of inneighbours of } v,\\
&N^{\mathrm{out}}_D(v) := N^{\mathrm{out}}_A(v) := N^{\mathrm{out}}(v) := \text{set of outneighbours of } v.
\end{aligned}$$

The *indegree* $\deg^{\mathrm{in}}_D(v)$ of a vertex v is the number of arcs entering v. The *outdegree* $\deg^{\mathrm{out}}_D(v)$ of a vertex v is the number of arcs leaving v. In notation,

(3.52) $$\deg^{\mathrm{in}}_D(v) := \deg^{\mathrm{in}}_A(v) := \deg^{\mathrm{in}}(v) := |\delta^{\mathrm{in}}_D(v)|,$$
$$\deg^{\mathrm{out}}_D(v) := \deg^{\mathrm{out}}_A(v) := \deg^{\mathrm{out}}(v) := |\delta^{\mathrm{out}}_D(v)|.$$

A vertex of indegree 0 is called a *source* and a vertex of outdegree 0 a *sink*. For any arc $a = (u, v)$ we denote

(3.53) $$a^{-1} := (v, u).$$

For any digraph $D = (V, A)$ the *reverse digraph* D^{-1} is defined by

(3.54) $$D^{-1} = (V, A^{-1}), \text{ where } A^{-1} := \{a^{-1} \mid a \in A\}.$$

A *mixed graph* is a triple (V, E, A) where (V, E) is an undirected graph and (V, A) is a directed graph.

The complete directed graph and the line digraph

The *complete directed graph* on a set V is the simple directed graph with vertex set V and arcs all pairs (u, v) with $u, v \in V$ and $u \neq v$. A *tournament* is any simple directed graph (V, A) such that for all distinct $u, v \in V$ precisely one of (u, v) and (v, u) belongs to A.

The *line digraph* of a directed graph $D = (V, A)$ is the digraph with vertex set A and arc set

(3.55) $$\{((u, v), (x, y)) \mid (u, v), (x, y) \in A, v = x\}.$$

Subgraphs of directed graphs

A digraph $D' = (V', A')$ is called a *subgraph* of a digraph $D = (V, A)$ if $V' \subseteq V$ and $A' \subseteq A$. If $D' \neq D$, then D' is called a *proper subgraph* of D. If $V' = V$, then D' is called a *spanning subgraph* of D. If A' consists of all arcs of D spanned by V', D' is called an *induced subgraph*, or the *subgraph induced by* V'. In notation,

(3.56) $$D[V'] := \text{subgraph of } D \text{ induced by } V',$$
$$A[V'] := \text{family of arcs spanned by } V'.$$

So $D[V'] = (V', A[V'])$. We further denote for any vertex v, any subset U of V, any arc a, and any subset B of A,

(3.57) $$D - v := D[V \setminus \{v\}],\ D - U := D[V \setminus U],\ D - a := (V, A \setminus \{a\}),$$
$$D - B := (V, A \setminus B).$$

We say that these graphs arise from D by *deleting* v, U, a, or B.

Two subgraphs of D are called *arc-disjoint* if they have no arc in common, and *vertex-disjoint* or *disjoint*, if they have no vertex in common.

Directed paths and circuits

A *directed walk*, or just a *walk*, in a directed graph $D = (V, A)$ is a sequence

$$P = (v_0, a_1, v_1, \dots, a_k, v_k), \tag{3.58}$$

where $k \geq 0$, $v_0, v_1, \dots, v_k \in V$, $a_1, \dots, a_k \in A$, and $a_i = (v_{i-1}, v_i)$ for $i = 1, \dots, k$. The path is called a *directed path*, or just a *path*, if $v_0, \dots, v_k$ are distinct. (Hence $a_1, \dots, a_k$ are all distinct.)

The vertex v_0 is called the *starting vertex* or the *first vertex* of P, and the vertex v_k the *end vertex* or the *last vertex* of P. Sometimes, both v_0 and v_k are called the *end vertices*, or just the *ends* of P. Similarly, arc a_1 is called the *starting arc* or *first arc* of P and arc a_k the *end arc* or *last arc* of P. Sometimes, both a_1 and a_k are called the *end arcs*.

The walk P is said to *connect* the vertices v_0 and v_k, to *run from* v_0 *to* v_k (or *between* v_0 *and* v_k), and to *traverse* $v_0, a_1, v_1, \dots, a_k, v_k$. The vertices $v_1, \dots, v_{k-1}$ are called the *internal vertices* of P. For $s, t \in V$, a walk P is called an $s - t$ *walk* if it runs from s to t, and for $S, T \subseteq V$, P is called an $S - T$ *walk* if it runs from a vertex in S to a vertex in T. If P is a path, we obviously speak of an $s - t$ *path* and an $S - T$ *path*, respectively.

A vertex t is called *reachable from* a vertex s (or from a set S) if there exists a directed $s - t$ path (or directed $S - t$ path). Similarly, a vertex s is said to *reach*, or to be *reachable to*, a vertex t (or to a set T) if there exists a directed $s - t$ path (or directed $s - T$ path).

The number k in (3.58) is called the *length* of P. (We deviate from this in case a function $l : A \to \mathbb{R}$ has been introduced as a length function. Then the *length* of P is equal to $l(a_1) + \cdots + l(a_k)$.)

The minimum length of a path from u to v is called the *distance* from u to v.

An *undirected walk* in a directed graph $D = (V, A)$ is a walk in the underlying undirected graph; more precisely, it is a sequence

$$P = (v_0, a_1, v_1, \dots, a_k, v_k) \tag{3.59}$$

where $k \geq 0$, $v_0, v_1, \dots, v_k \in V$, $a_1, \dots, a_k \in A$, and $a_i = (v_{i-1}, v_i)$ or $a_i = (v_i, v_{i-1})$ for $i = 1, \dots, k$. The arcs a_i with $a_i = (v_{i-1}, v_i)$ are called the *forward arcs* of P, and the arcs a_i with $a_i = (v_i, v_{i-1})$ the *backward arcs* of P.

If $P = (v_0, a_1, v_1, \dots, a_k, v_k)$ and $Q = (u_0, b_1, u_1, \dots, b_l, u_l)$ are walks satisfying $u_0 = v_k$, the *concatenation* PQ of P and Q is the walk

$$PQ := (v_0, a_1, v_1, \dots, a_k, v_k, b_1, u_1, \dots, b_l, u_l). \tag{3.60}$$

For any walk P, we denote by VP and AP the families of vertices and arcs, respectively, occurring in P:

$$VP := \{v_0, v_1, \dots, v_k\} \text{ and } AP := \{a_1, \dots, a_k\}. \tag{3.61}$$

If no confusion may arise, we sometimes identify the walk P with the subgraph (VP, AP) of D, or with the set VP of vertices in P, or with the family AP of arcs in P.

If the digraph is simple or (more generally) if the arcs traversed are irrelevant, we indicate the walk just by the sequence of vertices traversed:

(3.62) $$P = (v_0, v_1, \dots, v_k) \text{ or } P = v_0, v_1, \dots, v_k.$$

A path may be identified by the sequence of arcs:

(3.63) $$P = (a_1, \dots, a_k) \text{ or } P = a_1, \dots, a_k.$$

Two walks P and Q are called *vertex-disjoint* or *disjoint* if VP and VQ are disjoint, *internally vertex-disjoint* or *internally disjoint* if the set of internal vertices of P is disjoint from the set of internal vertices of Q, and *arc-disjoint* if AP and AQ are disjoint.

The directed walk P in (3.13) is called a *closed directed walk* or *directed cycle* if $v_k = v_0$. It is called a *directed circuit*, or just a *circuit*, if $v_k = v_0$, $k \geq 1$, $v_1, \dots, v_k$ are all distinct, and $a_1, \dots, a_k$ are all distinct. An *undirected circuit* is a circuit in the underlying undirected graph.

Connectivity and components of digraphs

A digraph $D = (V, A)$ is called *strongly connected* if for each two vertices u and v there is a directed path from u to v. The digraph D is called *weakly connected* if the underlying undirected graph is connected; that is, for each two vertices u and v there is an undirected path connecting u and v.

A maximal strongly connected nonempty subgraph of a digraph $D = (V, A)$ is called a *strongly connected component*, or a *strong component*, of D. Again, 'maximal' is taken with respect to taking subgraphs. A *weakly connected component*, or a *weak component*, of D is a component of the underlying undirected graph.

Each strong component is an induced subgraph. Each vertex belongs to exactly one strong component, but there may be arcs that belong to no strong component. One has:

(3.64) arc (u, v) belongs to a strong component $\iff$ there exists a directed path in D from v to u.

We sometimes identify a strong component K with the set VK of its vertices. Then the strong components are precisely the equivalence classes of the equivalence relation $\sim$ defined on V by: $u \sim v \iff$ there exist a directed path from u to v and a directed path from v to u.

Cuts

Let $D = (V, A)$ be a directed graph. For any $U \subseteq V$, we denote:

(3.65) $\delta^{\mathrm{in}}_D(U) := \delta^{\mathrm{in}}_A(U) := \delta^{\mathrm{in}}(U) :=$ set of arcs of D entering U,
$\delta^{\mathrm{out}}_D(U) := \delta^{\mathrm{out}}_A(U) := \delta^{\mathrm{out}}(U) :=$ set of arcs of D leaving U.

A subset B of A is called a *cut* if $B = \delta^{\mathrm{out}}(U)$ for some $U \subseteq V$. In particular, $\emptyset$ is a cut. If $\emptyset \neq U \neq V$, then $\delta^{\mathrm{out}}(U)$ is called a *nontrivial cut*.

If $s \in U$ and $t \notin U$, then $\delta^{\mathrm{out}}(U)$ is called an $s-t$ *cut*. If $S \subseteq U$ and $T \subseteq V \setminus U$, $\delta^{\mathrm{out}}(U)$ is called an $S-T$ *cut*. A cut of size k is called a k*-cut*.

A subset B of A is called a *disconnecting arc set* if $D-B$ is not strongly connected. For $s,t \in V$, it is said to be $s-t$ *disconnecting*, if B intersects each directed $s-t$ path. For $S,T \subseteq V$, B is said to be $S-T$ *disconnecting*, if B intersects each directed $S-T$ path.

One may easily check that for all $s,t \in V$:

(3.66) each $s-t$ cut is $s-t$ disconnecting; each inclusionwise minimal $s-t$ disconnecting arc set is an $s-t$ cut.

For any subset U of V we denote

(3.67) $d^{\mathrm{in}}_D(U) := d^{\mathrm{in}}_A(U) := d^{\mathrm{in}}(U) := |\delta^{\mathrm{in}}(U)|$,
$d^{\mathrm{out}}_D(U) := d^{\mathrm{out}}_A(U) := d^{\mathrm{out}}(U) := |\delta^{\mathrm{out}}(U)|$.

The following inequalities will be often used:

Theorem 3.4. *For any digraph $D=(V,A)$ and $X,Y \subseteq V$:*

(3.68) $d^{\mathrm{in}}(X) + d^{\mathrm{in}}(Y) \geq d^{\mathrm{in}}(X \cap Y) + d^{\mathrm{in}}(X \cup Y)$ *and*
$d^{\mathrm{out}}(X) + d^{\mathrm{out}}(Y) \geq d^{\mathrm{out}}(X \cap Y) + d^{\mathrm{out}}(X \cup Y)$,

Proof. The first inequality follows directly from the equation

(3.69) $d^{\mathrm{in}}(X) + d^{\mathrm{in}}(Y) =$
$d^{\mathrm{in}}(X \cap Y) + d^{\mathrm{in}}(X \cup Y) + |A[X \setminus Y, Y \setminus X]| + |A[Y \setminus X, X \setminus Y]|$,

where $A[S,T]$ denotes the set of arcs with tail in S and head in T. The second inequality follows similarly. ∎

A cut C is called a *directed cut* if $C = \delta^{\mathrm{in}}(U)$ for some $U \subseteq V$ with $\delta^{\mathrm{out}}(U) = \emptyset$ and $\emptyset \neq U \neq V$. An arc is called a *cut arc* if $\{a\}$ is a directed cut; equivalently, if a is a bridge in the underlying undirected graph.

Vertex-cuts

Let $D=(V,A)$ be a digraph. A subset U of V is called a *disconnecting vertex set*, or a *vertex-cut*, if $D-U$ is disconnected. A vertex-cut of size k is called a k*-vertex-cut*.

For $s,t \in V$, if U intersects each directed $s-t$ path in D, then U is said to *disconnect* s and t, or called $s-t$ *disconnecting*. If moreover $s,t \notin U$, then U is said to *separate* s and t, or called $s-t$ *separating*, or an $s-t$ *vertex-cut*.

For $S, T \subseteq V$, if U intersects each directed $S-T$ path, then U is said to *disconnect* S and T, or called $S-T$ *disconnecting*. If moreover U is disjoint from $S \cup T$, then U is said to *separate* S and T, or called $S-T$ *separating* or an $S-T$ *vertex-cut*.

Acyclic digraphs and directed trees

A directed graph $D = (V, A)$ is called *acyclic* if it has no directed circuits. It is easy to show that

(3.70) an acyclic digraph has at least one source and at least one sink,

provided that it has at least one vertex.

A directed graph is called a *directed tree* if the underlying undirected graph is a tree; that is, if D is weakly connected and has no undirected circuits. It is called a *rooted tree* if moreover D has precisely one source, called the *root*. If r is the root, we say that the rooted tree is *rooted at* r. If a rooted tree $D = (V, A)$ has root r, then each vertex $v \neq r$ has indegree 1, and for each vertex v there is a unique directed $r-v$ path. An *arborescence* in a digraph $D = (V, A)$ is a set B of arcs such that (V, B) is a rooted tree. If the rooted tree has root r, it is called an *r-arborescence*.

A directed graph is called a *directed forest* if the underlying undirected graph is a forest; that is, if D has no undirected circuits. It is called a *rooted forest* if moreover each weak component is a rooted tree. The roots of the weak components are called the *roots* of the rooted forest. A *branching* in a digraph $D = (V, A)$ is a set B of arcs such that (V, B) is a rooted forest.

Hamiltonian and Eulerian digraphs

A *Hamiltonian circuit* in a directed graph $D = (V, A)$ is a directed circuit C with $VC = VD$. A digraph is *Hamiltonian* if it has a Hamiltonian circuit. A *Hamiltonian path* is a directed path P with $VP = VD$.

A directed walk P is called *Eulerian* if each arc of D is traversed exactly once by P. A digraph D is called *Eulerian* if it has a closed Eulerian directed walk. Then a digraph $D = (V, A)$ is Eulerian if and only if D is weakly connected and $\deg^{\text{in}}(v) = \deg^{\text{out}}(v)$ for each vertex v. Sometimes, we call a digraph Eulerian if each weak component is Eulerian. This will be clear from the context.

An *Eulerian orientation* of an undirected graph $G = (V, E)$ is an orientation (V, A) of G with $\deg^{\text{in}}_A(v) = \deg^{\text{out}}_A(v)$ for each $v \in V$. A classical theorem in graph theory states that an undirected graph G has an Eulerian orientation if and only if all degrees of G are even.

Contraction

Contraction of directed graphs is similar to contraction of undirected graphs. Let $D = (V, A)$ be a digraph and let $a = (u, v) \in A$. *Contracting* a means deleting a and identifying u and v. We denote:

(3.71) $D/a :=$ digraph obtained from D by contracting a.

Related is the following contraction. Let $D = (V, A)$ be a digraph and let $S \subseteq V$. The digraph D/S (obtained by *contracting* S) is obtained by identifying all vertices in S to one new vertex, called S, deleting all arcs contained in S, and redefining any arc (u, v) to (S, v) if $u \in S$ and to (u, S) if $v \in S$.

Planar digraphs and their duals

A digraph D is called *planar* if its underlying undirected graph G is planar. There is a natural way of making the dual graph G^* of G into a directed graph D^*, the *dual*: if arc $a = (u, v)$ of D separates faces F and F', such that, when following a from u to v, F is at the left and F' is at the right of a, then the dual edge is oriented from F to F', giving the arc a^* of D^*. Then D^{**} is isomorphic to D^{-1}, if D is weakly connected. One may check that a subset C of D is a directed circuit in D if and only if the set $\{a^* \mid a \in C\}$ is an inclusionwise minimal directed cut in D^*.

Adjacency and incidence matrix

The *adjacency matrix* of a digraph $D = (V, A)$ is the $V \times V$ matrix M with

(3.72) $M_{u,v} :=$ number of arcs from u to v

for $u, v \in V$.

The *incidence matrix*, or $V \times A$ *incidence matrix*, of D is the $V \times A$ matrix B with

$$B_{v,a} := \begin{cases} -1 & \text{if } v \text{ is tail of } a, \\ +1 & \text{if } v \text{ is head of } a, \\ 0 & \text{otherwise,} \end{cases} \tag{3.73}$$

for any $v \in V$ and any nonloop $a \in A$. If a is a loop, we set $B_{v,a} := 0$ for each vertex v.

The transpose of B is called the $A \times V$ incidence matrix of D, or just the incidence matrix, if no confusion is expected.

3.3. Hypergraphs

Part VIII is devoted to hypergraphs, but we occasionally need the terminology of hypergraphs in earlier parts. A *hypergraph* is a pair $H = (V, \mathcal{E})$ where V is a finite set and $\mathcal{E}$ is a family of subsets of V. The elements of V and $\mathcal{E}$ are called the *vertices* and the *edges* respectively. If $|F| = k$ for each $F \in \mathcal{E}$, the hypergraph is called *k-uniform*.

A hypergraph $H = (V, \mathcal{E})$ is called *connected* if there is no $U \subseteq V$ such that $\emptyset \neq U \neq V$ and such that $F \subseteq U$ or $F \subseteq V \setminus U$ for each edge F. A *(connected) component* of H is a hypergraph $K = (V', \mathcal{E}')$ with $V' \subseteq V$ and $\mathcal{E}' \subseteq \mathcal{E}$, such that V' and $\mathcal{E}'$ are inclusionwise maximal with the property that K is connected. A component is uniquely identified by its set of vertices.

Packing and covering

A family $\mathcal{F}$ of sets is called a *packing* if the sets in $\mathcal{F}$ are pairwise disjoint. For $k \in \mathbb{Z}_+$, $\mathcal{F}$ is called a *k-packing* if each element of $\bigcup \mathcal{F}$ is in at most k sets in $\mathcal{F}$ (counting multiplicities). In other words, any $k+1$ sets from $\mathcal{F}$ have an empty intersection. If each set in $\mathcal{F}$ is a subset of some set S, and $c : S \to \mathbb{R}$, then $\mathcal{F}$ is called a *c-packing* if each element $s \in S$ is in at most $c(s)$ sets in $\mathcal{F}$ (counting multiplicities).

A *fractional packing* is a function $\lambda : \mathcal{F} \to \mathbb{R}_+$ such that, for each $s \in S$,

$$\sum_{\substack{U \in \mathcal{F} \\ s \in U}} \lambda_U \leq 1. \tag{3.74}$$

For $c : S \to \mathbb{R}$, the function $\lambda : \mathcal{F} \to \mathbb{R}_+$ is called a *fractional c-packing* if

$$\sum_{U \in \mathcal{F}} \lambda_U \chi^U \leq c. \tag{3.75}$$

The *size* of $\lambda : \mathcal{F} \to \mathbb{R}$ is, by definition,

$$\sum_{U \in \mathcal{F}} \lambda_U. \tag{3.76}$$

Similarly, a family $\mathcal{F}$ of sets is called a *covering* of a set S if S is contained in the union of the sets in $\mathcal{F}$. For $k \in \mathbb{Z}_+$, $\mathcal{F}$ is called a *k-covering* of S if each element of S is in at least k sets in $\mathcal{F}$ (counting multiplicities). For $c : S \to \mathbb{R}$, $\mathcal{F}$ is called a *c-covering* if each element $s \in S$ is in at least $c(s)$ sets in $\mathcal{F}$ (counting multiplicities).

A *fractional covering* of S is a function $\lambda : \mathcal{F} \to \mathbb{R}_+$ such that, for each $s \in S$,

$$\sum_{\substack{U \in \mathcal{F} \\ s \in U}} \lambda_U \geq 1. \tag{3.77}$$

For $c : S \to \mathbb{R}$, the function $\lambda : \mathcal{F} \to \mathbb{R}_+$ is called a *fractional c-covering* if

$$\sum_{U \in \mathcal{F}} \lambda_U \chi^U \geq c. \tag{3.78}$$

Again, the *size* of $\lambda : \mathcal{F} \to \mathbb{R}$ is, by definition,

$$\sum_{U \in \mathcal{F}} \lambda_U. \tag{3.79}$$

Cross-free and laminar families

A collection $\mathcal{C}$ of subsets of a set V is called *cross-free* if for all $T, U \in \mathcal{C}$:

$$T \subseteq U \text{ or } U \subseteq T \text{ or } T \cap U = \emptyset \text{ or } T \cup U = V. \tag{3.80}$$

$\mathcal{C}$ is called *laminar* if for all $T, U \in \mathcal{C}$:

$$T \subseteq U \text{ or } U \subseteq T \text{ or } T \cap U = \emptyset. \tag{3.81}$$

There is the following upper bound on the size of a laminar family:

Theorem 3.5. *If $\mathcal{C}$ is laminar and $V \neq \emptyset$, then $|\mathcal{C}| \leq 2|V|$.*

Proof. By induction on $|V|$. We can assume that $|V| \geq 2$ and that $V \in \mathcal{C}$. Let U be an inclusionwise minimal set in $\mathcal{C}$ with $|U| \geq 2$. Resetting $\mathcal{C}$ to $\mathcal{C} \setminus \{\{v\} \mid v \in U\}$, and identifying all elements in U, $|\mathcal{C}|$ decreases by at most $|U|$, and $|V|$ by $|U| - 1$. Since $|U| \leq 2(|U| - 1)$ (as $|U| \geq 2$), induction gives the required inequality. ∎

3.3a. Background references on graph theory

For background on graph theory we mention the books by Kőnig [1936] (historical), Harary [1969] (classical reference book), Wilson [1972b] (introductory), Bondy and Murty [1976], and Diestel [1997].

Chapter 4

Preliminaries on algorithms and complexity

This chapter gives an introduction to algorithms and complexity, in particular to polynomial-time solvability and NP-completeness. We restrict ourselves to a largely informal outline and keep formalisms at a low level. Most of the formalisms described in this chapter are not needed in the remaining of this book. A rough understanding of algorithms and complexity suffices.

4.1. Introduction

An informal, intuitive idea of what is an algorithm will suffice to understand the greater part of this book. An algorithm can be seen as a finite set of instructions that perform operations on certain data. The input of the algorithm will give the initial data. When the algorithm stops, the output will be found in prescribed locations of the data set. The instructions need not be performed in a linear order: an instruction determines which of the instructions should be followed next. Also, it can prescribe to stop the algorithm.

While the set of instructions constituting the algorithm is finite and fixed, the size of the data set may vary, and will depend on the input. Usually, the data are stored in arrays, that is, finite sequences. The lengths of these arrays may depend on the input, but the number of arrays is fixed and depends only on the algorithm. (A more-dimensional array like a matrix is stored in a linear fashion, in accordance with the linear order in which computer memory is organized.)

The data may consist of numbers, letters, or other symbols. In a computer model they are usually stored as finite strings of 0's and 1's (*bits*). The *size* of the data is the total length of these strings. In this context, the *size* of a rational number p/q with $p, q \in \mathbb{Z}$, $q \geq 1$, and g.c.d.$(p.q) = 1$, is equal to $1 + \lceil \log(|p| + 1) \rceil + \lceil \log q \rceil$.

4.2. The random access machine

We use the algorithmic model of the *random access machine*, sometimes abbreviated to *RAM*. It operates on entries that are $0,1$ strings, representing abstract objects (like vertices of a graph) or rational numbers. An instruction can read several (but a fixed number of) entries simultaneously, perform arithmetic operations on them, and store the answers in array positions prescribed by the instruction[2]. The array positions that should be read and written, are given in locations prescribed by the instruction.

We give a more precise description. The random access machine has a finite set of variables $z_0, \ldots, z_k$ and one array, f say, of length depending on the input. Each array entry is a $0,1$ string. They can be interpreted as rationals, in some binary encoding, but can also have a different meaning. Initially, $z_0, \ldots, z_k$ are set to 0, and f contains the input.

Each instruction is a finite sequence of resettings of one the following types, for $i, j, h \in \{1, \ldots, k\}$:

$$z_i := f(z_j);\ f(z_j) := z_i;\ z_i := z_j + z_h;\ z_i := z_j - z_h;\ z_i := z_j z_h;\ z_i := z_j / z_h;\ z_i := z_i + 1;\ z_i := 1 \text{ if } z_j > 0 \text{ and } z_i := 0 \text{ otherwise.} \tag{4.1}$$

These include the *elementary arithmetic operations*: addition, subtraction, multiplication, division, comparison. (One may derive other arithmetic operations from this like rounding and taking logarithm or square root, by performing $O(\sigma + |\log \varepsilon|)$ elementary arithmetic operations, where σ is the size of the rational number and ε is the required precision.)

The instructions are numbered $0, 1, \ldots, t$, and z_1 is the number of the instruction to be executed. If $z_1 > t$ we stop and return the contents of the array f as output.

4.3. Polynomial-time solvability

A *polynomial-time algorithm* is an algorithm that terminates after a number of steps bounded by a polynomial in the input size. Here a *step* consists of performing one instruction. Such an algorithm is also called a *good algorithm* or an *efficient algorithm*.

In this definition, the *input size* is the size of the input, that is, the number of bits that describe the input. We say that a problem is *polynomial-time solvable*, or is *solvable in polynomial time*, if it can be solved by a polynomial-time algorithm.

This definition may depend on the chosen algorithmic model, but it has turned out that for most models the set of problems solvable by a polynomial-time algorithm is the same. However, in giving order estimates of running

[2] This property has caused the term 'random' in random access machine: the machine has access, in constant time, to the data in *any* (however, well-determined) position. This is in contrast with the Turing machine, which can only move to adjacent positions.

times and in considering the concept of 'strongly polynomial-time' algorithm (cf. Section 4.12), we fix the above algorithmic model of the random access machine.

4.4. P

P, NP, and co-NP are collections of *decision problems*: problems that can be answered by 'yes' or 'no', like whether a given graph has a perfect matching or a Hamiltonian circuit. An optimization problem is no decision problem, but often can be reduced to it in a certain sense — see Section 4.7 below.

A decision problem is completely described by the inputs for which the answer is 'yes'. To formalize this, fix some finite set Σ, called the *alphabet*, of size at least 2 — for instance $\{0,1\}$ or the ASCII-set of symbols. Let Σ^* denote the set of all finite strings (*words*) of letters from Σ. The *size* of a word is the number of letters (counting multiplicities) in the word. We denote the size of a word w by size(w).

As an example, an undirected graph can be represented by the word

$$(\{a,b,c,d\},\{\{a,b\},\{b,c\},\{a,d\},\{b,d\},\{a,c\}\}) \tag{4.2}$$

(assuming that Σ contains each of these symbols). Its size is 43.

A *problem* is any subset Π of Σ^*. The corresponding 'informal' problem is:

(4.3) given a word $x \in \Sigma^*$, does x belong to Π?

As an example, the problem if a given graph is Hamiltonian is formalized by the collection of all strings representing a Hamiltonian graph.

The string x is called the *input* of the problem. One speaks of an *instance* of a problem Π if one asks for one concrete input x whether x belongs to Π.

A problem Π is called *polynomial-time solvable* if there exists a polynomial-time algorithm that decides whether or not a given word $x \in \Sigma^*$ belongs to Π. The collection of all polynomial-time solvable problems $\Pi \subseteq \Sigma^*$ is denoted by P.

4.5. NP

An easy way to characterize the class NP is: NP is the collection of decision problems that can be reduced in polynomial time to the satisfiability problem — that is, to checking if a Boolean expression can be satisfied. For instance, it is not difficult to describe the conditions for a perfect matching in a graph by a Boolean expression, and hence reduce the existence of a perfect matching to the satisfiability of this expression. Also the problem of finding a Hamiltonian circuit, or a clique of given size, can be treated this way.

However, this is not the definition of NP, but a theorem of Cook. Roughly speaking, NP is defined as the collection of all decision problems for which each input with positive answer, has a polynomial-time checkable 'certificate' of correctness of the answer. Consider, for instance, the question:

(4.4) Is a given graph Hamiltonian?

A positive answer can be 'certified' by giving a Hamiltonian circuit in the graph. The correctness of it can be checked in polynomial time. No such certificate is known for the opposite question:

(4.5) Is a given graph non-Hamiltonian?

Checking the certificate in polynomial time means: checking it in time bounded by a polynomial in the original input size. In particular, it implies that the certificate itself has size bounded by a polynomial in the original input size.

This can be formalized as follows. NP is the collection of problems $\Pi \subseteq \Sigma^*$ for which there is a problem $\Pi' \in$P and a polynomial p such that for each $w \in \Sigma^*$ one has:

(4.6) $w \in \Pi \iff$ there exists a word x of size at most $p(\text{size}(w))$ with $wx \in \Pi'$.

The word x is called a *certificate* for w. (NP stands for *nondeterministically polynomial-time*, since the string x could be chosen by the algorithm by guessing. So guessing well leads to a polynomial-time algorithm.)

For instance, the collection of Hamiltonian graphs belongs to NP since the collection Π' of strings GC, consisting of a graph G and a Hamiltonian circuit C in G, belongs to P. (Here we take graphs and circuits as strings like (4.2).)

Trivially, we have P$\subseteq$NP, since if $\Pi \in$P, we can take $\Pi' = \Pi$ and $p \equiv 0$ in (4.6).

About all problems that ask for the existence of a structure of a prescribed type (like a Hamiltonian circuit) belong to NP. The class NP is apparently much larger than the class P, and there might be not much reason to believe that the two classes are the same. But, as yet, nobody has been able to prove that they really are different. This is an intriguing mathematical question, but besides, answering the question might also have practical significance. If P=NP can be shown, the proof might contain a revolutionary new algorithm, or alternatively, it might imply that the concept of 'polynomial-time' is completely meaningless. If P$\neq$NP can be shown, the proof might give us more insight in the reasons why certain problems are more difficult than other, and might guide us to detect and attack the kernel of the difficulties.

4.6. co-NP and good characterizations

The collection co-NP consists of all problems Π for which the complementary problem $\Sigma^* \setminus \Pi$ belongs to NP. Since for any problem $\Pi \in$P, also $\Sigma^* \setminus \Pi$ belongs to P, we have

(4.7) $\quad$ P$\subseteq$NP$\cap$co-NP.

The problems in NP$\cap$co-NP are those for which both a positive answer and a negative answer have a polynomial-time checkable certificate. In other words, any problem Π in NP$\cap$co-NP has a *good characterization*: there exist $\Pi', \Pi'' \in$P and a polynomial p such that for each $w \in \Sigma^*$:

(4.8) $\quad$ there is an $x \in \Sigma^*$ with $wx \in \Pi'$ and size$(x) \leq p(\text{size}(w)) \iff$
there is no $y \in \Sigma^*$ with $wy \in \Pi''$ and size$(y) \leq p(\text{size}(w))$.

Therefore, the problems in NP$\cap$co-NP are called *well-characterized*.

A typical example is Tutte's 1-factor theorem:

(4.9) $\quad$ a graph $G = (V, E)$ has a perfect matching if and only if there is no $U \subseteq V$ such that $G - U$ has more than $|U|$ odd components.

So in this case Π consists of all graphs having a perfect matching, Π' of all strings GM where G is a graph and M a perfect matching in G, and Π'' of all strings GU where G is a graph and U is a subset of the vertex set of G such that $G - U$ has more than $|U|$ odd components. (To be more precise, since Σ^* is the universe, we must add all strings $w\{\}$ to Π'' where w is a word in Σ^* that does not represent a graph.) This is why Tutte's theorem is said to be a good characterization.

In fact, there are very few problems known that have been proved to belong to NP$\cap$co-NP, but that are not known to belong to P. Most problems having a good characterization, have been proved to be solvable in polynomial time. So one may ask: is P=NP$\cap$co-NP?

4.7. Optimization problems

Optimization problems can be transformed to decision problems as follows. Consider a *minimization* problem: minimize $f(x)$ over $x \in X$, where X is a collection of elements derived from the input of the problem, and where f is a rational-valued function on X. (For instance, minimize the length of a Hamiltonian circuit in a given graph, for a given length function on the edges.) This can be transformed to the following decision problem:

(4.10) $\quad$ given a rational number r, is there an $x \in X$ with $f(x) \leq r$?

If we have an upper bound β on the size of the minimum value (being proportional to the sum of the logarithms of the numerator and the denominator), then by asking question (4.10) for $O(\beta)$ choices of r, we can find the optimum

value (by binary search). In this way we usually can derive a polynomial-time algorithm for the minimization problem from a polynomial-time algorithm for the decision problem. Similarly, for maximization problems.

About all combinatorial optimization problems, when framed as a decision problem like (4.10), belong to NP, since a positive answer to question (4.10) can often be certified by just specifying an $x \in X$ satisfying $f(x) \leq r$.

If a combinatorial optimization problem is characterized by a min-max relation like

$$\min_{x \in X} f(x) = \max_{y \in Y} g(y), \tag{4.11}$$

this often leads to a good characterization of the corresponding decision problem. Indeed, if $\min_{x \in X} f(x) \leq r$ holds, it can be certified by an $x \in X$ satisfying $f(x) \leq r$. On the other hand, if $\min_{x \in X} f(x) > r$ holds, it can be certified by a $y \in Y$ satisfying $g(y) > r$. If these certificates can be checked in polynomial time, we say that the min-max relation is a *good characterization*, and that the optimization problem is *well-characterized*.

4.8. NP-complete problems

The NP-complete problems are the problems that are the hardest in NP: every problem in NP can be reduced to them. We make this more precise.

Problem $\Pi \subseteq \Sigma^*$ is said to be *reducible* to problem $\Lambda \subseteq \Sigma^*$ if there exists a polynomial-time algorithm that returns, for any input $w \in \Sigma^*$, an output $x \in \Sigma^*$ with the property:

$$w \in \Pi \iff x \in \Lambda. \tag{4.12}$$

This implies that if Π is reducible to Λ and Λ belongs to P, then also Π belongs to P. Similarly, one may show that if Π is reducible to Λ and Λ belongs to NP, then also Π belongs to NP.

A problem Π is said to be NP-*complete* if each problem in NP is reducible to Π. Hence

(4.13) if some NP-complete problem belongs to P, then P=NP.

Surprisingly, there exist NP-complete problems (Cook [1971]). Even more surprisingly, several prominent combinatorial optimization problems, like the traveling salesman problem, the maximum clique problem, and the maximum cut problem, are NP-complete (Karp [1972b]).

Since then one generally distinguishes between the polynomial-time solvable problems and the NP-complete problems, although there is no proof that these two concepts really are distinct. For almost every combinatorial optimization problem (and many other problems) one has been able to prove either that it is solvable in polynomial time, or that it is NP-complete — and no problem has been proved to be both. But it still has not been excluded that these two concepts are just the same!

The usual approach to prove NP-completeness of problems is to derive it from the NP-completeness of one basic problem, often the satisfiability problem. To this end, we prove NP-completeness of the satisfiability problem in the coming sections.

4.9. The satisfiability problem

To formulate the satisfiability problem, we need the notion of a *Boolean expression*. Examples are:

(4.14) $((x_2 \wedge x_3) \vee \neg(x_3 \vee x_5) \wedge x_2)$, $((\neg x_{47} \wedge x_2) \wedge x_{47})$, and $\neg(x_7 \wedge \neg x_7)$.

Boolean expressions can be defined inductively. We work with an alphabet Σ containing the 'special' symbols '(', ')', '$\wedge$', '$\vee$', '$\neg$', and ',', and not containing the symbols 0 and 1. Then any word not containing any special symbol is a Boolean expression, called a *variable*. Next, if v and w are Boolean expressions, then also $(v \wedge w)$, $(v \vee w)$, and $\neg v$ are Boolean expressions. These rules give us all Boolean expressions. We denote a Boolean expression f by $f(x_1, \ldots, x_k)$ if $x_1, \ldots, x_k$ are the variables occurring in f.

A Boolean expression $f(x_1, \ldots, x_k)$ is called *satisfiable* if there exist $\alpha_1, \ldots, \alpha_k \in \{0, 1\}$ such that $f(\alpha_1, \ldots, \alpha_k) = 1$, using the well-known identities

(4.15) $$\begin{aligned} &0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0, 1 \wedge 1 = 1,\\ &0 \vee 0 = 0, 0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1,\\ &\neg 0 = 1, \neg 1 = 0, (0) = 0, (1) = 1. \end{aligned}$$

Now let SAT$\subseteq \Sigma^*$ be the collection of satisfiable Boolean expressions. SAT is called the *satisfiability problem*.

The satisfiability problem SAT trivially belongs to NP: to certify that $f(x_1, \ldots, x_k)$ belongs to SAT, we can take the equations $x_i = \alpha_i$ that give f the value 1.

4.10. NP-completeness of the satisfiability problem

Let an algorithm be represented by the random access machine (we use notation as in Section 4.2). Consider the performance of the algorithm for some input w of size s (in the alphabet $\{0, 1\}$). We may assume that all entries in the random access machine are stored with the same number of bits, α say, only depending on s. Let q be the length of the array f. We may assume that q is invariant throughout the algorithm, and that q only depends on s. (So the initial input w is extended to an array f of length q.) Let r be the number of iterations performed by the algorithm. We may assume that r only depends on s.

Let m_i be the following word in $\{0, 1\}^*$:

(4.16) $z_0 z_1 \dots z_k f(0) f(1) \dots f(q)$

after performing i iterations (where each z_j and each $f(j)$ is a word in $\{0,1\}^*$ of size α). So it is the content of the machine memory after i iterations. We call the word

(4.17) $h = m_0 m_1 \dots m_r$

the *history*. The size of h is equal to

(4.18) $T := (r+1)(k+q+2)\alpha.$

We call a word h *correct* if there is an input w of size s that leads to history h.

The following observation is basic:

(4.19) given the list of instructions describing the random access machine and given s, we can construct, in time bounded by a polynomial in T, a Boolean expression $g(x_1, \dots, x_T)$ such that any 0,1 word $h = \alpha_1 \dots \alpha_T$ is correct if and only if $g(\alpha_1, \dots, \alpha_T) = 1$.

To see this, we must observe that each of the instructions (4.1) can be described by Boolean expressions in the 0,1 variables describing the corresponding entries.

We can permute the positions in g such that the first s variables correspond to the s input bits, and that the last variable gives the output bit (0 or 1). Let it give the Boolean expression $\tilde{g}(y_1, \dots, y_T)$. Then input $\beta_1 \dots \beta_s$ leads to output 1 if and only if

(4.20) $\tilde{g}(\beta_1, \dots, \beta_s, y_{s+1}, \dots, y_{T-1}, 1) = 1$

has a solution in the variables $y_{s+1}, \dots, y_{T-1}$.

Consider now a problem Π in NP. Let Π' be a problem in P and p a polynomial satisfying (4.6). We can assume that x has size precisely $p(\text{size}(w))$. So if input w of Π has size u, then wx has size $s := u + p(u)$. Let A be a polynomial-time algorithm as described above solving Π' and let $\tilde{g}$ be the corresponding Boolean expression as above. Let $w = \beta_1 \dots \beta_u$. Then w belongs to Π if and only if

(4.21) $\tilde{g}(\beta_1, \dots, \beta_u, y_{u+1}, \dots, y_s, y_{s+1}, \dots, y_{T-1}, 1) = 1$

is solvable. This reduces Π to the satisfiability problem. Hence we have the main result of Cook [1971] (also Levin [1973]):

Theorem 4.1. *The satisfiability problem is* NP*-complete.*

Proof. See above. ▮

4.11. NP-completeness of some other problems

For later reference, we derive from Cook's theorem the NP-completeness of some other problems. First we show that the 3-*satisfiability problem* 3-SAT is NP-complete (Cook [1971], cf. Karp [1972b]). Let B_1 be the set of all words $x_1, \neg x_1, x_2, \neg x_2, \ldots$, where the x_i are words not containing the symbols '$\neg$', '$\wedge$', '$\vee$', '(', ')'. Let B_2 be the set of all words $(w_1 \vee \cdots \vee w_k)$, where $w_1, \cdots, w_k$ are words in B_1 and $1 \leq k \leq 3$. Let B_3 be the set of all words $w_1 \wedge \ldots \wedge w_k$, where $w_1, \ldots, w_k$ are words in B_2. Again, we say that a word $f(x_1, x_2, \ldots) \in B_3$ is *satisfiable* if there exists an assignment $x_i := \alpha_i \in \{0, 1\}$ $(i = 1, 2, \ldots)$ such that $f(\alpha_1, \alpha_2, \ldots) = 1$ (using the identities (4.15)).

Now the 3-satisfiability problem 3-SAT is: given a word $f \in B_3$, decide if it is satisfiable. More formally, 3-SAT is the set of all satisfiable words in B_3.

Corollary 4.1a. *The* 3-*satisfiability problem* 3-SAT *is* NP-*complete.*

Proof. We give a polynomial-time reduction of SAT to 3-SAT. Let $f(x_1, x_2, \ldots)$ be a Boolean expression. Introduce a variable y_g for each subword g of f that is a Boolean expression (not splitting variables).

Now f is satisfiable if and only if the following system is satisfiable:

$$\begin{array}{ll} y_g = y_{g'} \vee y_{g''} & (\text{if } g = (g' \vee g'')), \\ y_g = y_{g'} \wedge y_{g''} & (\text{if } g = (g' \wedge g'')), \\ y_g = \neg y_{g'} & (\text{if } g = \neg g'), \\ y_f = 1. & \end{array} \tag{4.22}$$

Now $y_g = y_{g'} \vee y_{g''}$ can be equivalently expressed by: $y_g \vee \neg y_{g'} = 1, y_g \vee \neg y_{g''} = 1, \neg y_g \vee y_{g'} \vee y_{g''} = 1$. Similarly, $y_g = y_{g'} \wedge y_{g''}$ can be equivalently expressed by: $\neg y_g \vee y_{g'} = 1, \neg y_g \vee y_{g''} = 1, y_g \vee \neg y_{g'} \vee \neg y_{g''} = 1$. The expression $y_g = \neg y_{g'}$ is equivalent to: $y_g \vee y_{g'} = 1, \neg y_g \vee \neg y_{g'} = 1$.

By renaming variables, we thus obtain words $w_1, \ldots, w_k$ in B_2, such that f is satisfiable if and only if the word $w_1 \wedge \ldots \wedge w_k$ is satisfiable. ∎

(As Cook [1971] mentioned, a method of Davis and Putnam [1960] solves the 2-satisfiability problem in polynomial time.)

We next derive that the *partition problem* is NP-complete (Karp [1972b]). This is the problem:

(4.23) Given a collection of subsets of a finite set X, does it contain a subcollection that is a partition of X?

Corollary 4.1b. *The partition problem is* NP-*complete.*

Proof. We give a polynomial-time reduction of 3-SAT to the partition problem. Let $f = w_1 \wedge \ldots \wedge w_k$ be a word in B_3, where $w_1, \ldots, w_k$ are words in B_2. Let $x_1, \ldots, x_m$ be the variables occurring in f. Make a bipartite graph G with colour classes $\{w_1, \ldots, w_k\}$ and $\{x_1, \ldots, x_m\}$, by joining w_i and x_j by

an edge if and only if x_j or $\neg x_j$ occurs in w_i. Let X be the set of all vertices and edges of G.

Let $\mathcal{C}'$ be the collection of all sets $\{w_i\} \cup E'$, where E' is a nonempty subset of the edge set incident with w_i. Let $\mathcal{C}''$ be the collection of all sets $\{x_j\} \cup E'_j$ and $\{x_j\} \cup E''_j$, where E'_j is the set of all edges $\{w_i, x_j\}$ such that x_j occurs in w_i and where E''_j is the set of all edges $\{w_i, x_j\}$ such that $\neg x_j$ occurs in w_i.

Now f is satisfiable if and only if the collection $\mathcal{C}' \cup \mathcal{C}''$ contains a subcollection that partitions X. Thus we have a reduction of 3-SAT to the partition problem. ∎

In later chapters we derive from these results the NP-completeness of several other combinatorial optimization problems.

4.12. Strongly polynomial-time

Roughly speaking, an algorithm is strongly polynomial-time if the number of elementary arithmetic and other operations is bounded by a polynomial in the size of the input, where any number in the input is counted only for 1. Strong polynomial-timeness of an algorithm is of relevance only for problems that have numbers among its input data. (Otherwise, strongly polynomial-time coincides with polynomial-time.)

Consider a problem that has a number k of input parts, like a vertex set, an edge set, a length function. Let $f : \mathbb{Z}_+^{2k} \to \mathbb{R}$. We say that an algorithm takes $O(f)$ time if the algorithm terminates after

$$O(f(n_1, s_1, \ldots, n_k, s_k)) \tag{4.24}$$

operations (including elementary arithmetic operations), where the ith input part consists of n_i numbers of maximum size s_i $(i = 1, \ldots, k)$, and if the numbers occurring during the execution of the algorithm have size

$$O(\max\{s_1, \ldots, s_k\}). \tag{4.25}$$

The algorithm is called a *strongly polynomial-time algorithm* if the algorithm takes $O(f)$ time for some polynomial f in the array lengths $n_1, \ldots, n_k$, where f is independent of $s_1, \ldots, s_k$. If a problem can be solved by a strongly polynomial-time algorithm, we say that it is *solvable in strongly polynomial time* or *strongly polynomial-time solvable*.

An algorithm is called *linear-time* if f can be taken linear in $n_1, \ldots, n_k$, and independent of $s_1, \ldots, s_k$. If a problem can be solved by a linear-time algorithm, we say that it is *solvable in linear time* or *linear-time solvable*.

Rounding a rational x to $\lfloor x \rfloor$ can be done in polynomial-time, by $O(\text{size}(x))$ elementary arithmetic operations. It however cannot be done in strongly polynomial time. In fact, even checking if an integer k is odd or even cannot

be done in strongly polynomial time: for any strongly polynomial-time algorithm with one integer k as input, there is a number L and a rational function $q : \mathbb{Z} \to \mathbb{Q}$ such that if $k > L$, then the output equals $q(k)$. (This can be proved by induction on the number of steps of the algorithm.) However, there do not exist a rational function q and number L such that for $k > L$, $q(k) = 0$ if k is even, and $q(k) = 1$ if k is odd.

We say that an algorithm is *semi-strongly polynomial-time* if we count rounding a rational as one step (one time-unit). We sometimes say *weakly polynomial-time* for polynomial-time, to distinguish from strongly polynomial-time.

4.13. Lists and pointers

Algorithmically, sets (of vertices, edges, etc.) are often introduced and handled as *ordered* sets, called *lists*. Their elements can be indicated just by their positions (*addresses*) in the order: $1, 2, \ldots$. Then attributes (like the capacity, or the ends, of an edge) can be specified in arrays.

Arrays represent functions, and such functions are also called *pointers* if their value is taken as an address. Such functions also allow the value *void*, where the function is undefined. Pointers can be helpful to shorten the running time of an algorithm.

One way to store a list is just in an array. But then updating may take (relatively) much time, for instance, if we would like to perform operations on lists, such as removing or inserting elements or concatenating two lists.

A better way to store a list $S = \{s_1, \ldots, s_k\}$ is as a *linked list*. This is given by a pointer $f : S \setminus \{s_k\} \to S$ where $f(s_i) = s_{i+1}$ for $i = 1, \ldots, k-1$, together with the first element s_1 given by the variable b say (a fixed array of length 1). It makes that S can be scanned in time $O(|S|)$.

If we need to update the list after removing an element from S, it is convenient to store S as a *doubly linked list*. Then we keep, next to f and b, a pointer $g : S \setminus \{s_1\} \to S$ where $g(s_i) = s_{i-1}$ for $i = 2, \ldots, k$, and a variable l say, with $l := s_k$. The virtue of this data structure is that it can be restored in constant time if we remove some element s_j from S. Also concatenating two doubly linked lists can be done in constant time. It is usually easy to build up the doubly linked list along with reading the input, taking time $O(|S|)$.

A convenient (but usually too abundant) way to store a directed graph $D = (V, A)$ using these data structures is as follows. For each $v \in V$, order the sets $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$. Store V as a doubly linked list. Give pointers $t, h : A \to V$, where $t(a)$ and $h(a)$ are the tail and head of a. Give four pointers $V \to A$, indicating the first and last (respectively) arc in the lists $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$ (respectively). Give four pointers $A \to A$, indicating for each $a \in A$, the previous and next (respectively) arc in the lists $\delta^{\text{in}}(h(a))$ and $\delta^{\text{out}}(t(a))$ (respectively). (Values may be 'void'. One can avoid the value 'void'

by merging the latter eight pointers described into four pointers $V \cup A \to V \cup A$.)

If, in the input of a problem, a directed graph is given as a string (or file), like

$$(\{a, b, c, d\}, \{(a, c), (a, d), (b, d), (c, d)\}), \tag{4.26}$$

we can build up the above data structure in time linear in the length of the string. Often, when implementing a graph algorithm, a subset of this structure will be sufficient. Undirected graphs can be handled similarly by choosing an arbitrary orientation of the edges. (So each edge becomes a list.)

4.14. Further notes

4.14a. Background literature on algorithms and complexity

Background literature on algorithms and complexity includes Knuth [1968] (data structures), Garey and Johnson [1979] (complexity, NP-completeness), Papadimitriou and Steiglitz [1982] (combinatorial optimization and complexity), Aho, Hopcroft, and Ullman [1983] (data structures and complexity), Tarjan [1983] (data structures), Cormen, Leiserson, and Rivest [1990] (algorithms), Papadimitriou [1994] (complexity), Sipser [1997] (algorithms, complexity), and Mehlhorn and Näher [1999] (data structures, algorithms and algorithms).

In this book we restrict algorithms and complexity to deterministic, sequential, and exact. For other types of algorithms and complexity we refer to the books by Motwani and Raghavan [1995] (randomized algorithms and complexity), Leighton [1992,2001] (parallel algorithms and complexity), and Vazirani [2001] (approximation algorithms and complexity). A survey on practical problem solving with cutting planes was given by Jünger, Reinelt, and Thienel [1995].

4.14b. Efficiency and complexity historically

In the history of complexity, more precisely, in the conception of the notions 'polynomial-time' and 'NP-complete', two lines loom up: one motivated by questions in logic, recursion, computability, and theorem proving, the other more down-to-earth focusing on the complexity of some concrete problems, with background in discrete mathematics and operations research.

Until the mid-1960s, the notions of efficiency and complexity were not formalized. The notion of algorithm was often used for a method that was better than brute-force enumerating. We focus on how the ideas of polynomial-time and NP-complete got shape. We will not go into the history of data structures, abstract computational complexity, or the subtleties inside and beyond NP (for which we refer to Papadimitriou [1994]).

We quote references in chronological order. This order is quite arbitrary, since the papers mostly seem to be written isolated from each other and they react very seldom to each other.

Maybe the first paper that was concerned with the complexity of computation is an article by Lamé [1844], who showed that the number of iterations in the Euclidean g.c.d. algorithm is linear in the logarithm of the smallest of the two (natural) numbers:

> Dans les traités d'Arithmétique, on se contente de dire que le nombre des divisions à effectuer, dans la recherche du plus grand commun diviseur entre deux entiers, *ne pourra pas surpasser la moitié du plus petit.* Cette limite, qui peut être dépassée si les nombres sont petits, s'éloigne outre mesure quand ils ont plusieurs chiffres. L'exagération est alors semblable à celle qui assignerait la moitié d'un nombre comme la limite de son logarithme; l'analogie devient évidente quand on connaît le théorème suivant:
>
> Théorème. *Le nombre des divisions à effectuer, pour trouver le plus grand commun diviseur entre deux entiers* A*, et* B<A*, est toujours moindre que cinq fois le nombre des chiffres de* B.[3]

The first major combinatorial optimization problem for which a polynomial-time algorithm was given is the shortest spanning tree problem, by Borůvka [1926a, 1926b] and Jarník [1930], but these papers do not discuss the complexity issue — the efficiency of the method might have been too obvious. Choquet [1938] mentioned explicitly an estimate for the number of iterations in finding a shortest spanning tree:

> Le réseau cherché sera tracé après $2n$ opérations élémentaires au plus, en appelant opération élémentaire la recherche du continu le plus voisin d'un continu donné.[4]

The traveling salesman and the assignment problem

The traveling salesman problem and the assignment problem have been long-term bench-marks that gave shape to the ideas on efficiency and complexity.

Menger might have been the first to ask attention for the complexity of the traveling salesman problem. In the session of 5 February 1930 of his *mathematische Kolloquium* in Vienna (as reported in Menger [1932a]), he introduced *das Botenproblem*, later called the traveling salesman problem and raised the question for a better-than-finite algorithm:

> Dieses Problem ist natürlich stets durch endlichviele Versuche lösbar. Regeln, welche die Anzahl der Versuche unter die Anzahl der Permutationen der gegebenen Punkte herunterdrücken würden, sind nicht bekannt.[5]

---

[3] In the handbooks of Arithmetics, one contents oneself with saying that, in the search for the greatest common divisor of two integers, the number of divisions to execute *could not surpass half of the smallest [integer].* This bound, that can be exceeded if the numbers are small, goes away beyond measure when they have several digits. The exaggeration then is similar to that which would assign half of a number as bound of its logarithm; the analogy becomes clear when one knows the following theorem:

Theorem. *The number of divisions to execute, to find the greatest common divisor of two integers* A*, and* B<A*, is always smaller than five times the number of digits of* B*.*

[4] The network looked for will be traced after at most $2n$ elementary operations, calling the search for the continuum closest to a given continuum an elementary operation.

[5] Of course, this problem is solvable by finitely many trials. Rules which would push the number of trials below the number of permutations of the given points, are not known.

Ghosh [1949] observed that the problem of finding a shortest tour along n random points in the plane (which is the traveling salesman problem) is hard:

> After locating the n random points in a map of the region, it is very difficult to find out *actually* the shortest path connecting the points, unless the number n is very small, which is seldom the case for a large-scale survey.

We should realize however that at that time also the (now known to be polynomial-time solvable) assignment problem was considered to be hard. In an Address delivered on 9 September 1949 at a meeting of the American Psychological Association at Denver, Colorado, Thorndike [1950] studied the problem of the 'classification' of personnel:

> There are, as has been indicated, a finite number of permutations in the assignment of men to jobs. When the classification problem as formulated above was presented to a mathematician, he pointed to this fact and said that from the point of view of the mathematician there was no problem. Since the number of permutations was finite, one had only to try them all and choose the best. He dismissed the problem at that point. This is rather cold comfort to the psychologist, however, when one considers that only ten men and ten jobs mean over three and a half million permutations. Trying out all the permutations may be a mathematical solution to the problem, it is not a practical solution.

But, in a RAND Report dated 5 December 1949, Robinson [1949] reported that an 'unsuccessful attempt' to solve the traveling salesman problem, led her to a 'cycle-cancelling' method for the optimum assignment problem, which in fact stands at the basis of efficient algorithms for network problems. She gave an optimality criterion for the assignment problem (absence of negative-length cycles in the residual graph). As for the traveling salesman problem she mentions:

> Since there are only a finite number of paths to consider, the problem consists in finding a method for picking out the optimal path when n is moderately large, say $n = 50$. In this case, there are more than 10^{62} possible paths, so we can not simply try them all. Even for as few as 10 points, some short cuts are desirable.

She also observed that the number of feasible solutions is not a measure for the complexity (where 'it' refers to the assignment problem):

> However at first glance, it looks more difficult than the traveling salesman probl[e]m, for there are obviously many more systems of circuits than circuits.

The development of the simplex method for linear programming, and its, in practice successful, application to combinatorial optimization problems like assignment and transportation, led to much speculation on the theoretical efficiency of the simplex method. In his paper describing the application of the simplex method to the transportation problem, Dantzig [1951a] mentioned (after giving a variable selection criterion that he speculates to lead to favourable computational experience for large-scale practical problems):

> This does not mean that theoretical problems could not be "cooked up" where this criterion is weak, but that in practical problems the number of steps has not been far from $m + n - 1$.

(Here n and m are the numbers of vertices and arcs, respectively.)

At the Symposium on Linear Inequalities and Programming in Washington, D.C. in 1951, Votaw and Orden [1952] reported on early computational results with the simplex method (on the SEAC), and claimed (without proof) that the simplex method is polynomial-time for the transportation problem (a statement refuted by Zadeh [1973a]):

> As to computation time, it should be noted that for moderate size problems, say $m \times n$ up to 500, the time of computation is of the same order of magnitude as the time required to type the initial data. The computation time on a sample computation in which m and n were both 10 was 3 minutes. The time of computation can be shown by study of the computing method and the code to be proportional to $(m+n)^3$.

Another early mention of polynomial-time as efficiency criterion is by von Neumann, who considered the complexity of the assignment problem. In a talk in the Princeton University Game Seminar on 26 October 1951, he described a method which is equivalent to finding a best strategy in a certain zero-sum two-person game. According to a transcript of the talk (cf. von Neumann [1951,1953]), von Neumann noted the following on the number of steps:

> It turns out that this number is a moderate power of n, i.e., considerably smaller than the "obvious" estimate $n!$ mentioned earlier.

However, no further argumentation is given.

In a Cowles Commission Discussion Paper of 2 April 1953, also Beckmann and Koopmans [1953] asked for better-than-finite methods for the assignment problem, but no explicit complexity measure was proposed, except that the work should be reduced to 'manageable proportions':

> It should be added that in all the assignment problems discussed, there is, of course, the obvious brute force method of enumerating all assignments, evaluating the maximand at each of these, and selecting the assignment giving the highest value. This is too costly in most cases of practical importance, and by a method of solution we have meant a procedure that reduces the computational work to manageable proportions in a wider class of cases.

During the further 1950s, better-than-finite methods were developed for the assignment and several other problems like shortest path and maximum flow. These methods turned out to give polynomial-time algorithms (possibly after modification), and several speedups were found — but polynomial-time was, as yet, seldom marked as efficiency criterion. The term 'algorithm' was often used just to distinguish from complete enumeration, but no mathematical characterization was given.

Kuhn [1955b,1956] introduced the 'Hungarian method' for the assignment problem (inspired by the proof method of Egerváry [1931]). Kuhn contented himself with showing finiteness of the method, but Munkres [1957] showed that it is strongly polynomial-time:

> The final maximum on the number of operations needed is
>
> $$(11n^3 + 12n^2 + 31n)/6.$$
>
> This maximum is of theoretical interest, since it is much smaller than the $n!$ operations necessary in the most straightforward attack on the problem.

As for the maximum flow problem, Ford and Fulkerson [1955,1957b] showed that their augmenting path method is finite, but only Dinits [1970] and Edmonds and Karp [1970,1972] showed that it can be adapted to be (strongly) polynomial-time.

Several algorithms were given for finding shortest paths (Shimbel [1955], Leyzorek, Gray, Johnson, Ladew, Meaker, Petry, and Seitz [1957], Bellman [1958], Dantzig [1958,1960], Dijkstra [1959], Moore [1959]), and most of them are obviously strongly polynomial-time. (Ford [1956] gave a liberal shortest path algorithm that may require exponential time (Johnson [1973a,1973b,1977a]).)

Similarly, the interest in the shortest spanning tree problem revived, leading to old and new strongly polynomial-time algorithms (Kruskal [1956], Loberman and Weinberger [1957], Prim [1957], and Dijkstra [1959]).

The traveling salesman problem resisted these efforts. In the words of Dantzig, Fulkerson, and Johnson [1954a,1954b]:

> Although algorithms have been devised for problems of similar nature, e.g., the optimal assignment problem,[3,7,8] little is known about the traveling-salesman problem. We do not claim that this note alters the situation very much;

The papers [3,7,8] referred to, are the papers Dantzig [1951a], Votaw and Orden [1952], and von Neumann [1953], quoted above.

The use of the word 'Although' in the above quote makes it unclear what Dantzig, Fulkerson, and Johnson considered to be an algorithm. Their algorithm uses polyhedral methods to solve the traveling salesman problem, while Dantzig [1951a] and Votaw and Orden [1952] apply the simplex method to solve the assignment and transportation problems. In a follow-up paper, Dantzig, Fulkerson, and Johnson [1959] seem to have come to the conclusion that both methods are of a comparable level:

> Neither does the example, as we have solved it, indicate how one could make the combinatorial analysis a routine procedure. This can certainly be done (by enumeration, if nothing else)—but the fundamental question is: does the use of a few linear inequalities in general reduce the combinatorial magnitude of such problems significantly?
> We do not know the answer to this question in any theoretical sense, but it is our feeling, based on our experience in using the method, that it does afford a practical means of computing optimal tours in problems that are not too huge. It should be noted that a similar question, for example, arises when one uses the simplex method to find optimal solutions to linear programs, since no one has yet proved that the simplex method cuts down the computational task significantly from the crude method of examining all basic solutions, say. Nonetheless, people do use the simplex method because of successful experience with many hundreds of practical problems.

The feeling that the traveling salesman problem is more complex than the assignment problem was stated by Tompkins [1956]:

> A traveling-salesman problem is in some respects similar to the assignment problem. It seems definitely more difficult, however.

Tompkins described a branch-and-bound scheme to the permutation problem (including assignment and traveling salesman), but said:

> It must be noted, however, that this is not a completely satisfactory scheme for solution of such problems. In a few important cases (such as the assignment problem) more efficient machine methods have been devised.

The available algorithms for the traveling salesman problem were also not acceptable to Flood [1956]:

> There are as yet no acceptable computational methods, and surprisingly few mathematical results relative to the problem.

He mentioned that the problem might be 'fundamentally complex':

> Very recent mathematical work on the traveling-salesman problem by I. Heller, H.W. Kuhn, and others indicates that the problem is fundamentally complex. It seems very likely that quite a different approach from any yet used may be required for succesful treatment of the problem. In fact, there may well be no general method for treating the problem and impossibility results would also be valuable.

Logic and computability

Parallel to those motivated by concrete combinatorial problems, interest in complexity arose in the circles of logicians and recursion theorists.

A first quote is from a letter of K. Gödel to J. von Neumann of 20 March 1956. (The letter was reviewed by Hartmanis [1989], to whose attention it was brought by G. Heise. A reproduction and full translation was given by Sipser [1992].)

Turing [1937] proved that there is no algorithm that decides if a given statement in full first-order predicate logic has a proof (the unsolvability of Hilbert's *Entscheidungsproblem* of the *engere Funktionskalkül* (which is the term originally used by Hilbert for full first-order predicate calculus; Turing [1937] translated it into restricted functional calculus)). It implies the result of Gödel [1931] that there exist propositions A such that neither A nor $\neg A$ is provable (in the formalism of the Principia Mathematica).

But a *given* proof can algorithmically be checked, hence there is a finite algorithm to check if there exists a proof of any prescribed length n (simply by enumeration). Nowadays it is known that this is in fact an NP-complete problem (the satisfiability problem is a special case). Gödel asked for the opinion of von Neumann on whether a proof could be found algorithmically in time linear (or else quadratic) in the length of the proof — quite a bold statement, which Gödel yet seemed to consider plausible:

> Man kann offenbar leicht eine Turingmaschine konstruieren, welche von jeder Formel F des engeren Funktionenkalküls u. jeder natürl. Zahl n zu entscheiden gestattet ob F einen Beweis der Länge n hat [Länge = Anzahl der Symbole]. Sei $\psi(F,n)$ die Anzahl der Schritte die die Maschine dazu benötigt u. sei $\varphi(n) = \max_F \psi(F,n)$. Die Frage ist, wie rasch $\varphi(n)$ für eine optimale Maschine wächst. Man kann zeigen $\varphi(n) \geq Kn$. Wenn es wirklich eine Maschine mit $\varphi(n) \sim K.n$ (oder auch nur $\sim Kn^2$) gäbe, hätte das Folgerungen von der grössten Tragweite. Es würde nämlich offenbar bedeuten, dass man trotz der Unlösbarkeit des Entscheidungsproblems die Denkarbeit des Mathematikers bei ja-oder-nein Fragen vollständig* durch Maschinen ersetzen könnte. Man müsste ja bloss das n so gross wählen, dass, wenn die Maschine kein Resultat liefert es auch keinen Sinn hat über das Problem nachzudenken. Nun scheint es mir aber durchaus im Bereich der Möglichkeit zu liegen, dass $\varphi(n)$ so langsam wächst. Denn 1.) scheint $\varphi(n) \geq Kn$ die einzige Abschätzung zu sein, die man durch eine Verallgemeinerung des Beweises für die Unlösbarkeit des Entscheidungsproblems erhalten kann; 2. bedeutet ja $\varphi(n) \sim K.n$ (oder $\sim Kn^2$) bloss, dass die Anzahl der Schritte gegenüber dem blossen Probieren von N auf $\log N$ (oder $(\log N)^2$) verringert werden kann. So starke Verringerungen kommen aber bei andern finiten Problemen durchaus vor, z.B. bei der Berechnung eines quadratischen Restsymbols durch wiederholte Anwendung des Reziprozitätsgesetzes. Es wäre interessant zu wissen, wie es damit z.B. bei der Feststellung, ob eine Zahl Primzahl ist, steht u. wie stark im allgemeinen bei finiten kombinatorischen Problemen die Anzahl der Schritte gegenüber dem blossen Probieren verringert werden kann.
>
> * abgesehen von der Aufstellung der Axiome[6]

[6] Clearly, one can easily construct a Turing machine, which makes it possible to decide, for each formula F of the restricted functional calculus and each natural number n, whether F has a proof of length n [length = number of symbols]. Let $\psi(F,n)$ be the number of steps that the machine needs for that and let $\varphi(n) = \max_F \psi(F,n)$. The question is, how fast $\varphi(n)$ grows for an optimal machine. One can show $\varphi(n) \geq Kn$.

(For integers a, p with p prime, the Legendre symbol $(\frac{a}{p})$ indicates if a is a quadratic residue mod p (that is, if $x^2 = a \pmod p$ has an integer solution x), and can be calculated by $\log a + \log p$ arithmetic operations (using the Jacobi symbol and the reciprocity law) — so Gödel took the logarithms of the numbers as size.)

The unavoidability of brute-force search for finding the smallest Boolean representation for a function was claimed by Yablonskiĭ [1959] (cf. Trakhtenbrot [1984]).

Davis and Putnam [1960] gave a method for the satisfiability problem (in reaction to earlier, exponential-time methods of Gilmore [1960] and Wang [1960] based on elimination of variables), which they claimed to have some (not exactly formulated) efficiency:

> In the present paper, a uniform proof procedure for quantification theory is given which is feasible for use with some rather complicated formulas and which does not ordinarily lead to exponentiation.

(It was noticed later by Cook [1971] that Davis and Putnam's method gives a polynomial-time method for the 2-satisfiability problem.)

A mathematical framework for computational complexity of algorithms was set up by Hartmanis and Stearns [1965]. They counted the number of steps made by a multitape Turing machine to solve a decision problem. They showed that for all 'real-time countable' functions f, g (which include all functions n^k, k^n, $n!$, and sums, products, and compositions of them) the following holds: if each problem solvable in time $O(f)$ is also solvable in time $O(g)$, then $f = O(g^2)$. This implies, for instance, that there exist problems solvable in time $O(n^5)$ but not in time $O(n^2)$, and problems solvable in time $O(2^n)$ but not in time $O(2^{n/3})$ (hence not in polynomial time).

Polynomial-time

In the summer of 1963, at a Workshop at the RAND Corporation, Edmonds discovered that shrinking leads to a polynomial-time algorithm to find a maximum-size matching in any graph — a basic result in graph algorithmics. It was described in the paper Edmonds [1965d] (received November 22, 1963), in which he also gave his views on algorithms and complexity:

> When really there were a machine with $\varphi(n) \sim K.n$ (or even just $\sim Kn^2$), that would have consequences of the largest impact. In particular, it would obviously mean that, despite the unsolvability of the Entscheidungsproblem, one could replace the brainwork of the mathematician in case of yes-or-no questions fully* by machines. One should indeed only choose n so large that if the machine yields no result, there is also no sense in thinking about the problem. Now it seems to me, however, to lie completely within the range of possibility that $\varphi(n)$ grows that slowly. Because 1.) $\varphi(n) \geq Kn$ seems to be the only estimate that one can obtain by a generalization of the proof for the unsolvability of the Entscheidungsproblem; 2. $\varphi(n) \sim K.n$ (or $\sim Kn^2$) means indeed only that the number of steps can be reduced compared to mere trying from N to $\log N$ (or $(\log N)^2$). Such strong reductions occur however definitely at other finite problems, e.g. at the calculation of a quadratic residue symbol by repeated application of the reciprocity law. It would be interesting to know how this is e.g. for the decision if a number is prime, and how strong in general, for finite combinatorial problems, the number of steps can be reduced compared to mere trying.
>
> * apart from the set-up of the axioms

> For practical purposes computational details are vital. However, my purpose is only to show as attractively as I can that there is an efficient algorithm. According to the dictionary, "efficient" means "adequate in operation or performance." This is roughly the meaning I want—in the sense that it is conceivable for maximum matching to have no efficient algorithm. Perhaps a better word is "good."
> I am claiming, as a mathematical result, the existence of a *good* algorithm for finding a maximum size matching in a graph.
> There is an obvious finite algorithm, but that algorithm increases in difficulty exponentially with the size of the graph. It is by no means obvious whether *or not* there exists an algorithm whose difficulty increases only algebraically with the size of the graph.

Moreover:

> For practical purposes the difference between algebraic and exponential order is often more crucial than the difference between finite and non-finite.

In another paper, Edmonds [1965c] introduced the term *good characterization*:

> We seek a good characterization of the minimum number of independent sets into which the columns of a matrix of M_F can be partitioned. As the criterion of "good" for the characterization we apply the "principle of the absolute supervisor." The good characterization will describe certain information about the matrix which the supervisor can require his assistant to search out along with a minimum partition and which the supervisor can then use with ease to verify with mathematical certainty that the partition is indeed minimum. Having a good characterization does not mean necessarily that there is a good algorithm. The assistant might have to kill himself with work to find the information and the partition.

Further motivation for polynomial-time solvability was given by Edmonds [1967b]:

> An algorithm which is good in the sense used here is not necessarily very good from a practical viewpoint. However, the good-versus-not-good dichotomy is useful. It is easily formalized (say, relative to a Turing machine, or relative to a typical digital computer with an unlimited supply of tape), and usually it is easily recognized informally. Within limitations it does have practical value, and it does admit refinements to "how good" and "how bad". The classes of problems which are respectively known and not known to have good algorithms are very interesting theoretically.

Edmonds [1967a] conjectured that there is no polynomial-time algorithm for the traveling salesman problem — in language developed later, this is equivalent to NP≠P:

> I conjecture that there is no good algorithm for the traveling salesman problem. My reasons are the same as for any mathematical conjecture: (1) It is a legitimate mathematical possibility, and (2) I do not know.

Also Cobham [1965] singled out polynomial-time as a complexity criterion, in a paper on Turing machines and computability, presented at the 1964 International Congress on Logic, Methodology and Philosophy of Science in Jerusalem (denoting the size of n by $l(n)$):

> To obtain some idea as to how we might go about the further classification of relatively simple functions, we might take a look at how we ordinarily set about computing some of the more common of them. Suppose, for example, that m and n are two numbers given in decimal notation with one written above the other and their right ends aligned. Then to add m and n we start at the right and

> proceed digit-by-digit to the left writing down the sum. No matter how large m and n, this process terminates with the answer after a number of steps equal at most to one greater than the larger of $l(m)$ and $l(n)$. Thus the process of adding m and n can be carried out in a number of steps which is bounded by a linear polynomial in $l(m)$ and $l(n)$. Similarly, we can multiply m and n in a number of steps bounded by a quadratic polynomial in $l(m)$ and $l(n)$. So, too, the number of steps involved in the extraction of square roots, calculation of quotients, etc., can be bounded by polynomials in the lengths of the numbers involved, and this seems to be a property of simple functions in general. This suggests that we consider the class, which I will call $\mathcal{L}$, of all functions having this property.

At a symposium in New York in 1966, also Rabin [1967] noted the importance of polynomial-time solvability:

> In the following we shall consider an algorithm to be practical if, for automata with n states, it requires at most cn^k (k is a fixed integer and c a fixed constant) computational steps. This stipulation is, admittedly, both vague and arbitrary. We do not, in fact cannot, define what is meant by a computational step, thus have no precise and general measure for the complexity of algorithms. Furthermore, there is no compelling reason to classify algorithms requiring cn^k steps as practical.
> Several points may be raised in defense of the above stipulation. In every given algorithm the notion of a computational step is quite obvious. Hence there is not much vagueness about the measure of complexity of existing algorithms. Another significant pragmatic fact is that all existing algorithms either require up to about n^4 steps or else require 2^n or worse steps. Thus drawing the line of practicality between algorithms requiring n^k steps and algorithms for which no such bound exists seems to be reasonable.

NP-completeness

Cook [1971] proved the NP-completeness of the satisfiability problem ('Theorem 1') and of the 3-satisfiability problem and the subgraph problem ('Theorem 2') and mentioned (the class of polynomial-time solvable problems is denoted by $\mathcal{L}_*$; { tautologies } is the satisfiability problem):

> Theorem 1 and its corollary give strong evidence that it is not easy to determine whether a given proposition formula is a tautology, even if the formula is in normal disjunctive form. Theorems 1 and 2 together suggest that it is fruitless to search for a polynomial decision procedure for the subgraph problem, since success would bring polynomial decision procedures to many other apparently intractable problems. Of course, the same remark applies to any combinatorial problem to which { tautologies } is P-reducible.
> Furthermore, the theorems suggest that { tautologies } is a good candidate for an interesting set not in $\mathcal{L}_*$, and I feel it is worth spending considerable effort trying to prove this conjecture. Such a proof would be a major breakthrough in complexity theory.

So Cook conjectured that NP≠P.

Also Levin [1973] considered the distinction between NP and P:

> After the concept of the algorithm had been fully refined, the algorithmic unsolvability of a number of classical large-scale problems was proved (including the problems of the identity of elements of groups, the homeomorphism of varieties, the solvability of the Diophantine equations, etc.). These findings dispensed with the question of finding a practical technique for solving the indicated problems. However, the existence of algorithms for the solution of other problems does not

> eliminate the analogous question, because the volume of work mandated by those algorithms is fantastically large. This is the situation with so-called sequential (or exhaustive) search problems, including: the minimization of Boolean functions, the search for proofs of finite length, the determination of the isomorphism of graphs, etc. All of these problems are solved by trivial algorithms entailing the sequential scanning of all possibilities. The operating time of the algorithms, however, is exponential, and mathematicians nurture the conviction that it is impossible to find simpler algorithms.

Levin next announced that any problem in NP (in his terminology, any 'sequential search problem') can be reduced to the satisfiability problem, and to a few other problems.

The wide extent of NP-completeness was disclosed by Karp [1972b], by showing that a host of prominent combinatorial problems is NP-complete, therewith revealing the fissure in the combinatorial optimization landscape. According to Karp, his theorems

> strongly suggest, but do not imply, that these problems, as well as many others, will remain intractable perpetually.

Karp also introduced the notation P and NP, and in a subsequent paper, Karp [1975] introduced the term NP-complete.

Sipser [1992] gave an extensive account on the history of the P=NP question. Hartmanis [1989] reviewed the historic setting of 'Gödel, von Neumann and the P=?NP Problem'. Other papers on the history of complexity are Hartmanis [1981], Trakhtenbrot [1984] (Russian approaches), Karp [1986], and Iri [1987] (the Japanese view).

Chapter 5

Preliminaries on polyhedra and linear and integer programming

This chapter surveys what we need on polyhedra and linear and integer programming. Most background can be found in Chapters 7–10, 14, 16, 19, 22, and 23 of Schrijver [1986b]. We give proofs of a few easy further results that we need in later parts of the present book.

The results of this chapter are mostly formulated for real space, but are maintained when restricted to rational space. So the symbol $\mathbb{R}$ can be replaced by the symbol $\mathbb{Q}$. In applying these results, we add the adjective *rational* when we restrict ourselves to rational numbers.

5.1. Convexity and halfspaces

A subset C of $\mathbb{R}^n$ is *convex* if $\lambda x + (1-\lambda)y$ belongs to C for all $x, y \in C$ and each λ with $0 \leq \lambda \leq 1$. A *convex body* is a compact convex set.

The *convex hull* of a set $X \subseteq \mathbb{R}^n$, denoted by conv.hull$X$, is the smallest convex set containing X. Then:

$$\text{conv.hull}X = \{\lambda_1 x_1 + \cdots + \lambda_k x_k \mid k \geq 1, x_1, \ldots, x_k \in X, \lambda_1, \ldots, \lambda_k \in \mathbb{R}_+, \lambda_1 + \cdots + \lambda_k = 1\}. \tag{5.1}$$

A useful fundamental result was proved by Carathéodory [1911]:

Theorem 5.1 (Carathéodory's theorem)**.** *For any $X \subseteq \mathbb{R}^n$ and $x \in$ conv.hull$X$, there exist affinely independent vectors $x_1, \ldots, x_k$ in X with $x \in$ conv.hull$\{x_1, \ldots, x_k\}$.*

(Corollary 7.1f in Schrijver [1986b].)

A subset H of $\mathbb{R}^n$ is called an *affine halfspace* if $H = \{x \mid c^\mathsf{T} x \leq \delta\}$, for some $c \in \mathbb{R}^n$ with $c \neq \mathbf{0}$ and some $\delta \in \mathbb{R}$. If $\delta = 0$, then H is called a *linear halfspace*.

Let $X \subseteq \mathbb{R}^n$. The set conv.hull$X + \mathbb{R}^n_+$ is called the *up hull* of X, and the set conv.hull$X - \mathbb{R}^n_+$ the *down hull* of X.

5.2. Cones

A subset C of $\mathbb{R}^n$ is called a *(convex) cone* if $C \neq \emptyset$ and $\lambda x + \mu y \in C$ whenever $x, y \in C$ and $\lambda, \mu \in \mathbb{R}_+$. The cone *generated by* a set X of vectors is the smallest cone containing X:

(5.2) $\text{cone}X = \{\lambda_1 x_1 + \cdots + \lambda_k x_k \mid k \geq 0, \lambda_1, \ldots, \lambda_k \geq 0, x_1, \ldots, x_k \in X\}$.

There is a variant of Carathéodory's theorem:

Theorem 5.2. *For any $X \subseteq \mathbb{R}^n$ and $x \in \text{cone}X$, there exist linearly independent vectors $x_1, \ldots, x_k$ in X with $x \in \text{cone}\{x_1, \ldots, x_k\}$.*

A cone C is *polyhedral* if there is a matrix A such that

(5.3) $C = \{x \mid Ax \leq \mathbf{0}\}$.

Equivalently, C is polyhedral if it is the intersection of finitely many linear halfspaces.

Results of Farkas [1898,1902], Minkowski [1896], and Weyl [1935] imply that

(5.4) a convex cone is polyhedral if and only if it is finitely generated,

where a cone C is *finitely generated* if $C = \text{cone}X$ for some finite set X. (Corollary 7.1a in Schrijver [1986b].)

5.3. Polyhedra and polytopes

A subset P of $\mathbb{R}^n$ is called a *polyhedron* if there exists an $m \times n$ matrix A and a vector $b \in \mathbb{R}^m$ (for some $m \geq 0$) such that

(5.5) $P = \{x \mid Ax \leq b\}$.

So P is a polyhedron of and only if it is the intersection of finitely many affine halfspaces. If (5.5) holds, we say that $Ax \leq b$ *determines* P. Any inequality $c^{\mathsf{T}}x \leq \delta$ is called *valid* for P if $c^{\mathsf{T}}x \leq \delta$ holds for each $x \in P$.

A subset P of $\mathbb{R}^n$ is called a *polytope* if it is the convex hull of finitely many vectors in $\mathbb{R}^n$. Motzkin [1936] showed:

(5.6) a set P is a polyhedron if and only if $P = Q + C$ for some polytope Q and some cone C.

(Corollary 7.1b in Schrijver [1986b].) If $P \neq \emptyset$, then C is unique and is called the *characteristic cone* $\text{char.cone}(P)$ of P. Then:

(5.7) $\text{char.cone}(P) = \{y \in \mathbb{R}^n \mid \forall x \in P \forall \lambda \geq 0 : x + \lambda y \in P\}$.

If $P = \emptyset$, then by definition its characteristic cone is $\text{char.cone}(P) := \{\mathbf{0}\}$.

(5.6) implies the following fundamental result (Minkowski [1896], Steinitz [1916], Weyl [1935]):

(5.8) a set P is a polytope if and only if P is a bounded polyhedron.

(Corollary 7.1c in Schrijver [1986b].)

A polyhedron P is called *rational* if it is determined by a rational system of linear inequalities. Then a rational polytope is the convex hull of a finite number of rational vectors.

5.4. Farkas' lemma

A system $Ax \leq b$ is called *feasible* (or *solvable*) if it has a solution x. Feasibility of a system $Ax \leq b$ of linear inequalities is characterized by *Farkas' lemma* (Farkas [1894,1898], Minkowski [1896]):

Theorem 5.3 (Farkas' lemma). *$Ax \leq b$ is feasible $\iff$ $y^\mathsf{T} b \geq 0$ for each $y \geq \mathbf{0}$ with $y^\mathsf{T} A = \mathbf{0}^\mathsf{T}$.*

(Corollary 7.1e in Schrijver [1986b].) Theorem 5.3 is equivalent to:

Corollary 5.3a (Farkas' lemma — variant). *$Ax = b$ has a solution $x \geq \mathbf{0}$ $\iff$ $y^\mathsf{T} b \geq 0$ for each y with $y^\mathsf{T} A \geq \mathbf{0}^\mathsf{T}$.*

(Corollary 7.1d in Schrijver [1986b].) A second equivalent variant is:

Corollary 5.3b (Farkas' lemma — variant). *$Ax \leq b$ has a solution $x \geq \mathbf{0}$ $\iff$ $y^\mathsf{T} b \geq 0$ for each $y \geq \mathbf{0}$ with $y^\mathsf{T} A \geq \mathbf{0}^\mathsf{T}$.*

(Corollary 7.1f in Schrijver [1986b].) A third equivalent, affine variant of Farkas' lemma is:

Corollary 5.3c (Farkas' lemma — affine variant). *Let $Ax \leq b$ be a feasible system of inequalities and let $c^\mathsf{T} x \leq \delta$ be an inequality satisfied by each x with $Ax \leq b$. Then for some $\delta' \leq \delta$, the inequality $c^\mathsf{T} x \leq \delta'$ is a nonnegative linear combination of the inequalities in $Ax \leq b$.*

(Corollary 7.1h in Schrijver [1986b].)

5.5. Linear programming

Linear programming, abbreviated to *LP*, concerns the problem of maximizing or minimizing a linear function over a polyhedron. Examples are

(5.9) $\max\{c^\mathsf{T} x \mid Ax \leq b\}$ and $\min\{c^\mathsf{T} x \mid x \geq \mathbf{0}, Ax \geq b\}$.

If a supremum of a linear function over a polyhedron is finite, then it is attained as a maximum. So a maximum is finite if the value set is nonempty and has an upper bound. Similarly for infimum and minimum.

The *duality theorem of linear programming* says (von Neumann [1947], Gale, Kuhn, and Tucker [1951]):

Theorem 5.4 (duality theorem of linear programming)**.** *Let A be a matrix and b and c be vectors. Then*

$$\max\{c^\mathsf{T}x \mid Ax \le b\} = \min\{y^\mathsf{T}b \mid y \ge \mathbf{0}, y^\mathsf{T}A = c^\mathsf{T}\}, \tag{5.10}$$

if at least one of these two optima is finite.

(Corollary 7.1g in Schrijver [1986b].) So, in particular, if at least one of the optima is finite, then both are finite.

Note that the inequality $\le$ in (5.10) is easy, since $c^\mathsf{T}x = y^\mathsf{T}Ax \le y^\mathsf{T}b$. This is called *weak duality*.

There are several equivalent forms of the duality theorem of linear programming, like

$$\begin{aligned}
&\max\{c^\mathsf{T}x \mid x \ge \mathbf{0}, Ax \le b\} = \min\{y^\mathsf{T}b \mid y \ge \mathbf{0}, y^\mathsf{T}A \ge c^\mathsf{T}\},\\
&\max\{c^\mathsf{T}x \mid x \ge \mathbf{0}, Ax = b\} = \min\{y^\mathsf{T}b \mid y^\mathsf{T}A \ge c^\mathsf{T}\},\\
&\min\{c^\mathsf{T}x \mid x \ge \mathbf{0}, Ax \ge b\} = \max\{y^\mathsf{T}b \mid y \ge \mathbf{0}, y^\mathsf{T}A \le c^\mathsf{T}\},\\
&\min\{c^\mathsf{T}x \mid Ax \ge b\} = \max\{y^\mathsf{T}b \mid y \ge \mathbf{0}, y^\mathsf{T}A = c^\mathsf{T}\}.
\end{aligned} \tag{5.11}$$

Any of these equalities holds if at least one of the two optima is finite (implying that both are finite).

A most general formulation is: let $A, B, C, D, E, F, G, H, K$ be matrices and let a, b, c, d, e, f be vectors; then

$$\begin{aligned}
&\max\{d^\mathsf{T}x + e^\mathsf{T}y + f^\mathsf{T}z \mid x \ge \mathbf{0}, z \le \mathbf{0},\\
&Ax + By + Cz \le a,\\
&Dx + Ey + Fz = b,\\
&Gx + Hy + Kz \ge c\}\\
&= \min\{u^\mathsf{T}a + v^\mathsf{T}b + w^\mathsf{T}c \mid u \ge \mathbf{0}, w \le \mathbf{0},\\
&u^\mathsf{T}A + v^\mathsf{T}D + w^\mathsf{T}G \ge d^\mathsf{T},\\
&u^\mathsf{T}B + v^\mathsf{T}E + w^\mathsf{T}H = e^\mathsf{T},\\
&u^\mathsf{T}C + v^\mathsf{T}F + w^\mathsf{T}K \le f^\mathsf{T}\},
\end{aligned} \tag{5.12}$$

provided that at least one of the two optima is finite (cf. Section 7.4 in Schrijver [1986b]).

So there is a one-to-one relation between constraints in a problem and variables in its dual problem. The objective function in one problem becomes the right-hand side in the dual problem. We survey these relations in the following table:

maximize	minimize
$\leq$ constraint	variable ≥ 0
$\geq$ constraint	variable ≤ 0
$=$ constraint	unconstrained variable
variable ≥ 0	$\geq$ constraint
variable ≤ 0	$\leq$ constraint
unconstrained variable	$=$ constraint
right-hand side	objective function
objective function	right-hand side

Some LP terminology. Linear programming concerns maximizing or minimizing a linear function $c^{\mathsf{T}}x$ over a polyhedron P. The polyhedron P is called the *feasible region*, and any vector in P a *feasible solution*. If the feasible region is nonempty, the problem is called *feasible*, and *infeasible* otherwise. The function $x \to c^{\mathsf{T}}x$ is called the *objective function* or the *cost function*. Any feasible solution attaining the optimum value is called an *optimum solution*. An inequality $c^{\mathsf{T}}x \leq \delta$ is called *tight* or *active* for some x^* if $c^{\mathsf{T}}x^* = \delta$.

Equations like (5.10), (5.11), and (5.12) are called *linear programming duality equations*. The minimization problem is called the *dual problem* of the maximization problem (which problem then is called the *primal problem*), and conversely. A feasible solution of the dual problem is called a *dual solution*.

Complementary slackness. The following *complementary slackness conditions* characterize optimality of a pair of feasible solutions x, y of the linear programs (5.10):

(5.13) x and y are optimum solutions if and only if $(Ax)_i = b_i$ for each i with $y_i > 0$.

Similar conditions can be formulated for other pairs of dual linear programs (cf. Section 7.9 in Schrijver [1986b]).

Carathéodory's theorem. A consequence of Carathéodory's theorem (Theorem 5.1 above) is:

Theorem 5.5. *If the optimum value in the LP problems* (5.10) *is finite, then the minimum is attained by a vector $y \geq \mathbf{0}$ such that the rows of A corresponding to positive components of y are linearly independent.*

(Corollary 7.1l in Schrijver [1986b].)

5.6. Faces, facets, and vertices

Let $P = \{x \mid Ax \leq b\}$ be a polyhedron in $\mathbb{R}^n$. If c is a nonzero vector and $\delta = \max\{c^{\mathsf{T}}x \mid Ax \leq b\}$, the affine hyperplane $\{x \mid c^{\mathsf{T}}x = \delta\}$ is called a *supporting hyperplane* of P. A subset F of P is called a *face* if $F = P$ or if $F = P \cap H$ for some supporting hyperplane H of P. So

(5.14) F is a face of $P \iff F$ is the set of optimum solutions of $\max\{c^{\mathsf{T}}x \mid Ax \leq b\}$ for some $c \in \mathbb{R}^n$.

An inequality $c^{\mathsf{T}}x \leq \delta$ is said to *determine* or to *induce* face F of P if

(5.15) $F = \{x \in P \mid c^{\mathsf{T}}x = \delta\}$.

Alternatively, F is a face of P if and only if

(5.16) $F = \{x \in P \mid A'x = b'\}$

for some subsystem $A'x \leq b'$ of $Ax \leq b$ (cf. Section 8.3 in Schrijver [1986b]). So any face of a nonempty polyhedron is a nonempty polyhedron. We say that a constraint $a^{\mathsf{T}}x \leq \beta$ from $Ax \leq b$ is *tight* or *active* in a face F if $a^{\mathsf{T}}x = \beta$ holds for each $x \in F$.

An inequality $a^{\mathsf{T}}x \leq \beta$ from $Ax \leq b$ is called an *implicit equality* if $Ax \leq b$ implies $a^{\mathsf{T}}x = \beta$. Then:

Theorem 5.6. *Let $P = \{x \mid Ax \leq b\}$ be a polyhedron in $\mathbb{R}^n$. Let $A'x \leq b'$ be the subsystem of implicit inequalities in $Ax \leq b$. Then* $\dim P = n - \mathrm{rank}A'$.

(Cf. Section 8.2 in Schrijver [1986b].)

A *facet* of P is an inclusionwise maximal face F of P with $F \neq P$. An inequality determining a facet is called *facet-determining* or *facet-inducing*. Any facet has dimension one less than the dimension of P.

A system $Ax \leq b$ is called *minimal* or *irredundant* if each proper subsystem $A'x \leq b'$ has a solution x not satisfying $Ax \leq b$. If $Ax \leq b$ is irredundant and P is full-dimensional, then $Ax \leq b$ is the unique minimal system determining P, up to multiplying inequalities by positive scalars.

If $Ax \leq b$ is irredundant, then there is a one-to-one relation between the facets F of P and those inequalities $a^{\mathsf{T}}x \leq \beta$ in $Ax \leq b$ that are not implicit equalities, given by:

(5.17) $F = \{x \in P \mid a^{\mathsf{T}}x = \beta\}$

(cf. Theorem 8.1 in Schrijver [1986b]). This implies that each face $F \neq P$ is the intersection of facets.

A face of $P = \{x \mid Ax \leq b\}$ is called a *minimal face* if it is an inclusionwise minimal face. Any minimal face is an affine subspace of $\mathbb{R}^n$, and all minimal faces of P are translates of each other. They all have dimension $n - \mathrm{rank}A$.

If each minimal face has dimension 0, P is called *pointed*. A *vertex* of P is an element z such that $\{z\}$ is a minimal face. A polytope is the convex hull of its vertices.

For any element z of $P = \{x \mid Ax \leq b\}$, let $A_z x \leq b_z$ be the system consisting of those inequalities from $Ax \leq b$ that are satisfied by z with equality. Then:

Theorem 5.7. *Let $P = \{x \mid Ax \leq b\}$ be a polyhedron in $\mathbb{R}^n$ and let $z \in P$. Then z is a vertex of P if and only if* $\mathrm{rank}(A_z) = n$.

An *edge* of P is a bounded face of dimension 1. It necessarily connects two vertices of P. Two vertices connected by an edge are called *adjacent*. An *extremal ray* is a face of dimension 1 that forms a halfline.

The 1-*skeleton* of a pointed polyhedron P is the union of the vertices, edges, and extremal rays of P. If P is a polytope, the 1-skeleton is a topological graph. The *diameter* of P is the diameter of the associated (combinatorial) graph.

The *Hirsch conjecture* states that a d-dimensional polytope with m facets has diameter at most $m - d$. Naddef [1989] proved this for polytopes with $0, 1$ vertices. We refer to Kalai [1997] for a survey of bounds on the diameter and on the number of pivot steps in linear programming.

5.7. Polarity

(For the results of this section, see Section 9.1 in Schrijver [1986b].) For any subset C of $\mathbb{R}^n$, the *polar* of C is

$$C^* := \{z \in \mathbb{R}^n \mid x^\mathsf{T} z \leq 1 \text{ for all } x \in C\}. \tag{5.18}$$

If C is a cone, then C^* is again a cone, the *polar cone* of C, and satisfies

$$C^* := \{z \in \mathbb{R}^n \mid x^\mathsf{T} z \leq 0 \text{ for all } x \in C\}. \tag{5.19}$$

Let C be a polyhedral cone; so $C = \{x \mid Ax \leq \mathbf{0}\}$ for some matrix A. Trivially, if C is generated by the vectors $x_1, \ldots, x_k$, then C^* is equal to the cone determined by the inequalities $x_i^\mathsf{T} z \leq 0$ for $i = 1, \ldots, k$. It is less trivial, and can be derived from Farkas' lemma, that:

(5.20) the polar cone C^* is equal to the cone generated by the transposes of the rows of A.

This implies

$$C^{**} = C \text{ for each polyhedral cone } C. \tag{5.21}$$

So there is a symmetric duality relation between finite sets of vectors generating a cone and finite sets of vectors generating its polar cone.

5.8. Blocking polyhedra

(For the results of this section, see Section 9.2 in Schrijver [1986b].) A duality relation similar to polarity holds between convex sets 'of blocking type', and also between convex sets 'of antiblocking type'. This was shown by Fulkerson [1970b,1971a,1972a], who found several applications in combinatorial optimization.

We say that a subset P of $\mathbb{R}^n$ is *up-monotone* if $x \in P$ and $y \geq x$ imply $y \in P$. Similarly, P is *down-monotone* if $x \in P$ and $y \leq x$ imply $y \in P$.

Moreover, P is *down-monotone in* $\mathbb{R}^n_+$ if $x \in P$ and $\mathbf{0} \leq y \leq x$ imply $y \in P$. For any $P \subseteq \mathbb{R}^n$ we define

$$P^{\uparrow} := \{y \in \mathbb{R}^n \mid \exists x \in P : y \geq x\} = P + \mathbb{R}^n_+ \text{ and} \quad P^{\downarrow} := \{y \in \mathbb{R}^n \mid \exists x \in P : y \leq x\} = P - \mathbb{R}^n_+. \tag{5.22}$$

$P^{\uparrow}$ is called the *dominant* of P. So P is up-monotone if and only if $P = P^{\uparrow}$, and P is down-monotone if and only if $P = P^{\downarrow}$.

We say that a convex set $P \subseteq \mathbb{R}^n$ is of *blocking type* if P is a closed convex up-monotone subset of $\mathbb{R}^n_+$. Each polyhedron P of blocking type is pointed. Moreover, P is a polyhedron of blocking type if and only if there exist vectors $x_1, \ldots, x_k \in \mathbb{R}^n_+$ such that

$$P = \text{conv.hull}\{x_1, \ldots, x_k\}^{\uparrow}; \tag{5.23}$$

and also, if and only if

$$P = \{x \in \mathbb{R}^n_+ \mid Ax \geq \mathbf{1}\} \tag{5.24}$$

for some nonnegative matrix A.

For any polyhedron P in $\mathbb{R}^n$, the *blocking polyhedron* $B(P)$ of P is defined by

$$B(P) := \{z \in \mathbb{R}^n_+ \mid x^{\mathsf{T}} z \geq 1 \text{ for each } x \in P\}. \tag{5.25}$$

Fulkerson [1970b,1971a] showed:

Theorem 5.8. *Let $P \subseteq \mathbb{R}^n_+$ be a polyhedron of blocking type. Then $B(P)$ is again a polyhedron of blocking type and $B(B(P)) = P$. Moreover, for any $x_1, \ldots, x_k \in \mathbb{R}^n_+$:*

(5.26) (5.23) *holds if and only if* $B(P) = \{z \in \mathbb{R}^n_+ \mid x_i^{\mathsf{T}} z \geq 1$ *for* $i = 1, \ldots, k\}$.

Here the only if part is trivial, while the if part requires Farkas' lemma.

Theorem 5.8 implies that for vectors $x_1, \ldots, x_k \in \mathbb{R}^n_+$ and $z_1, \ldots, z_d \in \mathbb{R}^n_+$ one has:

$$\text{conv.hull}\{x_1, \ldots, x_k\} + \mathbb{R}^n_+ = \{x \in \mathbb{R}^n_+ \mid z_j^{\mathsf{T}} x \geq 1 \text{ for } j = 1, \ldots, d\} \tag{5.27}$$

if and only if

$$\text{conv.hull}\{z_1, \ldots, z_d\} + \mathbb{R}^n_+ = \{z \in \mathbb{R}^n_+ \mid x_i^{\mathsf{T}} z \geq 1 \text{ for } i = 1, \ldots, k\}. \tag{5.28}$$

Two polyhedra P, R are called a *blocking pair (of polyhedra)* if they are of blocking type and satisfy $R = B(P)$. So if P, R is a blocking pair, then so is R, P.

5.9. Antiblocking polyhedra

(For the results of this section, see Section 9.3 in Schrijver [1986b].) The theory of antiblocking polyhedra is almost fully analogous to the blocking case and arises mostly by reversing inequality signs.

We say that a set $P \subseteq \mathbb{R}^n$ is of *antiblocking type* if P is a nonempty closed convex subset of $\mathbb{R}^n_+$ that is down-monotone in $\mathbb{R}^n_+$. Then P is a polyhedron of antiblocking type if and only if

$$P = \{x \in \mathbb{R}^n_+ \mid Ax \leq b\} \tag{5.29}$$

for some nonnegative matrix A and nonnegative vector b.

For any subset P of $\mathbb{R}^n$, the *antiblocking set* $A(P)$ of P is defined by

$$A(P) := \{z \in \mathbb{R}^n_+ \mid x^{\mathsf{T}} z \leq 1 \text{ for each } x \in P\}. \tag{5.30}$$

If $A(P)$ is a polyhedron we speak of the *antiblocking polyhedron*, and if $A(P)$ is a convex body, of the *antiblocking body*.

Fulkerson [1971a,1972a] showed:

Theorem 5.9. *Let $P \subseteq \mathbb{R}^n_+$ be of antiblocking type. Then $A(P)$ is again of antiblocking type and $A(A(P)) = P$.*

The antiblocking analogue of (5.26) is a little more complicated to formulate, but we need it only for full-dimensional polytopes. For any full-dimensional polytope $P \subseteq \mathbb{R}^n$ of antiblocking type and $x_1, \ldots, x_k \in \mathbb{R}^n_+$ we have:

$$P = \text{conv.hull}\{x_1, \ldots, x_k\}^{\downarrow} \cap \mathbb{R}^n_+ \text{ if and only if } A(P) = \{z \in \mathbb{R}^n_+ \mid x_i^{\mathsf{T}} z \leq 1 \text{ for } i = 1, \ldots, k\}. \tag{5.31}$$

Two convex sets P, R are called an *antiblocking pair (of polyhedra)* if they are of antiblocking type and satisfy $R = A(P)$. So if P, R is an antiblocking pair, then so is R, P.

5.10. Methods for linear programming

The simplex method was designed by Dantzig [1951b] to solve linear programming problems. It is in practice and on average quite efficient, but no polynomial-time worst-case running time bound has been proved (most of the pivot selection rules that have been proposed have been proved to take exponential time in the worst case).

The simplex method consists of finding a path in the 1-skeleton of the feasible region, ending at an optimum vertex (in preprocessing, the problem first is transformed to one with a pointed feasible region). An important issue when implementing this is that the LP problem is not given by vertices and

edges, but by linear inequalities, and that vertices are determined by a, not necessarily unique, 'basis' among the inequalities.

The first polynomial-time method for linear programming was given by Khachiyan [1979,1980], by adapting the 'ellipsoid method' for nonlinear programming of Shor [1970a,1970b,1977] and Yudin and Nemirovskiĭ [1976]. The method consists of finding a sequence of shrinking ellipsoids each containing at least one optimum solution, until we have an ellipsoid that is small enough so as to derive an optimum solution. The method however is practically quite infeasible.

Karmarkar [1984a,1984b] showed that 'interior point' methods can solve linear programming in polynomial time, and moreover that they have efficient implementations, competing with the simplex method. Interior point methods make a tour not along vertices and edges, but across the feasible region.

5.11. The ellipsoid method

While the ellipsoid method is practically infeasible, it turned out to have features that are useful for deriving complexity results in combinatorial optimization. Specifically, the ellipsoid method does not require listing all constraints of an LP problem a priori, but allows that they are generated when needed. In this way, one can derive the polynomial-time solvability of a number of combinatorial optimization problems. This should be considered as existence proofs of polynomial-time algorithms — the algorithms are not practical.

This application of the ellipsoid method was described by Karp and Papadimitriou [1980,1982], Padberg and Rao [1980], and Grötschel, Lovász, and Schrijver [1981]. The book by Grötschel, Lovász, and Schrijver [1988] is devoted to it. We refer to Chapter 6 of this book or to Chapter 14 of Schrijver [1986b] for proofs of the results that we survey below.

The ellipsoid method applies to classes of polyhedra (and more generally, classes of convex sets) which are described as follows.

Let Σ be a finite alphabet and let Π be a subset of the set Σ^* of words over Σ. In applications, we take for Π very simple sets like the set of strings representing a graph or the set of strings representing a digraph.

For each $\sigma \in \Pi$, let E_σ be a finite set and let P_σ be a rational polyhedron in $\mathbb{Q}^{E_\sigma}$. (When we apply this, E_σ is often the vertex set or the edge or arc set of the (di)graph represented by σ.) We make the following assumptions:

(5.32)
- (i) there is a polynomial-time algorithm that, given $\sigma \in \Sigma^*$, tests if σ belongs to Π and, if so, returns the set E_σ;
- (ii) there is a polynomial p such that, for each $\sigma \in \Pi$, P_σ is determined by linear inequalities each of size at most $p(\text{size}(\sigma))$.

Here the *size* of a rational linear inequality is proportional to the sum of the sizes of its components, where the *size* of a rational number p/q (for integers

p, q) is proportional to $\log(|p|+1) + \log q$. Condition (5.32)(ii) is equivalent to (cf. Theorem 10.2 in Schrijver [1986b]):

(5.33) there is a polynomial q such that, for each $\sigma \in \Pi$, we can write $P_\sigma = Q + C$, where Q is a polytope with vertices each of input size at most $q(\text{size}(\sigma))$ and where C is a cone generated by vectors each of input size at most $q(\text{size}(\sigma))$.

(The *input size*[7] of a vector is the sum of the sizes of its components.) In most applications, the existence of the polynomial p in (5.32)(ii) or of the polynomial q in (5.33) is obvious.

We did not specify how the polyhedra P_σ are given algorithmically. In applications, they might have an exponential number of vertices or facets, so listing them would not be an algorithmic option. To handle this, we formulate two, in a sense dual, problems. An algorithm for either of them would determine the polyhedra P_σ.

First, the *optimization problem for* $(P_\sigma \mid \sigma \in \Pi)$ is the problem:

(5.34) given: $\sigma \in \Pi$ and $c \in \mathbb{Q}^{E_\sigma}$,
find: $x \in P_\sigma$ maximizing $c^{\mathsf{T}}x$ over P_σ or $y \in \text{char.cone}(P_\sigma)$ with $c^{\mathsf{T}}y > 0$, if either of them exists.

Second, the *separation problem for* $(P_\sigma \mid \sigma \in \Pi)$ is the problem:

(5.35) given: $\sigma \in \Pi$ and $z \in \mathbb{Q}^{E_\sigma}$,
find: $c \in \mathbb{Q}^{E_\sigma}$ such that $c^{\mathsf{T}}x < c^{\mathsf{T}}z$ for all $x \in P_\sigma$ (if such a c exists).

So c gives a separating hyperplane if $z \notin P_\sigma$.

Then the ellipsoid method implies that these two problems are 'polynomial-time equivalent':

Theorem 5.10. *Let $\Pi \subseteq \Sigma^*$ and let $(P_\sigma \mid \sigma \in \Pi)$ satisfy* (5.32)*. Then the optimization problem for $(P_\sigma \mid \sigma \in \Pi)$ is polynomial-time solvable if and only if the separation problem for $(P_\sigma \mid \sigma \in \Pi)$ is polynomial-time solvable.*

(Cf. Theorem (6.4.9) in Grötschel, Lovász, and Schrijver [1988] or Corollary 14.1c in Schrijver [1986b].)

The equivalence in Theorem 5.10 makes that we call $(P_\sigma \mid \sigma \in \Pi)$ *polynomial-time solvable* if it satisfies (5.32) and the optimization problem (equivalently, the separation problem) for it is polynomial-time solvable.

Using simultaneous diophantine approximation based on the basis reduction method given by Lenstra, Lenstra, and Lovász [1982], Frank and Tardos [1985,1987] extended these results to strong polynomial-time solvability:

[7] We will use the term *size* of a vector for the sum of its components.

Theorem 5.11. *The optimization problem and the separation problem for any polynomial-time solvable system of polyhedra are solvable in strongly polynomial time.*

(Theorem (6.6.5) in Grötschel, Lovász, and Schrijver [1988].)

For polynomial-time solvable classes of polyhedra, the separation problem can be strengthened so as to obtain a facet as separating hyperplane:

Theorem 5.12. *Let $(P_\sigma \mid \sigma \in \Pi)$ be a polynomial-time solvable system of polyhedra. Then the following problem is strongly polynomial-time solvable:*

(5.36) given: $\sigma \in \Pi$ and $z \in \mathbb{Q}^{E_\sigma}$,
find: $c \in \mathbb{Q}^{E_\sigma}$ and $\delta \in \mathbb{Q}$ such that $c^\mathsf{T} z > \delta$ and such that $c^\mathsf{T} x \leq \delta$ is facet-inducing for P_σ (if it exists).

(Cf. Theorem (6.5.16) in Grötschel, Lovász, and Schrijver [1988].) Also a weakening of the separation problem turns out to be equivalent, under certain conditions. The *membership problem for* $(P_\sigma \mid \sigma \in \Pi)$ is the problem:

(5.37) given $\sigma \in \Pi$ and $z \in \mathbb{Q}^{E_\sigma}$, does z belong to P_σ?

Theorem 5.13. *Let $(P_\sigma \mid \sigma \in \Pi)$ be a system of full-dimensional polytopes satisfying* (5.32)*, such that there is a polynomial-time algorithm that gives for each $\sigma \in \Pi$ a vector in the interior of P_σ. Then $(P_\sigma \mid \sigma \in \Pi)$ is polynomial-time solvable if and only if the membership problem for $(P_\sigma \mid \sigma \in \Pi)$ is polynomial-time solvable.*

(This follows from Corollary (4.3.12) and Theorem (6.3.2) in Grötschel, Lovász, and Schrijver [1988].)

The theorems above imply:

Theorem 5.14. *Let $(P_\sigma \mid \sigma \in \Pi)$ and $(Q_\sigma \mid \sigma \in \Pi)$ be polynomial-time solvable classes of polyhedra, such that for each $\sigma \in \Pi$, the polyhedra P_σ and Q_σ are in the same space $\mathbb{R}^{E_\sigma}$. Then also $(P_\sigma \cap Q_\sigma \mid \sigma \in \Pi)$ and $(\text{conv.hull}(P_\sigma \cup Q_\sigma) \mid \sigma \in \Pi)$ are polynomial-time solvable.*

(Corollary 14.1d in Schrijver [1986b].)

Corollary 5.14a. *Let $(P_\sigma \mid \sigma \in \Pi)$ be a polynomial-time solvable system of polyhedra, all of blocking type. Then also the system of blocking polyhedra $(B(P_\sigma) \mid \sigma \in \Pi)$ is polynomial-time solvable.*

(Corollary 14.1e in Schrijver [1986b].) Similarly:

Corollary 5.14b. *Let $(P_\sigma \mid \sigma \in \Pi)$ be a polynomial-time solvable system of polyhedra, all of antiblocking type. Then also the system of antiblocking polyhedra $(A(P_\sigma) \mid \sigma \in \Pi)$ is polynomial-time solvable.*

(Corollary 14.1e in Schrijver [1986b].)

Also the following holds:

Theorem 5.15. *Let $(P_\sigma \mid \sigma \in \Pi)$ be a polynomial-time solvable system of polyhedra, where each P_σ is a polytope. Then the following problems are strongly polynomial-time solvable:*

(5.38)
- (i) *given $\sigma \in \Pi$, find an internal vector, a vertex, and a facet-inducing inequality of P_σ;*
- (ii) *given $\sigma \in \Pi$ and $x \in P_\sigma$, find affinely independent vertices $x_1, \ldots, x_k$ of P_σ and write x as a convex combination of $x_1, \ldots, x_k$;*
- (iii) *given $\sigma \in \Pi$ and $c \in \mathbb{R}^{E_\sigma}$, find facet-inducing inequalities $c_1^\mathsf{T} x \leq \delta_1, \ldots, c_k^\mathsf{T} x \leq \delta_k$ of P_σ with $c_1, \ldots, c_k$ linearly independent, and find $\lambda_1, \ldots, \lambda_k \geq 0$ such that $\lambda_1 c_1 + \cdots + \lambda_k c_k = c$ and $\lambda_1 \delta_1 + \cdots + \lambda_k \delta_k = \max\{c^\mathsf{T} x \mid x \in P_\sigma\}$ (i.e., find an optimum dual solution).*

(Corollary 14.1f in Schrijver [1986b].)

The ellipsoid method can be applied also to nonpolyhedral convex sets, in which case only approximative versions of the optimization and separation problems can be shown to be equivalent. We only need this in Chapter 67 on the convex body TH(G), where we refer to the appropriate theorem in Grötschel, Lovász, and Schrijver [1988].

5.12. Polyhedra and NP and co-NP

An appropriate polyhedral description of a combinatorial optimization problem relates to the question NP≠co-NP. More precisely, unless NP=co-NP, the polyhedra associated with an NP-complete problem cannot be described by 'certifiable' inequalities. (These insights go back to observations in the work of Edmonds of the 1960s.)

Again, let $(P_\sigma \mid \sigma \in \Pi)$ be a system of polyhedra satisfying (5.32). Consider the decision version of the optimization problem:

(5.39) given $\sigma \in \Pi$, $c \in \mathbb{Q}^{E_\sigma}$, and $k \in \mathbb{Q}$, is there an $x \in P_\sigma$ with $c^\mathsf{T} x > k$?

Then:

Theorem 5.16. *Problem* (5.39) *belongs to* co-NP *if and only if for each $\sigma \in \Pi$, there exists a collection $\mathcal{I}_\sigma$ of inequalities determining P_σ such that the problem:*

(5.40) *given $\sigma \in \Pi$, $c \in \mathbb{Q}^{E_\sigma}$, and $\delta \in \mathbb{Q}$, does $c^\mathsf{T} x \leq \delta$ belong to $\mathcal{I}_\sigma$,*

belongs to NP.

Proof. To see necessity, we can take for $\mathcal{I}_\sigma$ the collection of *all* valid inequalities for P_σ. Then co-NP-membership of (5.39) is equivalent of NP-membership of (5.40).

To see sufficiency, a negative answer to question (5.39) can be certified by giving inequalities $c_i^T x \leq \delta_i$ from $\mathcal{I}_\sigma$ and $\lambda_i \in \mathbb{Q}_+$ $(i = 1, \ldots, k)$ such that $c = \lambda_1 c_1 + \cdots + \lambda_k c_k$ and $\delta \geq \lambda_1 \delta_1 + \cdots + \lambda_k \delta_k$. As we can take $k \leq |E_\sigma|$, and as each inequality in $\mathcal{I}_\sigma$ has a polynomial-time checkable certificate (as (5.40) belongs to NP), this gives a polynomial-time checkable certificate for the negative answer. Hence (5.39) belongs to co-NP. ▮

This implies for NP-complete problems:

Corollary 5.16a. *Let* (5.39) *be* NP*-complete and suppose* NP≠co-NP*. For each* $\sigma \in \Pi$*, let* $\mathcal{I}_\sigma$ *be a collection of inequalities determining* P_σ*. Then problem* (5.40) *does not belong to* NP*.*

Proof. If problem (5.40) would belong to NP, then by Theorem 5.16, problem (5.39) belongs to co-NP. If (5.39) is NP-complete, this implies NP=co-NP. ▮

Roughly speaking, this implies that if (5.39) is NP-complete and NP≠co-NP, then P_σ has 'difficult' facets, that is, facets which have no polynomial-time checkable certificate of validity for P_σ.

(Related work on the complexity of facets was reported in Karp and Papadimitriou [1980,1982] and Papadimitriou and Yannakakis [1982,1984].)

5.13. Primal-dual methods

As a generalization of similar methods for network flow and transportation problems, Dantzig, Ford, and Fulkerson [1956] designed the 'primal-dual method' for linear programming. The general idea is as follows. Starting with a dual feasible solution y, the method searches for a primal feasible solution x satisfying the complementary slackness condition with respect to y. If such a primal feasible solution x is found, x and y form a pair of optimum solutions (by (5.13)). If no such primal solution is found, the method prescribes a modification of y, after which the method iterates.

The problem now is how to find a primal feasible solution x satisfying the complementary slackness condition, and how to modify the dual solution y if no such primal solution is found. For general linear programs this problem can be seen to amount to another linear program, generally simpler than the original linear program. To solve the simpler linear program we could use any LP method. In many combinatorial applications, however, this simpler linear program is a simpler combinatorial optimization problem, for which direct

methods are available. Thus, if we can describe a combinatorial optimization problem as a linear program, the primal-dual method gives us a scheme for reducing one combinatorial problem to an easier combinatorial problem. The efficiency of the method depends on the complexity of the easier problem and on the number of primal-dual iterations.

We describe the primal-dual method more precisely. Suppose that we wish to solve the LP problem

$$\min\{c^\mathsf{T}x \mid x \geq \mathbf{0}, Ax = b\}, \tag{5.41}$$

where A is an $m \times n$ matrix, with columns $a_1, \ldots, a_n$, and where $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. The dual problem is

$$\max\{y^\mathsf{T}b \mid y^\mathsf{T}A \leq c^\mathsf{T}\}. \tag{5.42}$$

The primal-dual method consists of repeating the following *primal-dual iteration*. Suppose that we have a feasible solution y_0 for problem (5.42). Let A' be the submatrix of A consisting of those columns a_j of A for which $y_0^\mathsf{T}a_j = c_j$ holds. To find a feasible primal solution satisfying the complementary slackness, solve the *restricted linear program*

$$x' \geq \mathbf{0}, A'x' = b. \tag{5.43}$$

If such an x' exists, by adding components 0, we obtain a vector $x \geq \mathbf{0}$ such that $Ax = b$ and such that $x_j = 0$ if $y_0^\mathsf{T}a_j < c_j$. By complementary slackness ((5.13)), it follows that x and y_0 are optimum solutions for problems (5.41) and (5.42).

On the other hand, if no x' satisfying (5.43) exists, by Farkas' lemma (Corollary 5.3a), there exists a y' such that ${y'}^\mathsf{T}A' \leq 0$ and ${y'}^\mathsf{T}b > 0$. Let α be the largest real number satisfying

$$(y_0 + \alpha y')^\mathsf{T}A \leq c^\mathsf{T}. \tag{5.44}$$

(Note that $\alpha > 0$.) Reset $y_0 := y_0 + \alpha y'$, and start the iteration anew. (If $\alpha = \infty$, (5.42) is unbounded, hence (5.41) is infeasible.)

This describes the primal-dual method. It reduces problem (5.41) to (5.43), which often is an easier problem.

The primal-dual method can equally well be considered as a *gradient method*. Suppose that we wish to solve problem (5.42), and we have a feasible solution y_0. This y_0 is not optimum if and only if there exists a vector y' such that ${y'}^\mathsf{T}b > 0$ and y' is a *feasible direction* at y_0 (that is, $(y_0 + \alpha y')^\mathsf{T}A \leq c^\mathsf{T}$ for some $\alpha > 0$). If we let A' consist of those columns of A in which $y_0^\mathsf{T}A \leq c^\mathsf{T}$ has equality, then y' is a feasible direction if and only if ${y'}^\mathsf{T}A' \leq 0$. So y' can be found by solving (5.43).

5.14. Integer linear programming

A vector $x \in \mathbb{R}^n$ is called *integer* if each component is an integer, i.e., if x belongs to $\mathbb{Z}^n$. Many combinatorial optimization problems can be described as

maximizing a linear function $c^{\mathsf{T}}x$ over the *integer* vectors in some polyhedron $P = \{x \mid Ax \leq b\}$.

So this type of problems can be described as:

$$\max\{c^{\mathsf{T}}x \mid Ax \leq b; x \in \mathbb{Z}^n\}. \tag{5.45}$$

Such problems are called *integer linear programming*, or *ILP*, problems. They consist of maximizing a linear function over the intersection $P \cap \mathbb{Z}^n$ of a polyhedron P with the set $\mathbb{Z}^n$ of integer vectors.

Clearly, always the following inequality holds:

$$\max\{c^{\mathsf{T}}x \mid Ax \leq b; x \text{ integer}\} \leq \max\{c^{\mathsf{T}}x \mid Ax \leq b\}. \tag{5.46}$$

It is easy to make an example where strict inequality holds. This implies, that generally one will have strict inequality in the following duality relation:

$$\begin{aligned}&\max\{c^{\mathsf{T}}x \mid Ax \leq b; x \text{ integer}\}\\ &\leq \min\{y^{\mathsf{T}}b \mid y \geq \mathbf{0}; y^{\mathsf{T}}A = c^{\mathsf{T}}; y \text{ integer}\}.\end{aligned} \tag{5.47}$$

No polynomial-time algorithm is known to exist for solving an integer linear programming problem in general. In fact, the general integer linear programming problem is NP-complete (since the satisfiability problem is easily transformed to an integer linear programming problem). However, for special classes of integer linear programming problems, polynomial-time algorithms have been found. These classes often come from combinatorial problems.

5.15. Integer polyhedra

A polyhedron P is called an *integer polyhedron* if it is the convex hull of the integer vectors contained in P. This is equivalent to: P is rational and each face of P contains an integer vector. So a polytope P is integer if and only if each vertex of P is integer. If a polyhedron $P = \{x \mid Ax \leq b\}$ is integer, then the linear programming problem

$$\max\{c^{\mathsf{T}}x \mid Ax \leq b\} \tag{5.48}$$

has an integer optimum solution if it is finite. Hence, in that case,

$$\max\{c^{\mathsf{T}}x \mid Ax \leq b;\ x \text{ integer}\} = \max\{c^{\mathsf{T}}x \mid Ax \leq b\}. \tag{5.49}$$

This in fact characterizes integer polyhedra, since:

Theorem 5.17. *Let P be a rational polyhedron in $\mathbb{Q}^n$. Then P is integer if and only if for each $c \in \mathbb{Q}^n$, the linear programming problem* $\max\{c^{\mathsf{T}}x \mid Ax \leq b\}$ *has an integer optimum solution if it is finite.*

A stronger characterization is (Edmonds and Giles [1977]):

Theorem 5.18. *A rational polyhedron P in $\mathbb{Q}^n$ is integer if and only if for each $c \in \mathbb{Z}^n$ the value of* $\max\{c^{\mathsf{T}}x \mid x \in P\}$ *is an integer if it is finite.*

(Corollary 22.1a in Schrijver [1986b].) We also will use the following observation:

Theorem 5.19. *Let P be an integer polyhedron in $\mathbb{R}^n_+$ with $P+\mathbb{R}^n_+ = P$ and let $c \in \mathbb{Z}^n_+$ be such that $x \leq c$ for each vertex x of P. Then $P \cap \{x \mid x \leq c\}$ is an integer polyhedron again.*

Proof. Let $Q := P \cap \{x \mid x \leq c\}$ and let R be the convex hull of the integer vectors in Q. We must show that $Q \subseteq R$.

Let $x \in Q$. As $P = R + \mathbb{R}^n_+$ there exists a $y \in R$ with $y \leq x$. Choose such a y with $y_1 + \cdots + y_n$ maximal. Suppose that $y_i < x_i$ for some component i. Since $y \in R$, y is a convex combination of integer vectors in Q. Since $y_i < x_i \leq c_i$, at least one of these integer vectors, z say, has $z_i < c_i$. But then the vector $z' := z + \chi^i$ belongs to R. Hence we could increase y_i, contradicting the maximality of y. ∎

We call a polyhedron P *box-integer* if $P \cap \{x \mid d \leq x \leq c\}$ is an integer polyhedron for each choice of integer vectors d, c. The set $\{x \mid d \leq x \leq c\}$ is called a *box*.

A $0,1$ *polytope* is a polytope with all vertices being 0,1 vectors.

5.16. Totally unimodular matrices

Total unimodularity of matrices is an important tool in integer programming. A matrix A is called *totally unimodular* if each square submatrix of A has determinant equal to 0, $+1$, or -1. In particular, each entry of a totally unimodular matrix is 0, $+1$, or -1.

An alternative way of characterizing total unimodularity is by requiring that the matrix is integer and that each nonsingular submatrix has an integer inverse matrix. This implies the following easy, but fundamental result:

Theorem 5.20. *Let A be a totally unimodular $m \times n$ matrix and let $b \in \mathbb{Z}^m$. Then the polyhedron*

$$P := \{x \mid Ax \leq b\} \tag{5.50}$$

is integer.

(Cf. Theorem 19.1 in Schrijver [1986b].) It follows that each linear programming problem with integer data and totally unimodular constraint matrix has integer optimum primal and dual solutions:

Corollary 5.20a. *Let A be a totally unimodular $m \times n$ matrix, let $b \in \mathbb{Z}^m$, and let $c \in \mathbb{Z}^n$. Then both optima in the LP duality equation*

$$\max\{c^\mathsf{T} x \mid Ax \leq b\} = \min\{y^\mathsf{T} b \mid y \geq \mathbf{0}, y^\mathsf{T} A = c^\mathsf{T}\} \tag{5.51}$$

have integer optimum solutions (if the optima are finite).

(Corollary 19.1a in Schrijver [1986b].) Hoffman and Kruskal [1956] showed that this property is close to a characterization of total unimodularity.

Corollary 5.20a implies:

Corollary 5.20b. *Let A be an $m \times n$ matrix, let $b \in \mathbb{Z}^m$, and let $c \in \mathbb{R}^n$. Suppose that*

$$\max\{c^\mathsf{T} x \mid x \geq \mathbf{0}, Ax \leq b\} \tag{5.52}$$

has an optimum solution x^* such that the columns of A corresponding to positive components of x^* form a totally unimodular matrix. Then (5.52) *has an integer optimum solution.*

Proof. Since x^* is an optimum solution, we have

$$\max\{c^\mathsf{T} x \mid x \geq \mathbf{0}, Ax \leq b\} = \max\{{c'}^\mathsf{T} x' \mid x' \geq \mathbf{0}, A'x' \leq b\}, \tag{5.53}$$

where A' and c' are the parts of A and c corresponding to the support of x^*. As A' is totally unimodular, the right-hand side maximum in (5.53) has an integer optimum solution ${x'}^*$. Extending ${x'}^*$ by components 0, we obtain an integer optimum solution of the left-hand side maximum in (5.53). ∎

We will use the following characterization of Ghouila-Houri [1962b] (cf. Theorem 19.3 in Schrijver [1986b]):

Theorem 5.21. *A matrix M is totally unimodular if and only if each collection R of rows of M can be partitioned into classes R_1 and R_2 such that the sum of the rows in R_1, minus the sum of the rows in R_2, is a vector with entries $0, \pm 1$ only.*

5.17. Total dual integrality

Edmonds and Giles [1977] introduced the powerful notion of total dual integrality. It is not only useful as a tool to derive combinatorial min-max relation, but also it gives an efficient way of expressing a whole bunch of min-max relations simultaneously.

A system $Ax \leq b$ in n dimensions is called *totally dual integral*, or just *TDI*, if A and b are rational and for each $c \in \mathbb{Z}^n$, the dual of maximizing $c^\mathsf{T} x$ over $Ax \leq b$:

$$\min\{y^\mathsf{T} b \mid y \geq \mathbf{0}, y^\mathsf{T} A = c^\mathsf{T}\} \tag{5.54}$$

has an integer optimum solution y, if it is finite.

By extension, a system $A'x \leq b', A''x = b''$ is defined to be TDI if the system $A'x \leq b', A''x \leq b'', -A''x \leq -b''$ is TDI. This is equivalent to requiring that A', A'', b', b'' are rational and for each $c \in \mathbb{Z}^n$ the dual of maximizing $c^\mathsf{T}x$ over $A'x \leq b'$, $A''x = b''$ has an integer optimum solution, if finite.

Problem (5.54) is the problem dual to $\max\{c^\mathsf{T}x \mid Ax \leq b\}$, and Edmonds and Giles showed that total dual integrality implies that also this primal problem has an integer optimum solution, if b is integer. In fact, they showed Theorem 5.18, which implies (since if (5.54) has an integer optimum solution, the optimum value is an integer):

Theorem 5.22. *If $Ax \leq b$ is TDI and b is integer, then $Ax \leq b$ determines an integer polyhedron.*

So total dual integrality implies 'primal integrality'. For combinatorial applications, the following observation is useful:

Theorem 5.23. *Let A be a nonnegative integer $m \times n$ matrix such that the system $x \geq \mathbf{0}, Ax \geq \mathbf{1}$ is TDI. Then also the system $\mathbf{0} \leq x \leq \mathbf{1}, Ax \geq \mathbf{1}$ is TDI.*

Proof. Choose $c \in \mathbb{Z}^n$. Let c_+ arise from c by setting negative components to 0. By the total dual integrality of $x \geq \mathbf{0}$, $Ax \geq \mathbf{1}$, there exist integer optimum solutions x, y of

$$\min\{c_+^\mathsf{T}x \mid x \geq \mathbf{0}, Ax \geq \mathbf{1}\} = \max\{y^\mathsf{T}\mathbf{1} \mid y \geq \mathbf{0}, y^\mathsf{T}A \leq c_+^\mathsf{T}\}. \tag{5.55}$$

As A is nonnegative and integer and as $c_+ \geq \mathbf{0}$, we may assume that $x \leq \mathbf{1}$. Moreover, we can assume that $x_i = 1$ if $(c_+)_i = 0$, that is, if $c_i \leq 0$.

Let $z := c - c_+$. So $z \leq \mathbf{0}$. We show that x, y, z are optimum solutions of

$$\begin{aligned}&\min\{c^\mathsf{T}x \mid \mathbf{0} \leq x \leq \mathbf{1}, Ax \geq \mathbf{1}\}\\ &= \max\{y^\mathsf{T}\mathbf{1} + z^\mathsf{T}\mathbf{1} \mid y \geq \mathbf{0}, z \leq \mathbf{0}, y^\mathsf{T}A + z^\mathsf{T} \leq c^\mathsf{T}\}.\end{aligned} \tag{5.56}$$

Indeed, x is feasible, as $\mathbf{0} \leq x \leq \mathbf{1}$ and $Ax \geq \mathbf{1}$. Moreover, y, z is feasible, as $y^\mathsf{T}A + z^\mathsf{T} \leq c_+^\mathsf{T} + z^\mathsf{T} = c^\mathsf{T}$. Optimality of x, y, z follows from

$$c^\mathsf{T}x = c_+^\mathsf{T}x + z^\mathsf{T}x = y^\mathsf{T}\mathbf{1} + z^\mathsf{T}x = y^\mathsf{T}\mathbf{1} + z^\mathsf{T}\mathbf{1}. \tag{5.57}$$

∎

In certain cases, to obtain total dual integrality one can restrict oneself to nonnegative objective functions:

Theorem 5.24. *Let A be a nonnegative $m \times n$ matrix and let $b \in \mathbb{R}_+^m$. Then $x \geq \mathbf{0}, Ax \leq b$ is TDI if and only if $\min\{y^\mathsf{T}b \mid y \geq \mathbf{0}, y^\mathsf{T}A \geq c^\mathsf{T}\}$ is attained by an integer optimum solution (if finite), for each $c \in \mathbb{Z}_+^n$.*

Proof. Necessity is trivial. To see sufficiency, let $c \in \mathbb{Z}^n$ with $\min\{y^\mathsf{T}b \mid y \geq \mathbf{0}, y^\mathsf{T}A \geq c^\mathsf{T}\}$ finite. Let it be attained by y. Let c_+ arise from c by setting negative components to 0. Then

(5.58) $$\min\{y^\mathsf{T} b \mid y \geq \mathbf{0}, y^\mathsf{T} A \geq c_+^\mathsf{T}\} = \min\{y^\mathsf{T} b \mid y \geq \mathbf{0}, y^\mathsf{T} A \geq c^\mathsf{T}\},$$

since $y^\mathsf{T} A \geq \mathbf{0}$ if $y \geq \mathbf{0}$. As the first minimum has an integer optimum solution, also the second minimum has an integer optimum solution. ∎

Total dual integrality is maintained under setting an inequality to an equality (Theorem 22.2 in Schrijver [1986b]):

Theorem 5.25. *Let $Ax \leq b$ be TDI and let $A'x \leq b'$ arise from $Ax \leq b$ by adding $-a^\mathsf{T} x \leq -\beta$ for some inequality $a^\mathsf{T} x \leq \beta$ in $Ax \leq b$. Then also $A'x \leq b'$ is TDI.*

Total dual integrality is also maintained under translation of the solution set, as follows directly from the definition of total dual integrality:

Theorem 5.26. *If $Ax \leq b$ is TDI and $w \in \mathbb{R}^n$, then $Ax \leq b - Aw$ is TDI.*

For future reference, we prove:

Theorem 5.27. *Let $A_{11}, A_{12}, A_{21}, A_{22}$ be matrices and let b_1, b_2 be column vectors, such that the system*

(5.59) $$\begin{aligned} &A_{1,1}x_1 + A_{1,2}x_2 = b_1,\\ &A_{2,1}x_1 + A_{2,2}x_2 \leq b_2 \end{aligned}$$

is TDI and such that $A_{1,1}$ is nonsingular. Then also the system

(5.60) $$(A_{2,2} - A_{2,1}A_{1,1}^{-1}A_{1,2})x_2 \leq b_2 - A_{2,1}A_{1,1}^{-1}b_1$$

is TDI.

Proof. We may assume that $b_1 = \mathbf{0}$, since by Theorem 5.26 total dual integrality is invariant under replacing (5.59) by

(5.61) $$\begin{aligned} &A_{1,1}x_1 + A_{1,2}x_2 = b_1 - A_{1,1}A_{1,1}^{-1}b_1 = \mathbf{0},\\ &A_{2,1}x_1 + A_{2,2}x_2 \leq b_2 - A_{2,1}A_{1,1}^{-1}b_1. \end{aligned}$$

Let x_2 minimize $c^\mathsf{T} x_2$ over (5.60), for some integer vector c of appropriate dimension. Define $x_1 := -A_{1,1}^{-1}A_{1,2}x_2$. Then x_1, x_2 minimizes $c^\mathsf{T} x_2$ over (5.59), since any solution x_1', x_2' of (5.59) satisfies $x_1' = -A_{1,1}^{-1}A_{1,2}x_2'$, and therefore x_2' satisfies (5.60); hence $c^\mathsf{T} x_2' \geq c^\mathsf{T} x_2$.

Let y_1, y_2 be an integer optimum solution of the problem dual to maximizing $c^\mathsf{T} x_2$ over (5.59). So y_1, y_2 satisfy

(5.62) $$y_1^\mathsf{T} A_{1,1} + y_2^\mathsf{T} A_{2,1} = \mathbf{0},\ y_1^\mathsf{T} A_{1,2} + y_2^\mathsf{T} A_{2,2} = c^\mathsf{T},\ y_2^\mathsf{T} b_2 = c^\mathsf{T} x_2.$$

Hence

(5.63) $$y_2^\mathsf{T}(A_{2,2} - A_{2,1}A_{1,1}^{-1}A_{1,2}) = y_2^\mathsf{T} A_{2,2} + y_1^\mathsf{T} A_{1,2} = c^\mathsf{T}$$

and

(5.64) $y_2^\mathsf{T} b_2 = c^\mathsf{T} x_2$.

So y_2 is an integer optimum solution of the problem dual to maximizing $c^\mathsf{T} x_2$ over (5.60). ▮

This has as consequence (where a_0 is a column vector):

Corollary 5.27a. *If $x_0 = \beta$, $a_0 x_0 + Ax \leq b$ is TDI, then $Ax \leq b - \beta a_0$ is TDI.*

Proof. This is a special case of Theorem 5.27. ▮

We also have:

Theorem 5.28. *Let $A = [a_1 \;\; a_2 \;\; A'']$ be an integer $m \times n$ matrix and let $b \in \mathbb{R}^m$. Let A' be the $m \times (n-1)$ matrix $[a_1 + a_2 \;\; A'']$. Then $A'x' \leq b$ is TDI if and only if $Ax \leq b, x_1 - x_2 = 0$ is TDI.*

Proof. To see necessity, choose $c \in \mathbb{Z}^n$. Let $c' := (c_1 + c_2, c_3, \ldots, c_n)^\mathsf{T}$. Then

(5.65) $\mu := \max\{c^\mathsf{T} x \mid Ax \leq b, x_1 - x_2 = 0\} = \max\{{c'}^\mathsf{T} x' \mid A'x' \leq b\}$.

Let $y \in \mathbb{Z}_+^m$ be an integer optimum dual solution of the second maximum. So $y^\mathsf{T} A' = c'$ and $y^\mathsf{T} b = \mu$. Then $y^\mathsf{T} a_1 + y^\mathsf{T} a_2 = c_1 + c_2$. Hence $y^\mathsf{T} A = c^\mathsf{T} + \lambda(1, -1, 0, \ldots, 0)$ for some $\lambda \in \mathbb{Z}$. So y, λ form an integer optimum dual solution of the first maximum.

To see sufficiency, choose $c' = (c_2, \ldots, c_n)^\mathsf{T} \in \mathbb{Z}^{n-1}$. Define $c := (0, c_2, \ldots, c_n)^\mathsf{T}$. Again we have (5.65). Let $y \in \mathbb{Z}_+^m$, $\lambda \in \mathbb{Z}$ constitute an integer optimum dual solution of the first maximum, where λ corresponds to the constraint $x_1 - x_2 = 0$. So $y^\mathsf{T} A + \lambda(1, -1, 0, \ldots, 0) = c$ and $y^\mathsf{T} b = \mu$. Hence $y^\mathsf{T} A' = c^\mathsf{T}$, and therefore, y is an integer optimum dual solution of the second maximum. ▮

Let A be a rational $m \times n$ matrix and let $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$. Consider the following series of inequalities (where a vector z is *half-integer* if $2z$ is integer):

(5.66)
$$\begin{aligned}
&\max\{c^\mathsf{T} x \mid Ax \leq b, x \text{ integer}\} \leq \max\{c^\mathsf{T} x \mid Ax \leq b\}\\
&= \min\{y^\mathsf{T} b \mid y \geq \mathbf{0}, y^\mathsf{T} A = c^\mathsf{T}\}\\
&\leq \min\{y^\mathsf{T} b \mid y \geq \mathbf{0}, y^\mathsf{T} A = c^\mathsf{T}, y \text{ half-integer}\}\\
&\leq \min\{y^\mathsf{T} b \mid y \geq \mathbf{0}, y^\mathsf{T} A = c^\mathsf{T}, y \text{ integer}\}.
\end{aligned}$$

Under certain circumstances, equality in the last inequality implies equality throughout:

Theorem 5.29. *Let $Ax \leq b$ be a system with A and b rational. Then $Ax \leq b$ is TDI if and only if*

(5.67) $\min\{y^\mathsf{T} b \mid y \geq \mathbf{0}, y^\mathsf{T} A = c^\mathsf{T}, y \text{ half-integer}\}$

is finite and is attained by an integer optimum solution y, for each integer vector c with $\max\{c^\mathsf{T}x \mid Ax \leq b\}$ *finite.*

Proof. Necessity follows directly from (5.66). To see sufficiency, choose $c \in \mathbb{Z}^n$ with $\max\{c^\mathsf{T}x \mid Ax \leq b\}$ finite. We must show that $\min\{y^\mathsf{T}b \mid y \geq \mathbf{0}, y^\mathsf{T}A = c^\mathsf{T}\}$ is attained by an integer optimum solution.

For each $k \geq 1$, define

$$\alpha_k = \min\{y^\mathsf{T}b \mid y \geq \mathbf{0}, y^\mathsf{T}A = kc^\mathsf{T}, y \text{ integer}\}. \tag{5.68}$$

This is well-defined, as $\max\{kc^\mathsf{T}x \mid Ax \leq b\}$ is finite.

The condition in the theorem gives that, for each $t \geq 0$,

$$\frac{\alpha_{2^t}}{2^t} = \alpha_1. \tag{5.69}$$

This can be shown by induction on t, the case $t = 0$ being trivial. If $t \geq 1$, then

$$\begin{aligned} \alpha_{2^t} &= \min\{y^\mathsf{T}b \mid y^\mathsf{T}A = 2^t c^\mathsf{T}, y \in \mathbb{Z}^m_+\} \\ &= 2\min\{y^\mathsf{T}b \mid y^\mathsf{T}A = 2^{t-1}c^\mathsf{T}, y \in \tfrac{1}{2}\mathbb{Z}^m_+\} \\ &= 2\min\{y^\mathsf{T}b \mid y^\mathsf{T}A = 2^{t-1}c^\mathsf{T}, y \in \mathbb{Z}^m_+\} = 2\alpha_{2^{t-1}}, \end{aligned} \tag{5.70}$$

implying (5.69) by induction.

Now $\alpha_{k+l} \leq \alpha_k + \alpha_l$ for all k, l. Hence we can apply Fekete's lemma, and get:

$$\begin{aligned} \min\{y^\mathsf{T}b \mid y \geq \mathbf{0}, y^\mathsf{T}A = c^\mathsf{T}\} &= \min_k \frac{\alpha_k}{k} = \lim_{k\to\infty} \frac{\alpha_k}{k} = \lim_{t\to\infty} \frac{\alpha_{2^t}}{2^t} \\ &= \alpha_1. \end{aligned} \tag{5.71}$$

∎

The following analogue of Carathéodory's theorem holds (Cook, Fonlupt, and Schrijver [1986]):

Theorem 5.30. *Let $Ax \leq b$ be a totally dual integral system in n dimensions and let $c \in \mathbb{Z}^n$. Then $\min\{y^\mathsf{T}b \mid y \geq \mathbf{0}, y^\mathsf{T}A \geq c^\mathsf{T}\}$ has an integer optimum solution y with at most $2n-1$ nonzero components.*

(Theorem 22.12 in Schrijver [1986b].)

We also will need the following substitution property:

Theorem 5.31. *Let $A_1x \leq b_1, A_2x \leq b_2$ be a TDI system with A_1 integer, and let $A_1' \leq b_1'$ be a TDI system with*

$$\{x \mid A_1x \leq b_1\} = \{x \mid A_1'x \leq b_1'\}. \tag{5.72}$$

Then the system $A_1'x \leq b_1', A_2x \leq b_2$ is TDI.

Proof. Let $c \in \mathbb{Z}^n$ with

$$\begin{aligned} &\max\{c^\mathsf{T}x \mid A_1'x \leq b_1', A_2x \leq b_2\} \\ &= \min\{y^\mathsf{T}b_1' + z^\mathsf{T}b_2 \mid y, z \geq \mathbf{0}, y^\mathsf{T}A_1' + z^\mathsf{T}A_2 = c^\mathsf{T}\} \end{aligned} \tag{5.73}$$

finite. By (5.72), also

$$\begin{aligned}(5.74)\qquad &\max\{c^{\mathsf T}x \mid A_1x \le b_1, A_2x \le b_2\}\\ &= \min\{y^{\mathsf T}b_1 + z^{\mathsf T}b_2 \mid y, z \ge \mathbf{0}, y^{\mathsf T}A_1 + z^{\mathsf T}A_2 = c^{\mathsf T}\}\end{aligned}$$

is finite. Hence, since $A_1x \le b_1, A_2x \le b_2$ is TDI, the minimum in (5.74) has an integer optimum solution y, z. Set $d := y^{\mathsf T}A_1$. Then, as d is an integer vector,

$$\begin{aligned}(5.75)\qquad &y^{\mathsf T}b_1 = \min\{u^{\mathsf T}b_1 \mid u \ge \mathbf{0}, u^{\mathsf T}A_1 = d^{\mathsf T}\}\\ &= \max\{d^{\mathsf T}x \mid A_1x \le b_1\} = \max\{d^{\mathsf T}x \mid A_1'x \le b_1'\}\\ &= \min\{v^{\mathsf T}b_1' \mid v \ge \mathbf{0}, v^{\mathsf T}A_1' = d^{\mathsf T}\}\end{aligned}$$

is finite. Hence, since $A_1'x \le b_1'$ is TDI, the last minimum in (5.75) has an integer optimum solution v. Then v, z is an integer optimum solution of the minimum in (5.73). ∎

A system $Ax \le b$ is called *totally dual half-integral* if A and b are rational and for each $c \in \mathbb{Z}^n$, the dual of maximizing $c^{\mathsf T}x$ over $Ax \le b$ has a half-integer optimum solution, if it is finite. Similarly, $Ax \le b$ is called *totally dual quarter-integral* if A and b are rational and for each $c \in \mathbb{Z}^n$, the dual of maximizing $c^{\mathsf T}x$ over $Ax \le b$ has a quarter-integer optimum solution y, if it is finite.

5.18. Hilbert bases and minimal TDI systems

For any $X \subseteq \mathbb{R}^n$ we denote

$$(5.76)\qquad \text{lattice}X := \{\lambda_1x_1 + \cdots + \lambda_kx_k \mid k \ge 0, \lambda_1, \dots, \lambda_k \in \mathbb{Z}, x_1, \dots, x_k \in X\}.$$

A subset L of $\mathbb{R}^n$ is called a *lattice* if $L = \text{lattice}X$ for some base X of $\mathbb{R}^n$. So for general X, lattice$X$ need not be a lattice.

The *dual lattice* of X is, by definition:

$$(5.77)\qquad \{x \in \mathbb{R}^n \mid y^{\mathsf T}x \in \mathbb{Z} \text{ for each } y \in X\}.$$

Again, this need not be a lattice in the proper sense.

A set X of vectors is called a *Hilbert base* if each vector in lattice$X \cap$cone$X$ is a nonnegative integer combination of vectors in X. The Hilbert base is called *integer* if it consists of integer vectors only.

One may show:

(5.78) Each rational polyhedral cone C is generated by an integer Hilbert base. If C is pointed, there exists a unique inclusionwise minimal integer Hilbert base generating C.

(Theorem 16.4 in Schrijver [1986b].)

There is a close relation between Hilbert bases and total dual integrality:

Theorem 5.32. *A rational system $Ax \leq b$ is TDI if and only if for each face F of $P := \{x \mid Ax \leq b\}$, the rows of A which are active in F form a Hilbert base.*

(Theorem 22.5 in Schrijver [1986b].)

(5.78) and Theorem 5.32 imply (Giles and Pulleyblank [1979], Schrijver [1981b]):

Theorem 5.33. *Each rational polyhedron P is determined by a TDI system $Ax \leq b$ with A integer. If moreover P is full-dimensional, there exists a unique minimal such system.*

(Theorem 22.6 in Schrijver [1986b].)

5.19. The integer rounding and decomposition properties

A system $Ax \leq b$ is said to have the *integer rounding property* if $Ax \leq b$ is rational and

$$\begin{aligned}(5.79)\qquad &\min\{y^{\mathsf{T}}b \mid y \geq \mathbf{0}, y^{\mathsf{T}}A = c^{\mathsf{T}}, y \text{ integer}\}\\ &= \lceil \min\{y^{\mathsf{T}}b \mid y \geq \mathbf{0}, y^{\mathsf{T}}A = c^{\mathsf{T}}\}\rceil\end{aligned}$$

for each integer vector c for which $\min\{y^{\mathsf{T}}b \mid y \geq \mathbf{0}, y^{\mathsf{T}}A = c^{\mathsf{T}}\}$ is finite. So any TDI system has the integer rounding property.

A polyhedron P is said to have the *integer decomposition property* if for each natural number k, each integer vector in $k \cdot P$ is the sum of k integer vectors in P.

Baum and Trotter [1978] showed that an integer matrix A is totally unimodular if and only if the polyhedron $\{x \mid x \geq \mathbf{0}, Ax \leq b\}$ has the integer decomposition property for each integer vector b. In another paper, Baum and Trotter [1981] observed the following relation between the integer rounding and the integer decomposition property:

(5.80) Let A be a nonnegative integer matrix. Then the system $x \geq \mathbf{0}, Ax \geq \mathbf{1}$ has the integer rounding property if and only if the blocking polyhedron $B(P)$ of $P := \{x \mid x \geq \mathbf{0}, Ax \geq \mathbf{1}\}$ has the integer decomposition property and all minimal integer vectors in $B(P)$ are transposes of rows of A (minimal with respect to $\leq$).

Similarly,

(5.81) Let A be a nonnegative integer matrix. Then the system $x \geq \mathbf{0}, Ax \leq \mathbf{1}$ has the integer rounding property if and only if the

antiblocking polyhedron $A(P)$ of $P := \{x \mid x \geq \mathbf{0}, Ax \leq \mathbf{1}\}$ has the integer decomposition property and all maximal integer vectors in $A(P)$ are transposes of rows of A (maximal with respect to $\leq$).

(Theorem 22.19 in Schrijver [1986b].)

5.20. Box-total dual integrality

A system $Ax \leq b$ is called *box-totally dual integral*, or just *box-TDI*, if the system $d \leq x \leq c, Ax \leq b$ is totally dual integral for each choice of vectors $d, c \in \mathbb{R}^n$. By Theorem 5.22,

(5.82) if $Ax \leq b$ is box-totally dual integral, then the polyhedron $\{x \mid Ax \leq b\}$ is box-integer.

We will need the following two results.

Theorem 5.34. *If $Ax \leq b$ is box-TDI in n dimensions and $w \in \mathbb{R}^n$, then $Ax \leq b - Aw$ is box-TDI.*

Proof. Directly from the definition of box-total dual integrality. ∎

Theorem 5.35. *Let $Ax \leq b$ be a system of linear inequalities, with A an $m \times n$ matrix. Suppose that for each $c \in \mathbb{R}^n$, $\max\{c^\mathsf{T} x \mid Ax \leq b\}$ has (if finite) an optimum dual solution $y \in \mathbb{R}^m_+$ such that the rows of A corresponding to positive components of y form a totally unimodular submatrix of A. Then $Ax \leq b$ is box-TDI.*

Proof. Choose $d, c \in \mathbb{R}^n$, with $d \leq c$, and choose $c \in \mathbb{Z}^n$. Consider the dual of maximizing $c^\mathsf{T} x$ over $Ax \leq b$, $d \leq x \leq c$:

$$\min\{y^\mathsf{T} b + z_1^\mathsf{T} c - z_2^\mathsf{T} d \mid y \in \mathbb{R}^m_+, z_1, z_2 \in \mathbb{R}^n_+, y^\mathsf{T} A + z_1^\mathsf{T} - z_2^\mathsf{T} = c^\mathsf{T}\}. \tag{5.83}$$

Let y, z_1, z_2 attain this optimum. Define $c' := c - z_1 + z_2$. By assumption, $\min\{y'^\mathsf{T} b \mid y' \in \mathbb{R}^m_+, y'^\mathsf{T} A = c'^\mathsf{T}\}$ has an optimum solution such that the rows of A corresponding to positive components of y' form a totally unimodular matrix. Now y', z_1, z_2 is an optimum solution of (5.83). Also, the rows in $Ax \leq b$, $d \leq x \leq c$ corresponding to positive components of y', z_1, z_2 form a totally unimodular matrix. Hence by Corollary 5.20b, (5.83) has an integer optimum solution. ∎

5.21. The integer hull and cutting planes

Let P be a rational polyhedron. The *integer hull* P_I of P is the convex hull of the integer vectors in P:

(5.84) $P_I = \text{conv.hull}(P \cap \mathbb{Z}^n)$.

It can be shown that P_I is a rational polyhedron again.

Consider any rational affine halfspace $H = \{x \mid c^T x \leq \delta\}$, where c is a nonzero integer vector such that the g.c.d. of its components is equal to 1 and where $\delta \in \mathbb{Q}$. Then it is easy to show that

(5.85) $H_I = \{x \mid c^T x \leq \lfloor \delta \rfloor\}$.

The inequality $c^T x \leq \lfloor \delta \rfloor$ (or, more correctly, the hyperplane $\{x \mid c^T x = \lfloor \delta \rfloor\}$) is called a *cutting plane*.

Define for any rational polyhedron P:

(5.86) $P' := \bigcap_{H \supseteq P} H_I$,

where H ranges over all rational affine halfspaces H containing P. Then P' is a rational polyhedron contained in P. Since $P \subseteq H$ implies $P_I \subseteq H_I$, we know

(5.87) $P_I \subseteq P' \subseteq P$.

For $k \in \mathbb{Z}_+$, define $P^{(k)}$ inductively by:

(5.88) $P^{(0)} := P$ and $P^{(k+1)} := (P^{(k)})'$.

Then (Gomory [1958,1960], Chvátal [1973a], Schrijver [1980b]):

Theorem 5.36. *For each rational polyhedron there exists a $k \in \mathbb{Z}_+$ with $P_I = P^{(k)}$.*

(For a proof, see Theorem 23.2 in Schrijver [1986b].)

5.21a. Background literature

Most background on polyhedra and linear and integer programming needed for this book can be found in Schrijver [1986b].

More background can be found in Dantzig [1963] (linear programming), Grünbaum [1967] (polytopes), Hu [1969] (integer programming), Garfinkel and Nemhauser [1972a] (integer programming), Brøndsted [1983] (polytopes), Chvátal [1983] (linear programming), Lovász [1986] (ellipsoid method), Grötschel, Lovász, and Schrijver [1988] (ellipsoid method), Nemhauser and Wolsey [1988] (integer programming), Padberg [1995] (linear programming), Ziegler [1995] (polytopes), and Wolsey [1998] (integer programming).