Contents

Introduction

The goal of *Mathematical Programming* is the design of mathematical solution methods for optimization problems. These methods should be *algorithmic* in the sense that they can be converted into computer programs without much effort. The main concern, however, is not the eventual concrete implementation of an algorithm but the necessary prerequisit thereof: the exhibition of a solution strategy that hopefully makes the ensuing algorithm ”efficient” in practice. Mathematical programming thus offers an approach to the theory of mathematical optimization that is very much motivated by the question whether certain parameters (solutions of an optimization problem or eigenvalues of a matrix) not only exist in an abstract way but can actually be computed well enough to satisfy practical needs.

Mathematical optimization traditionally decomposes into three seemingly rather disjoint areas: *Discrete* (or *combinatorial*) *optimization*, *linear optimization* and *nonlinear optimization*. Yet, a closer look reveals a different picture. Efficiently solvable discrete optimization problems are typically those that can be cast into the framework of linear optimization. And, as a rule of thumb, nonlinear problems are solved by repeated linear (or quadratic) approximation.

The dominant role of linearity in optimization is not surprising. It has long been known that much of the structural analysis of mathematical optimization can be achieved taking advantage of the language of vector spaces (see, for example, Luenberger’s elegant classic treatment [55]). Moreover, it appears to be an empirical fact that not only computations in linear algebra can be carried out numerically efficiently in practice but that, indeed, efficient numerical computation is tantamount to being able to reduce the computational task as much as possible to linear algebra.

The present book wants to introduce the reader to the fundamental algorithmic techniques in mathematical programming with a strong emphasis on the central position of linear algebra both in the structural analysis and the computational procedures. Although an optimization problem often admits a geometric picture involving sets of points in Euclidean space, which may guide the intuition in the structural analysis, we stress the role of the *presentation* of a problem in terms of explicit functions that encode the set of admissible solutions and the quantity to be optimized. The presentation is crucial for the design of a solution method and its efficiency.

The book attempts to be as much self-contained as possible. Only basic knowledge about (real) vector spaces and differentiable functions is assumed at the outset. Chapter 1 reviews this material, providing proofs of facts that might be not (yet) so familiar to the reader. We really begin in Chapter 2, which introduces the fundamental techniques of numerical linear algebra we will rely on later. Chapter 3 provides the corresponding geometric point of view. Then linear programs are treated.

Having linear programming techniques at our disposal, we investigate discrete optimization problems and discuss theories for analyzing their ”complexity” with respect to their solvability by ”efficient” algorithms. Nonlinear programs proper are presented in the last three chapters. Convex minimization problems occupy here a position between linear and nonlinear structures: while the feasible sets of linear programs are *finite* intersections of half-spaces, convex problems may be formulated with respect to *infinite* intersections of half-spaces. Convex optimization problems mark the border of efficient solvability. For example, quadratic optimization problems turn out to be “efficiently” solvable if and only if they are convex.

The book contains many items marked ”**Ex**”. These items are intended to provide both ”examples” and ”exercises” to which also details of proofs or additional observations are deferred. They are meant to be an integral part of the presentation of the material. We cordially invite the interested reader to test his or her understanding of the text by working them out in detail .

CHAPTER 1

Network Flows

1.1. Graphs

A *graph* $G = (V, E)$ is a combinatorial object consisting of a finite set V of *vertices* (or *nodes*) and a finite set E of *edges* together with an incidence relation that associates with every edge $e \in E$ two *endpoints* $v, w \in V$. We say that e is *incident* with v and w (and vice versa).

It is common to write $e = (v, w)$ if $e \in E$ has endpoints v and w, although this notation is somewhat misleading: First, it suggests that an edge is an *ordered* pair of nodes, which it is not. (Later we will introduce "directed" graphs, however, for which this is well the case!) Second, writing $e = (v, w)$ does not necessarily specify the edge $e \in E$ uniquely. It is possible that E contains several edges with the same endpoints v and w. Such edges are said to be *parallel*. An edge of the form $e = (v, v)$ is called a *loop*. We will usually consider graphs without loops, but we do allow parallel edges.

If $e = (v, w) \in E$, we call v and w *adjacent* (or *neighbors*). We also say that e *joins* v and w. The set of edges incident with $v \in V$ will be denoted by $\delta(v) \subseteq E$. The *degree* of a node $v \in V$, denoted by $\deg(v)$, is the number of edges incident with v.

A *path* $P = v_0, e_1, v_1, e_2, \ldots, v_k$ is an alternating sequence of vertices and edges such that edge e_j joins the vertices v_{j-1} and v_j. More precisely, we say that P is a path *from* v_0 *to* v_k or a $v_0 - v_k$ *path*. P is *simple* if all vertices $v_0, \ldots, v_k$ are distinct. We will often identify simple paths with their corresponding edge sets. So for our purposes, a simple path is a subset $P \subseteq E$ that can be arranged to yield a $v_0 - v_k$ path (or, equivalently, a $v_k - v_0$ path). The number of edges in P is the *length* of P.

In the same spirit, we define a *circuit* to be a subset $C \subseteq E$ consisting of a simple path P plus an edge $e = (v_0, v_k)$ joining the two endpoints v_0 and v_k of P.

A graph $G = (V, E)$ is *connected* if every pair of nodes is joined by a path. A connected graph containing no circuit is a *tree*.

Ex. 1.1. *Prove that a tree with $n \geq 1$ nodes has exactly $n - 1$ edges. Conclude that each such tree has at least two nodes of degree 1. (Such nodes are called* leaves *of the tree.)*

Ex. 1.2. *Show that every two nodes of a tree are joined by exactly one path.*

EX. 1.3. *Prove that a circuit free graph with n nodes and $n-1$ edges is a tree. (What if circuits may exist?)*

A *disconnected* (*i.e.*, not connected) graph $G = (V, E)$ decomposes in an obvious way into a number of "connected components". In order to define these, we need to introduce the notion of a "subgraph".

A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$ (with incidences as in G). If $V' \subseteq V$ we denote by $E(V') \subseteq E$ the set of all edges with both endpoints in V'. The graph $G' = (V', E(V'))$ is the subgraph *induced* by $V' \subseteq V$, which we denote by $G[V']$.

A *connected component* (or *component* for short) of G is a (with respect to inclusion) maximal connected subgraph. An alternative definition can be given as follows.

For any $U \subseteq V$, let $\delta(U) \subseteq E$ be the set of edges joining nodes in U to nodes in the complement $\overline{U} = V \setminus U$. The set $\delta(U) \subseteq E$ is called the *cut* induced by $U \subseteq V$. A component of G is then an induced subgraph $G[U]$, where $\emptyset \neq U \subseteq V$ is a (with respect to inclusion) minimal subset of V with $\delta(U) = \emptyset$. The equivalence of these two definitions is immediate from the following "theorem of the alternative":

THEOREM 1.1. *For each graph $G = (V, E)$ exactly one of the following is true:*

(i) *G is connected*
(ii) *There exists some subset $\emptyset \neq U \neq V$ of nodes with $\delta(U) = \emptyset$.*

Proof. If (ii) holds, then there can be no path joining a node $u \in U$ to a node $\overline{u} \in \overline{U} = V \setminus U$. Hence G is disconnected. Conversely, assume G is disconnected and let $u, \overline{u} \in V$ be two nodes that are not joined by any path. Construct a connected component $G[U]$ with $u \in U$ as follows:

Start with $U = \{u\}$ and extend U by adding endpoints of edges in $\delta(U)$ as long as $\delta(U) \neq \emptyset$. By construction, each node that enters U is connected to u by some path. Therefore, the construction will stop with $\delta(U) = \emptyset$ for some $U \neq V$ (since $\overline{u}$ never enters U).

$\diamond$

A *subtree* of $G = (V, E)$ is a subgraph that is a tree. Again we identify a subtree with its edge set $T \subseteq E$. A subtree $T \subseteq E$ is a *spanning tree* of G if each node of G is incident with T. Clearly, G admits a spanning tree if and only if G is connected. In this case, we may construct a spanning tree as follows (*cf.* the construction used in the proof of Theorem 1.1):

Spanning Tree

INIT: $U \leftarrow \{u\}$ for some $u \in V$; $T \leftarrow \emptyset$.
ITER: WHILE $\delta(U) \neq \emptyset$ DO
BEGIN
  Choose $e \in \delta(U)$.
  Extend T by e and U by the endpoint of e that is not in U.
END

The following lemma exhibits a fundamental property of spanning trees.

LEMMA 1.1 ("Exchange Property"). *Let $T \subseteq E$ be a spanning tree and $e \in E \setminus T$ arbitrary. Then $T \cup e$ contains a unique circuit C. The removal of any edge $f \in C$ yields a spanning tree $T' = (T \cup e) \setminus f$.*

Proof. Let $P \subseteq T$ be the unique path joining the two endpoints of e (*cf.* Ex. 1.2). Then $C = P \cup e$ is a circuit. Uniqueness of C follows from the uniqueness of P. Removal of any $f \in C$ destroys the unique circuit C, so $T' = (T \cup e) \setminus f$ is circuit free. Since $|T'| = |T|$, T' is again a tree (*cf.* Ex. 1.3). The claim follows. ◇

After these preliminary observations, we study our first basic combinatorial optimization problem.

The Minimum Cost Spanning Tree Problem. Given a connected graph $G = (V, E)$ and a cost vector $\mathbf{c} \in \mathbb{R}^E$ that associates a *cost* $c_e \in \mathbb{R}$ with every edge $e \in E$, the problem is to find a spanning tree $T \subseteq E$ of minimal cost

$$\mathbf{c}(T) = \sum_{e \in T} c_e \ .$$

EX. 1.4. Consider a set V of n points ("locations") in the Euclidean plane $\mathbb{R}^2$ that are to be connected by some "communication network". Assume that for a certain set E of pairs (v, w) of points it is feasible to establish a direct communication link e between v and w at a cost of c_e say, proportional to the distance of v and w. A minimum cost network connecting all points is then a minimum cost spanning tree.

The problem can be solved by a straightforward adaptation of the algorithm Spanning Tree above:

Min Cost Spanning Tree

INIT: $U \leftarrow \{u\}$ for some $u \in V$; $T \leftarrow \emptyset$.
ITER: WHILE $\delta(U) \neq \emptyset$ DO
BEGIN
    Choose $e \in \delta(U)$ of minimal cost c_e.
    Extend T by e and U by the endpoint of e that is not in U.
END

THEOREM 1.2. *Algorithm Min Cost Spanning Tree computes a spanning tree T of minimal cost $c(T)$.*

Proof. We claim that each current subtree $T \subseteq E$ can be extended to some min cost spanning tree $T^* \supseteq T$. This is clearly true for $T = \emptyset$. Assume inductively that $T^* \supseteq T$ is a min cost spanning tree and T is extended by $e \in \delta(U)$ in the next step.

If $e \in T^*$, there is nothing left to show. If $e \notin T^*$, let C be the unique circuit in $T^* \cup e$. The path $P = C \setminus e$ connects a node in U to a node outside U. So P must contain an edge $f \in \delta(U)$. Clearly, $f \in C$ and $f \neq e$. Since $e \in \delta(U)$ is cost minimal, $(T^* \cup e) \setminus f$ is again a min cost spanning tree and extends $T \cup e$, which proves the claim.

◇

Running Time. An *algorithm* for solving a special type of problems (such as the min cost spanning tree problem) proceeds by performing a series of "elementary operations", *e.g.*, comparing two numbers, adding or labeling an edge *etc.* until the problem (more precisely the problem *instance*) at hand is solved. The *running time* estimates the number of operations necessary to solve a problem instance of a given *size*. (We discuss these notions in detail in Chapter 8.) For example, measuring the size of a graph $G = (V, E)$ in terms of $n = |V|$ and $m = |E|$, we obtain the following straightforward bound on the number of operations for the min cost spanning tree algorithm: There are $n - 1$ tree extension steps. Each step finds the min cost edge $e \in \delta(U)$ by comparing $|\delta(U)| \leq m$ numbers. So the total number of operations is bounded by a constant times $m \cdot n$. (The constant accounts for "implementational details", *e.g.*, the data structures we use for accessing the edges incident with a vertex.) We express this fact by saying that the algorithm has *running time* $O(n\,m) = O(|V|\,|E|)$. (*Cf.* p. **??** for a formal definition of the "big O" notation.)

REMARK. The bound $O(n\,m)$ can be improved by using clever data-structures to speed up the the "minimum search" in each extension step. For example, a running time of $O(m \log n) = O(m \log m)$ can be achieved this way (see, *e.g.*, [12]).

1.1.1. Directed Graphs. Loosely speaking, a "directed graph" is obtained from a graph $G = (V, E)$ by giving a "direction" to each edge $e \in E$. We denote the resulting directed graph (again) by $D = (V, E)$ and call G its *underlying* (undirected) graph. More precisely, a *directed graph* $D = (V, E)$ consists of a finite set V of vertices and a finite set E of *directed edges* (or *arcs*) together with an incidence relation that assigns to each directed edge $e \in E$ two distinguished *endpoints* in V: The *tail* of e and the *head* of e.

Directed edges with the same endpoints are called *parallel* or *antiparallel*, according to whether or not they have the same head (or tail). We denote directed edges again as ordered pairs $e = (v, w)$, where now the ordering indicates that v is the tail and w is the head of e. We say that e is *directed* from v to w or that e *leaves* v and *enters* w.

A set $S \subseteq E$ of directed edges in the directed graph $D = (V, E)$ is a *path (circuit, tree)*, if this is true for the corresponding set of undirected edges in the underlying graph G. Similarly, we say that D is *connected*, if G is connected.

When we traverse a simple $r - s$ path $P \subseteq E$ in a directed graph from r to s, we traverse each edge $e = (v, w) \in P$ in either the forward direction (*i.e.*, from v to w) or backward direction (*i.e.*, from w to v). Correspondingly, the $r - s$ path P is partitioned into a set P^+ of *forward edges* and a set P^- of *backward edges*. P is a *directed* $r - s$ path if $P = P^+$.

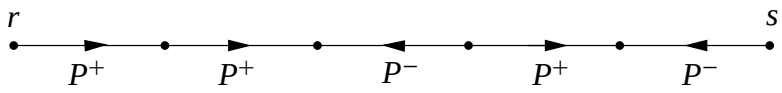

FIGURE 1.1. *Forward and backward edges of P*

Similarly, if $C \subseteq E$ is a circuit, we may fix one of the two possible *orientations* of C, thereby splitting C into a set of forward edges C^+ and a set of backward edges C^-. (With respect to the opposite orientation, the roles of C^+ and C^- are interchanged.) The circuit C is *directed* if $C = C^+$ (or $C = C^-$).

A similar partition can be defined for a tree $T \subseteq E$ by specifying a *root* node r of T. With respect to a fixed root r, a tree T splits into a set T^+ of *forward edges* (those pointing away from r) and a set T^- of *backward edges* (those pointing towards r). We call T a *directed tree* (*rooted at* r) if $T = T^+$.

The directed graph $D = (V, E)$ is called *strongly connected* , if any two nodes $v, w \in V$ are joined by directed paths (one from v to w and one from w to v).

EX. 1.5. *Show that a directed graph $D = (V, E)$ is strongly connected if and only if it is connected and each edge is contained in a directed circuit.*

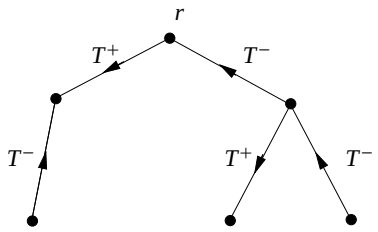

FIGURE 1.2. *Forward and backward edges of a rooted tree*

To derive a directed analogue of Theorem 1.1, we also partition cuts into forward and backward parts as follows. For $U \subseteq V$ and $\overline{U} = V \setminus U$, let

$$\begin{aligned} \delta^+(U) &= \left\{e \in E \mid e = (u, \overline{u}) \quad \text{for some} \quad u \in U, \overline{u} \in \overline{U}\right\}, \\ \delta^-(U) &= \left\{e \in E \mid e = (\overline{u}, u) \quad \text{for some} \quad u \in U, \overline{u} \in \overline{U}\right\}. \end{aligned}$$

THEOREM 1.3. *For the directed graph $D = (V, E)$ exactly one of the following is true:*

(i) *D is strongly connected.*
(ii) *There exists some subset $\emptyset \neq U \neq V$ of nodes with $\delta^+(U) = \emptyset$.*

Proof. Similar to the proof of Theorem 1.1.

◇

1.1.2. Incidence Matrices. Combinatorial problems on graphs, directed graphs and other "incidence structures" can often be formulated as (integer) linear programming problems. Basically, such formulations are obtained by representing sets *via* corresponding "incidence vectors" and incidence relations *via* corresponding "incidence matrices".

If S is a finite set, we denote by $\mathbb{R}^S$ the set of real vectors with coordinates indexed by the elements of S (as we had already done in the min cost spanning tree problem). Hence $\mathbb{R}^S \cong \mathbb{R}^{|S|}$. If T is another finite set then $\mathbb{R}^{S\times T}$ denotes the set of real matrices with rows and columns indexed by elements from S resp. T. Hence $\mathbb{R}^{S\times T} \cong \mathbb{R}^{|S|\times|T|}$.

REMARK. We do not need to assume any (implicit) ordering of the elements in S or T for the following reason: Even without such an ordering, operations like vector addition, matrix multiplication *etc.* are defined in the obvious way.
For example, if $\mathbf{A} = (a_{st}) \in \mathbb{R}^{S\times T}$ and $\mathbf{x} \in \mathbb{R}^T$, then $\mathbf{y} = \mathbf{Ax} \in \mathbb{R}^S$ has coordinates defined by

$$y_s = \sum_{t\in T} a_{st}x_t \qquad (s \in S) \ .$$

Let $D = (V, E)$ now be a directed graph without loops. The *incidence matrix* of D is the matrix $\mathbf{A} = (a_{ve}) \in \mathbb{R}^{V \times E}$ defined by

$$a_{ve} = \begin{cases} -1 & \text{if } v \text{ is the tail of } e \\ +1 & \text{if } v \text{ is the head of } e \\ 0 & \text{otherwise.} \end{cases}$$

For each edge $e = (u, v)$ of D, the sum of the entries in the column $\mathbf{A}_{\cdot e}$ of the incidence matrix $\mathbf{A}$ is $a_{ue} + a_{ve} = 0$. So the sum of all rows is the zero vector $\mathbf{0}^T \in \mathbb{R}^E$, which implies rank $\mathbf{A} \leq |V| - 1$.

We introduce incidence vectors of paths, circuits and trees analogously. If $P \subseteq E$ is a simple path and P^+ (P^-) is the set of forward (backward) edges, then the *incidence vector* of P is the vector $\mathbf{x} \in \mathbb{R}^E$ given by

$$x_e = \begin{cases} +1 & \text{if } e \in P^+ \\ -1 & \text{if } e \in P^- \\ 0 & \text{otherwise.} \end{cases}$$

The incidence vectors of circuits and trees (relative to a given root node) are defined similarly.

Ex. 1.6. *Let $\mathbf{A} \in \mathbb{R}^{V \times E}$ be the incidence matrix of $D = (V, E)$ and $\mathbf{x} \in \mathbb{R}^E$ the incidence vector of a circuit. Show: $\mathbf{Ax} = \mathbf{0}$. Similarly, if $\mathbf{x} \in \mathbb{R}^E$ is the incidence vector of an $r - s$ path, show: $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{b} \in \mathbb{R}^V$ has components $b_r = -1$, $b_s = +1$ and $b_v = 0$ otherwise.*

Ex. 1.6 indicates how incidence matrices can be used to translate combinatorial problems into linear programming problems. To study this relation in more detail, we need to identify those submatrices of $\mathbf{A}$ that are column bases, *i.e.*, generate the column space col $\mathbf{A}$.

Theorem 1.4. *Let $\mathbf{A} \in \mathbb{R}^{V \times E}$ be the incidence matrix of $D = (V, E)$ and let $F \subseteq E$. Then the set $\mathbf{A}_{\cdot F}$ of columns of $\mathbf{A}$ corresponding to F is linearly independent if and only if F contains no circuit. Hence, if D is connected, the column bases of $\mathbf{A}$ are exactly the submatrices $\mathbf{A}_{\cdot T}$ where $T \subseteq E$ is a spanning tree.*

Proof. If $F \subseteq E$ contains a circuit C with incidence vector $\mathbf{x} \in \mathbb{R}^E$, then $\mathbf{Ax} = \mathbf{0}$ (*cf.* Ex. 1.6). So the column set $\mathbf{A}_{\cdot C}$ (and hence $\mathbf{A}_{\cdot F}$) is not independent.

Conversely, if F does not contain any circuit, then there exists a node $v \in V$ that is incident with exactly one edge $f \in F$ (why?). Consequently, the submatrix $\mathbf{A}_{\cdot F}$ has a unique non-zero entry in row v, namely in the column $\mathbf{A}_{\cdot f}$. Assuming inductively that the columns corresponding to $F \setminus \{f\}$ are independent, we conclude that also the columns in $A_{\cdot F}$ are independent, which proves the first statement.

Since spanning trees are the maximal circuit free edge sets in connected graphs, also the second statement follows.

◇

EX. 1.7. *Let $D = (V, E)$ be a connected graph with incidence matrix $\mathbf{A} \in \mathbb{R}^{V \times E}$. Show:* rank $\mathbf{A} = |V| - 1$.

EX. 1.8. *Reprove Lemma 1.1 with Theorem 1.4 and linear algebra.*

COROLLARY 1.1. *Assume D is connected and r, s are two different nodes. Define $\mathbf{b} \in \mathbb{R}^V$ by $b_r = -1$, $b_s = +1$ and $b_v = 0$ otherwise (*cf. *Ex. 1.6). Then the vertices (basic solutions) of $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$ are exactly the incidence vectors of directed simple $r - s$ paths.*

Proof. Let $\mathbf{x} \in \mathbb{R}^E$ be a basic solution of $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$. By definition, this means that there exists some column basis $\mathbf{A}_{\cdot T}$ of $\mathbf{A}$ such that $x_e = 0$ for all $e \in E \setminus T$ (*cf.* p.**??**). Since these requirements uniquely determine $\mathbf{x} \in \mathbb{R}^E$, we conclude that $\mathbf{x}$ must be the incidence vector of the unique $r - s$ path $P \subseteq T$ (since this also satisfies these requirements (*cf.* Ex. 1.6)).

Conversely, assume that $\mathbf{x} \in \mathbb{R}^E$ is the incidence vector of a simple directed $r - s$ path $P \subseteq E$. Extend P to a spanning tree $T \supseteq P$. Since $\mathbf{Ax} = \mathbf{b}$ and $x_e = 0$ for $e \in E \setminus T$, $\mathbf{x}$ is a basic solution corresponding to the column basis $\mathbf{A}_{\cdot T}$.

◇

1.2. Shortest Paths

Given a graph $G = (V, E)$ and two nodes $r, s \in V$, the *shortest path problem* asks for an $r - s$ path $P \subseteq E$ of minimum length (= number of edges). This minimum length is denoted by $d(r, s)$ and is called the *distance* between r and s. Note that such a path of minimum length will necessarily be simple. In the following we will assume without loss of generality that G is connected. (Otherwise we restrict ourselves to the component containing r.)

The shortest path problem can be solved in a straightforward way by *Dijkstra's algorithm* [17]. Start with $U_0 = \{r\}$ and then determine the set $U_k \subseteq V$ of nodes at *distance* k from r recursively:

$$U_{k+1} = \{v \in V \setminus \bigcup_{i=0}^{k} U_i \mid (u, v) \in E \text{ for some } u \in U_k\}.$$

Once the distances of all nodes $v \in V$ from r are computed, the corresponding shortest paths can be easily deduced from that information.

Alternatively, one may keep track directly of the necessary information while computing the distances: Whenever a node $v \in V$ enters U_{k+1}, label one of the edges (u, v) with $u \in U_k$. The set of labeled edges will then finally form a spanning tree $T \subseteq E$ with the property that each $r - v$ path $P \subseteq T$ is a shortest $r - v$ path. Such a tree is called a *shortest path tree* (relative to r).

REMARK. If we are only interested in a shortest $r - s$ path, we can of course stop as soon as s enters some U_k. In the worst case, however, the node s may happen to be the

last one we reach. So the running time of our algorithm can only be bounded by the time needed to construct the full shortest path tree.
Note that the computation of U_{k+1} scans each edge incident with a node in U_k at most once. Counting such an *edge scan* as an elementary step, we obtain a running time of $O(|E|)$ for Dijkstra's algorithm.

The same method works for directed graphs and shortest directed $r-s$ paths. Again, we assume that each node $v \in V$ can be reached from r along a directed path. So each node $v \in V$ will have a well-defined (finite) *distance* $y_v \in \mathbb{Z}_+$ from r, which we compute as above by setting $y_v = k$ if $v \in U_k$. Once we have computed the complete *distance vector* $\mathbf{y} = (y_v) \in \mathbb{Z}_+^V$, we again recover shortest directed $r-v$ paths for all $v \in V$ and compute a *directed shortest path tree* $T \subseteq E$, *i.e.*, a directed spanning tree rooted at r such that each $r-v$ path $P \subseteq T$ is shortest.

Computing shortest paths and distances are, in some way, "equivalent" tasks. Let us investigate their mutual relationship in more detail. Consider an arbitrary vector $\mathbf{y} \in \mathbb{R}^V$. To be a distance vector, $\mathbf{y}$ must at least satisfy

$$y_w \leq y_v + 1 \qquad \text{for all } e = (v, w) \in E. \tag{1.1}$$

The relation between shortest paths and distances is as follows.

LEMMA 1.2.

$$\min \{|P| \mid P \textit{ is a directed } r-s \textit{ path}\} = \max \{y_s - y_r \mid \mathbf{y} \in \mathbb{R}^V \textit{ satisfies (1.1)}\}$$

Proof. Assume $P \subseteq E$ is a directed $r-s$ path (which we assume to exist) and $\mathbf{y} \in \mathbb{R}^V$ satisfies (1.1). Then "min $\geq$ max" follows from the observation

$$|P| = \sum_{e \in P} 1 \;\geq\; \sum_{(v,w) \in P} y_w - y_v = v_s - y_r \;.$$

Equality is achieved when we take P as a shortest $r-s$ path and $\mathbf{y} \in \mathbb{R}^V$ as the distance vector.

◇

The above min-max relation will be studied again later in a more general setting. In particular, we shall see that it is just a very special instance of linear programming duality. (This is why we denote distance vectors by $\mathbf{y} \in \mathbb{R}^V$.) For the time being, note that the maximization problem in Lemma 1.2 is a linear program which is always feasible (take $\mathbf{y} = \mathbf{0}$) and bounded (since we assume $r-s$ paths exist). So optimal solutions $\mathbf{y} \in \mathbb{R}^V$ exist. Given any optimal solution $\mathbf{y} \in \mathbb{R}^V$, we can immediately read off shortest $r-s$ paths: We let

$$E^* = \{e = (v, w) \in E \mid y_w = y_v + 1\}.$$

Then a directed $r-s$ path $P \subseteq E$ is a shortest $r-s$ path if and only if $P \subseteq E^*$ (*cf.* the proof of Lemma 1.2).

Let us illustrate the use of this fact by proving the following (intuitively obvious?) result.

LEMMA 1.3. *Let $D = (V, E)$ be a directed graph, $r, s \in V$ and $P \subseteq E$ be a shortest directed $r - s$ path. Extending E by edges that are antiparallel to edges in P does not create any new $r - s$ path of length $\leq |P|$.*

Proof. Let $\mathbf{y}$ be an optimal solution of the maximization problem with respect to $D = (V, E)$ and consider $E^* \subseteq E$ as defined above. Then, if $e = (v, w) \in E^*$, we have $y_v = y_w - 1 < y_w + 1$.

So $\mathbf{y}$ satisfies (1.1) even when we extend E by edges that are antiparallel to edges in E^* (and hence to the shortest $r - s$ path $P \subseteq E^*$). Since $y_s - y_r = |P|$, $\mathbf{y}$ is also an optimal solution of the max problem with respect to the extended graph. Since none of the new edges enters E^* in the extended graph, the shortest $r - s$ paths in the extended graph are exactly the original shortest paths.

◇

EX. 1.9. *Prove Lemma 1.3 "directly" (i.e., without "distances" and Lemma 1.2).*

The general shortest path problem allows a weight vector $\mathbf{c} \in \mathbb{R}^E$ that associates a *cost* c_e with every edge $e \in E$. We now search for an $r - s$ path P of minimal *cost* (*i.e.,* weighted length)

$$\mathbf{c}(P) = \sum_{e \in P} c_e.$$

The unweighted shortest path problem corresponds to $\mathbf{c} = \mathbf{1}$.

The principle of Dijkstra's algorithm for the unweighted shortest path applies equally to the case of nonnegative costs $c_e \geq 0$. With the understanding $c_{vw} = +\infty$ if $(v, w) \notin E$, one easily sees that y_v is the distance from r to v in the subgraph $G[U \cup v]$ in each stage of the following algorithm:

Min Cost Paths under Nonnegative Costs (Dijkstra)

INIT: $U \leftarrow \{r\}$; $y_r \leftarrow 0$; $y_v \leftarrow c_{rv}$ for all $v \in V \setminus U$.
ITER: WHILE $V \setminus U \neq \emptyset$ DO
BEGIN
  Choose $v \in V \setminus U$ with y_v minimal and extend U by v.
  Update $y_w \leftarrow \min\{y_w, y_v + c_{vw}\}$ for all $w \in V \setminus U$.
END

The most general setting allows cost coefficients to be also negative, *i.e.,* to model "gains" along edges. Algorithms for this problem will again compute *minimum cost (directed) paths trees* $T \subseteq E$, consisting of min cost paths from r to all nodes.

Before turning to the general minimum cost path problem in detail, let us see whether it is well-defined. Observe first that the case of undirected graphs is just a special case of the directed version. (Just replace each undirected edge by two

antiparallel arcs of the same cost). Thus we consider right away a directed graph $D = (V, E)$ and $\mathbf{c} \in \mathbb{R}^E$. As before, we assume that directed $r - v$ paths exist for all $v \in V$.

It may now happen that some directed circuit $C \subseteq E$ has negative cost

$$\mathbf{c}(C) = \sum_{e \in C} c_e < 0.$$

In the presence of such a *negative circuit* C, obviously, the cost of a path that traverses C arbitrarily often will tend to "$-\infty$". Therefore, we explicitly assume for the following discussion that negative circuits do not occur. This condition will automatically be checked by the algorithms for computing min cost paths, as we shall see below. Note that a min cost path P can be assumed to be simple if no negative circuits exist (as the removal of any 'non-negative' circuit from P cannot increase the cost.)

REMARK. One may wonder why we do not generally, *i.e.*, when negative circuits may occur, look for min cost *simple* directed paths. The latter problem, however, turns out to be *NP*-hard. Indeed, with $\mathbf{c} = -\mathbf{1}$, we would actually look for a *longest* simple $r - s$ path, which is at least as difficult to find as a Hamiltonian circuit (*cf.* Chapter **??**)

Let us now see how to construct minimum cost $r - s$ paths algorithmically in the general case (with possibly some negative edge costs). As before, we compute for all nodes $v \in V$ their *distance* y_v, *i.e.*, the cost of a min cost $r - v$ path. The following algorithm, due to Bellman [5] and Ford [23], aims at computing all distances in $n = |V|$ rounds. In the k-th round ($k = 1, \ldots, n$) the vector $\mathbf{y}^{(k)} \in \mathbb{R}^V$ represents the "distances" with respect to paths of length at most k.

Min Cost Path (Bellman-Ford)

INIT: $y_r^{(0)} \leftarrow 0\,, \quad y_v^{(0)} \leftarrow +\infty\,, \quad v \in V \backslash \{r\}$

ITER: FOR $k = 1, \ldots, n = |V|$ DO

$y_w^{(k)} \leftarrow \min\left\{y_w^{(k-1)}, \min\left\{y_v^{(k-1)} + c_e \mid e = (v, w) \in E\right\}\right\}$

THEOREM 1.5. *Let $\mathbf{y}^{(k)}$ ($k = 0, \ldots, n$) be the sequence of vectors computed by the Bellman-Ford algorithm. Then $\mathbf{y}^{(n)} = \mathbf{y}^{(n-1)}$ if and only if negative circuits do not exist. In this case, $\mathbf{y} = \mathbf{y}^{(n-1)}$ is the distance vector.*

Proof. It is straightforward to see (by induction on k) that $\mathbf{y}^{(k)} \in \mathbb{R}^V$ indeed specifies the distances with respect to min cost $r - v$ paths of length at most k. In case negative circuits do not exist, simple min cost paths (of length $k \leq n - 1$) exist. This observation proves the "if" case and the second statement.

To prove the converse implication, assume $\mathbf{y}^{(n)} = \mathbf{y}^{(n-1)}$. Suppose for a moment that we would continue to compute $\mathbf{y}^{(k)}$ as in the Bellman-Ford algorithm for $k = n+1, n+2, \ldots$. It is immediate from the recursive definition that $\mathbf{y}^{(n)} = \mathbf{y}^{(n-1)}$ implies $\mathbf{y}^{(k)} = \mathbf{y}^{(k-1)}$ for all $k \geq n$. If there were negative circuits, then at least one component of $\mathbf{y}^{(k)}$ would tend to "$-\infty$" as $k \to \infty$. So no negative circuit exists. $\diamond$

Ex. 1.10. *How can one identify negative circuits with the Bellman-Ford algorithm?*

Ex. 1.11. *Show that the Bellman-Ford algorithm runs in time $O(|V|\,|E|)$.*

Ex. 1.12. *Assuming that negative circuits do not exist, how can one determine a min cost $r-s$ path, given the distance vector $\mathbf{y} = \mathbf{y}^{(n)}$ as computed by the Bellman-Ford algorithm? Show that such a path can be found in time $O(|E|)$.*

The essential ingredient of the Bellman-Ford algorithm is the modification of the current "tentative" distance vector $\mathbf{y} \in \mathbb{R}^V$ (partly defined by $\mathbf{y}^{(k-1)}$ and partly by $\mathbf{y}^{(k)}$) by decreasing a component y_w with positive slack $y_w > y_v + c_e$ relative to some edge $e = (v, w)$. In case negative circuits do not exist, we eventually end in a situation where

$$y_w \leq y_v + c_e \qquad \text{for all } e = (v, w) \in E \tag{1.2}$$

holds and no further modifications are carried out. This idea also motivates a version of the (linear programming) simplex algorithm for solving min cost path problems, which we investigate next.

1.2.1. The Simplex Algorithm for Min Cost Paths. We will formulate the min cost path problem as a linear program and adjust the simplex algorithm to solve it. The result will be an algorithm for min cost paths that is somewhat different from the Bellman-Ford method.

Let $\mathbf{A} \in \mathbb{R}^{V \times E}$ be the incidence matrix of $D = (V, E)$ and let $\mathbf{b} \in \mathbb{R}^V$ be defined by $b_r = -1, b_s = +1, b_v = 0$ otherwise (*cf.* Corollary 1.1). Consider the pair of mutually dual linear programs

$$\begin{array}{llcl} (P) & \min & \mathbf{c}^T\mathbf{x} & \\ & s.t. & \mathbf{A}\mathbf{x} & = \mathbf{b} \\ & & \mathbf{x} & \geq \mathbf{0} \end{array} \qquad \text{and} \qquad \begin{array}{llcl} (D) & \max & \mathbf{y}^T\mathbf{b} & \\ & s.t. & \mathbf{y}^T\mathbf{A} & \leq \mathbf{c}^T. \end{array}$$

By Corollary 1.1, the feasible basic solutions of (P) are the incidence vectors of directed $r-s$ paths. So an optimal basic solution of (P) gives a min cost $r-s$ path. As we shall see, the dual program (D) is suited for computing the distance vector.

A vector $\mathbf{y} \in \mathbb{R}^V$ is often referred to as a *node potential* or simply a *potential* in this context. (This terminology originates from electric network theory, where

distances correspond to potential differences and costs are interpreted as resistances.) If $\mathbf{y} \in \mathbb{R}^V$ is a potential, the linear combinations $\mathbf{y}^T\mathbf{A}$ and $\mathbf{y}^T\mathbf{b} = y_s - y_r$ remain unchanged when we add (or subtract) a constant vector to $\mathbf{y}$. Thus, with respect to (D), we may restrict ourselves to potentials $\mathbf{y} \in \mathbb{R}^V$ that are *normalized* in the sense that $y_r = 0$.

Consider a column basis of $\mathbf{A}$, which (by Theorem 1.4) is a submatrix $\mathbf{A}_{\cdot T}$ arising from a spanning tree $T \subseteq E$. Assume that T is actually a directed tree rooted at r (which we assume to exist). Then the corresponding primal basic solution $\mathbf{x} \in \mathbb{R}^E$ of $\mathbf{Ax} = \mathbf{b}$ is the incidence vector of the unique directed $r - s$ path $P \subseteq T$ and hence a feasible primal solution, *i.e.*, $\mathbf{x} \geq \mathbf{0}$ with cost $\mathbf{c}^T\mathbf{x}$. A corresponding dual basic solution is a potential $\mathbf{y} \in \mathbb{R}^V$ such that $\mathbf{y}^T\mathbf{A}_{\cdot e} = c_e$ for $e \in T$, *i.e.*,

$$y_w = y_v + c_e \qquad \text{for all } e = (v, w) \in T. \tag{1.3}$$

The equations (1.3) determine a unique normalized potential $\mathbf{y} \in \mathbb{R}^V$, which can be easily computed "along the directed paths" in T, starting in r with $y_r = 0$. For $v \in V$, y_v is the cost of the unique $r - v$ path in T. In particular, $y_s = y_s - y_r = \mathbf{y}^T\mathbf{b}$ equals the cost of the unique $r - s$ path $P \subseteq T$. So $\mathbf{c}^T\mathbf{x} = \mathbf{y}^T\mathbf{b}$ and hence by linear programming duality we conclude that both $\mathbf{x}$ and $\mathbf{y}$ are optimal solutions to our problems (P) and (D) if and only if $\mathbf{y}$ is feasible for (D).

What if $\mathbf{y}$ is not feasible? Then some edge $e = (v, w) \in E \setminus T$ must be *infeasible* in the sense that

$$y_w > y_v + c_e.$$

In this case we modify the directed tree T to the directed tree (see Ex. 1.13)

$$T' = (T \cup e) \setminus f$$

where $f \in T$ is the unique edge in T with head w.

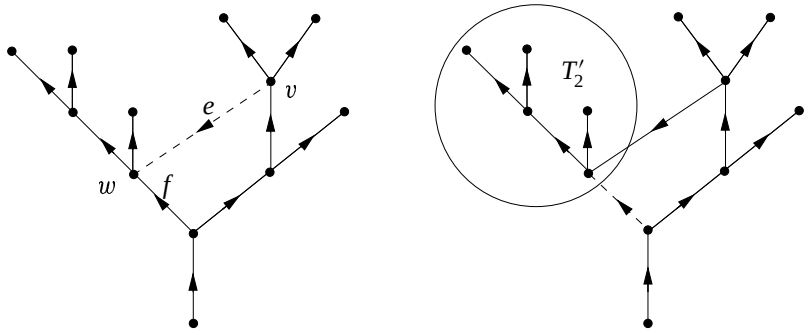

FIGURE 1.3. *Moving from T to $T' = (T \cup e) \setminus f$*

EX. 1.13. *Show that T' is again a directed tree unless the unique circuit $C \subseteq T \cup e$ is a negative directed circuit (in which case we stop since a negative circuit has been detected).*

The normalized potential $\mathbf{y}' \in \mathbb{R}^V$ corresponding to T' can easily be obtained by updating $\mathbf{y} \in \mathbb{R}^V$ as follows. The removal of e splits T' into two components T'_1 and T'_2. One of these, say T'_1, contains the root r and the other does not. By definition, the old potential $\mathbf{y} \in \mathbb{R}^V$ satisfies all edge constraints (1.3) for edges in $T'_1 \cup T'_2$ with equality. Hence the new potential $\mathbf{y}' \in \mathbb{R}^V$ is obtained when we decrease $\mathbf{y}$ on all nodes in T'_2 by an amount of

$$\delta = y_w - y_v - c_e \ > \ 0 \ .$$

(Note the difference to the Bellman-Ford algorithm: Here we decrease $\mathbf{y}$ on all nodes in T'_2 simultaneously.)

The move $T \to T'$ corresponds exactly to one iteration in the simplex algorithm as we replace one column $\mathbf{A}_{\cdot f}$ in the basis $\mathbf{A}_{\cdot T}$ by the column $\mathbf{A}_{\cdot e}$ from outside. Such a move may leave the primal solution $\mathbf{x} \in \mathbb{R}^E$ unchanged (in case the unique $r-s$ path $P \subseteq T$ does not contain the leaving edge f). In particular, the objective value $\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}$ may remain unchanged. Yet we do make a measurable progress in each step: The sum of all y_v ($v \in V$), *i.e.*, the sum of all costs of directed paths in T decreases strictly. Hence, in particular, cycling can not occur and the algorithm terminates.

Our description of the above simplex algorithm does not specify the infeasible edge to choose in each step. Specific rules for choosing the entering edge e lead to different variants of the simplex algorithm. One of them is as follows:

We let the simplex algorithm proceed in *rounds*. In each round we scan all vertices $w \in V$ once (in some order). When scanning $w \in V$, we check whether there are any infeasible edges entering w. In case there are such edges, we choose the "most infeasible" edge $e = (v, w)$ (*i.e.*, the edge which maximizes δ) to enter T.

Ex. 1.14. *Show that this variant of the simplex algorithm for shortest paths terminates after at most n rounds. As a consequence, conclude that its running time is* $O(n^3)$.
*(*Hint: *Show that, after k rounds, the potential* $\mathbf{y} \in \mathbb{R}^V$ *satisfies* $\mathbf{y} \leq \mathbf{y}^{(k)}$*, where* $\mathbf{y}^{(k)}$ *is the potential computed by Bellman-Ford.)*

1.2.2. Applications. Min cost path problems mainly occur as subproblems in more complex optimization problems related to traffic routing or communication network management. Sometimes, however, one also encounters them as stand-alone problems – and quite often in "disguise". The following two examples are also cited in [2].

Approximating Piecewise Linear Functions. Let $x_0 < x_1 < \ldots < x_n$ be $n+1$ real numbers and $f : [x_0, x_n] \to \mathbb{R}$ a piecewise linear function that is linear on each of the intervals $[x_{i-1}, x_i]$, $i = 1, \ldots, n$. If n is large, storing all information about f is very expensive. Therefore one might wish to approximate f by another piecewise linear function $\hat{f}$ that equals f in $x_0 = x_{j_0}, x_{j_1}, \ldots, x_{j_k} = x_n$.

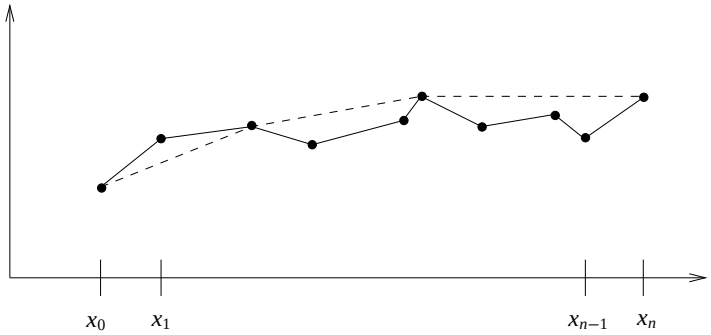

FIGURE 1.4. *Approximating a piecewise linear function*

Assume for example that we measure the approximation error $e(\hat{f})$ by

$$e(\hat{f}) = \sum_{i=0}^{n} \Big[f(x_i) - \hat{f}(x_i) \Big]^2 .$$

Assume, furthermore, that storing an approximating function $\hat{f}$ costs $\alpha \cdot k$, where k is the number of linear pieces of $\hat{f}$ and $\alpha > 0$ is the fixed cost of storing one linear piece (*i.e.*, storing the endpoints and the slope of $\hat{f}$ in this interval).

The total "cost" of the approximation function $\hat{f}$ that agrees with f on $x_0 = x_{j_0}, \ldots, x_{j_k} = x_n$ is then

$$c(\hat{f}) = \sum_{i=1}^{n-1} \Big[f(x_i) - \hat{f}(x_i) \Big]^2 + \alpha k.$$

Minimizing $c(\hat{f})$ can be formulated as a shortest path problem as follows: Let $V = \{0, 1, \ldots, n\}$ and let G be the complete graph on V. Define costs on the directed edges (i, j) with $i < j$ by

$$c_{ij} = \sum_{s=i}^{j} \Big[f(x_s) - \hat{f}_{ij}(x_s) \Big]^2 + \alpha$$

where $\hat{f}_{ij}$ is the linear function which coincides with f in x_i and x_j. Then there is a one-to-one correspondence between shortest $0 - n$ paths and optimal approximations $\hat{f}$.

Allocating Inspections on a Production Line. A production line consists of an entry node M_0 and a sequence of n production (manufacturing) nodes $M_1, \ldots, M_n$, each of which may be followed by a potential inspection.

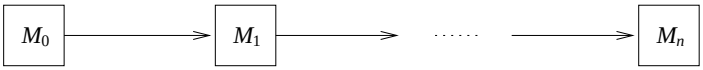

FIGURE 1.5

We assume that the product leaves node M_0 (and enters node M_1) in batches of size $B(0) \geq 1$. As a batch moves through the production line, the various operations performed on its items may render some of the items defective. The probability that a particular item is ruined in production phase M_i is say $p_i \geq 0$. Thus, if a batch with $B(i-1)$ non-defective items enters node M_i, the expected number of non-defective items leaving M_i in this batch is $(1-p_i)B(i-1)$.

If we decide to install an inspection station after a node M_j, then we will inspect all items leaving M_j and remove those which are defective from the batch. The cost of such an inspection depends on the last inspection having taken place because we have to look for all possible defects that have been introduced since then. Suppose g_{ij} is the cost of inspecting a single item after node M_j given that the last check on it took place after node M_i. Furthermore, suppose that there is a fixed ("set up") cost s_{ij} for inspecting a batch after M_j, given that it was last inspected after M_i. This cost is independent of the current number of items in the batch. (Think of the cost for removing the batch from the line and getting the necessary inspection tools ready.) Finally, suppose that the production cost in M_i is c_i per item in the batch that enters M_i.

There are two conflicting objectives. On the one hand, we try to avoid production costs incurred by further processing items that are ruined. In the extreme case, this objective would require to install an inspection station after each M_i. On the other hand, we try to keep the inspection cost low. In the extreme case, the latter objective would result in carrying out no inspections at all. (Usually, however, there is at least one final inspection prescribed to take place after M_n.)

The problem of finding an optimal inspection policy (minimizing the expected total cost) can be formulated as a shortest path problem. First, recall that the expected number of non-defective items in a batch leaving node j equals

$$B(j) \;=\; B(j-1)(1-p_j) \;=\; \ldots \;=\; B(0)\prod_{k=1}^{j}(1-p_k).$$

Construct now the graph $D = (V, A)$ with $V = \{0, \ldots, n\}$ and arcs (i, j) for all $i < j$. Introduce edge costs according to

$$c_{ij} = s_{ij} + B(i)g_{ij} + B(i)\sum_{k=i+1}^{j} c_k.$$

Then c_{ij} equals the total expected cost incurred in phases $i+1$ through j if inspections are done after i and j (and none in between). Solving the shortest path problem in D (for $r = 0$ and $s = n$) is equivalent with minimizing the total expected production/inspection cost.

1.3. Maximum Flow

A *network* (in the simplest case) is given by a directed graph $D = (V, E)$ with an (upper) *capacity vector* $\mathbf{u} \in \mathbb{R}_+^E$ associating a *capacity* u_e with every edge $e \in E$. Furthermore, there are two distinguished nodes: The *root* node $r \in V$ (also called *source*) and the *sink* $s \in V$. In this section we again assume that D is connected.

REMARK. One may imagine the edges to represent (unidirectional) pipelines. As much flow as possible is to be sent from r to s, respecting the edge capacities as upper bounds on the maximum throughput (per time unit). This way we would obtain the standard maximum flow problem as discussed below. In Section 1.4, we generalize this model, allowing in addition each edge $e \in E$ to be assigned a cost $c_e \in \mathbb{R}$ for sending one unit of flow through e.

We give a formal definition of the max flow problem. "Flows" are certain vectors $\mathbf{x} \in \mathbb{R}^E$, specifying the amount x_e of flow on each edge $e \in E$. When we send a flow from r to s then in each node $v \in V\backslash\{r, s\}$, the total flow into v should equal the total flow out of v. Formally, a *flow* is therefore defined to be a vector $\mathbf{x} \in \mathbb{R}^E$ such that

$$\sum_{e\in\delta^+(v)} x_e = \sum_{e\in\delta^-(v)} x_e \qquad \text{for all } v \in V\backslash\{r, s\}.$$

A flow $\mathbf{x} \in \mathbb{R}^E$ is *feasible* if $\mathbf{0} \le \mathbf{x} \le \mathbf{u}$. With our general shorthand notation (*cf.* p. 6), we define the *outflow* and the *inflow* of $v \in V$ (relative to $\mathbf{x}$) as

$$\mathbf{x}(\delta^+(v)) = \sum_{e\in\delta^+(v)} x_e \qquad \text{and} \qquad \mathbf{x}(\delta^-(v)) = \sum_{e\in\delta^-(v)} x_e.$$

Their difference is the *net outflow* of $v \in V$, denoted by

$$f_{\mathbf{x}}(v) \;=\; \mathbf{x}(\delta^+(v)) - \mathbf{x}(\delta^-(v)) \;.$$

In terms of the incidence matrix $\mathbf{A} \in \mathbb{R}^{V\times E}$ of $D = (V, E)$, we have

$$f_{\mathbf{x}}(v) \;=\; -\mathbf{A}_{v\cdot}\,\mathbf{x} \qquad (v \in V)$$

where $\mathbf{A}_{v\cdot}$ denotes the row of $\mathbf{A}$ corresponding to $v \in V$.

Our max flow problem can now be formally stated as the linear program

$$\max_{\mathbf{0}\le\mathbf{x}\le\mathbf{u}} \; f_{\mathbf{x}}(r) \quad \text{s.t.} \quad \mathbf{A}_{v\cdot}\,\mathbf{x} \;=\; 0 \quad \text{for all } v \in V\backslash\{r, s\} \tag{1.4}$$

The net outflow $f_{\mathbf{x}}(r)$ of the root r is called the *value* of the flow $\mathbf{x}$. The constraints $\mathbf{A}_{v\cdot}\,\mathbf{x} = 0$ are called the *flow constraints*. As mentioned earlier, they ensure that no flow is added or lost at any node $v \in V \setminus \{r, s\}$. Intuitively, it is clear from these constraints that the net outflow of the root equals the net inflow (inflow minus outflow) of the sink. More generally, we note

LEMMA 1.4. *If $\mathbf{x} \in \mathbb{R}^E$ is a flow and $R \subseteq V \setminus \{s\}$ such that $r \in R$, then*

$$f_{\mathbf{x}}(r) = \mathbf{x}(\delta^+(R)) - \mathbf{x}(\delta^-(R)).$$

In particular, if $\mathbf{x}$ is feasible, then $f_{\mathbf{x}}(r) \le \mathbf{u}(\delta^+(R))$.

Proof. We have

$$\mathbf{x}(\delta^+(R)) - \mathbf{x}(\delta^-(R)) = \sum_{e\in\delta^+(R)} x_e - \sum_{e\in\delta^-(R)} x_e = \sum_{v\in R}\Big(\sum_{e\in\delta^+(v)} x_e - \sum_{e\in\delta^-(v)} x_e\Big)$$

since the contributions of those edges e having both endpoints in R cancel out.

The first claim now follows from the assumption that all nodes $v \in R\backslash\{r\}$ have net outflow zero. The second claim is an immediate consequence of the condition $\mathbf{0} \le \mathbf{x} \le \mathbf{u}$.

◇

If the subset $R \subseteq V$ contains r but not s, we say that R induces an $r-s$ *cut* $\delta(R) \subseteq E$ and we call

$$\mathrm{cap}(R) = \mathbf{u}(\delta^+(R)) = \sum_{e\in\delta^+(R)} u_e$$

its *capacity*. With this terminology, Lemma 1.4 says that the maximum value $f_{\mathbf{x}}(r)$ of a feasible flow $\mathbf{x}$ is less than or equal to the minimum capacity of an $r-s$ cut. The so-called *Max Flow–Min Cut Theorem* of the network pioneers Ford and Fulkerson [24] states that, in fact, equality holds.

THEOREM 1.6 (Ford and Fulkerson)**.** *The maximum flow value equals the minimum capacity of an $r-s$ cut. Moreover, if all capacities c_e ($e \in E$) are integral, then an integral maximum flow $\mathbf{x} \in \mathbb{Z}_+^E$ exists.*

The proof of Theorem 1.6 is constructive, *i.e.*, we will describe an algorithm for constructing a feasible flow with value equal to some cut capacity. The underlying idea is to increase a current feasible flow $\mathbf{x} \in \mathbb{R}^E$ along "augmenting paths" as follows. Suppose $\mathbf{x} \in \mathbb{R}^E$ is a feasible flow (initially, $\mathbf{x} = \mathbf{0}$). Let

$$E^+ = \{e \in E \mid x_e < u_e\} \quad \text{and} \quad E^- = \{e \in E \mid x_e > 0\}$$

be the sets of edges on which $\mathbf{x}$ can (still) be increased or decreased without violating the capacity constraints. If $0 < x_e < u_e$, then e is, by definition, contained in both E^+ and E^-.

An *augmenting path* (relative to the current flow $\mathbf{x}$) is an $r-s$ path $P \subseteq E$ with $P^+ \subseteq E^+$ and $P^- \subseteq E^-$. Given such a path $P \subseteq E$, we *augment* $\mathbf{x}$ along P, *i.e.*, we increase $\mathbf{x}$ on P^+ and decrease $\mathbf{x}$ on P^- until some edge $e \in P$ becomes *tight*, *i.e.*, $\mathbf{x}$ reaches its upper bound u_e (in case $e \in P^+$) or $\mathbf{x}$ reaches its lower bound 0 (in case $e \in P^-$):

Augment x along P

Increase $\mathbf{x}$ on P^+ and decrease $\mathbf{x}$ on P^- by an amount of

$$\varepsilon = \min\{\min_{e\in P^+} u_e - x_e\ ,\ \min_{e\in P^-} x_e\} > 0\ .$$

Augmenting along P results in a feasible flow $\mathbf{x}'$ with larger flow value $f_{\mathbf{x}'}(r) = f_{\mathbf{x}}(r) + \varepsilon$.

EX. 1.15. *Show that augmenting paths can be computed by applying a shortest path algorithm. (Reverse edges in E^-.)*

Augmenting Path Algorithm	
INIT:	$\mathbf{x} = \mathbf{0}$
ITER:	WHILE there exists an augmenting path P DO augment $\mathbf{x}$ along P

LEMMA 1.5. *Assume that the current flow $\mathbf{x} \in \mathbb{R}^E$ does not allow any augmenting path. Then $\mathbf{x}$ is maximal and its flow value $f_{\mathbf{x}}(r)$ equals the capacity of an $r-s$ cut.*

Proof. In view of Lemma 1.4, it suffices to prove the second part of the claim. To this end, consider the way we search for augmenting paths. Starting with $R_0 = \{r\}$, we successively construct the sets R_k ($k \geq 0$) of nodes that can be reached from r along "augmenting paths" of length k, *i.e.*, $r - v$ paths $P \subseteq E$ with $|P| = k$ and $P^+ \subseteq E^+$ and $P^- \subseteq E^-$. After at most n steps we either end up with $s \in R_k$ for some $k \leq n$ (in which case an augmenting path is found) or we get stuck with a node set

$$R = \bigcup_{i=0}^{k} R_i \quad \text{such that} \quad \begin{cases} \mathbf{x} = \mathbf{u} & \text{on} \quad \delta^+(R) \\ \mathbf{x} = \mathbf{0} & \text{on} \quad \delta^-(R), \end{cases}$$

i.e., the value of the current flow $\mathbf{x}$ equals the capacity of the $r-s$ cut induced by R.

◇

REMARK. Consider the network as indicated in Figure 1.6. The augmenting path method could start with $\mathbf{x} = \mathbf{0}$ and alternatingly augment $\mathbf{x}$ along the paths $r - v - w - s$ and $r - w - v - s$. A maximum flow would be found after 2^{100} iterations.
This example shows that without further specifications, the augmenting path method is not "efficient". The situation is even worse. If we allow arbitrary real capacities, one can design networks on which the above algorithm may perform an infinite number of iterations. The corresponding sequence of feasible flows will of course converge (as the flow values increase), but even the limiting flow might be suboptimal. (Details can be found in [2] or [54].) So the augmenting path algorithm as described above is not an algorithm in the strict sense. In particular, we have not yet proved the Max Flow–Min Cut Theorem 1.6 (except in the case of integral capacities; in this case the proof follows by noticing that the value of the flow must increase at least by 1 in each step).

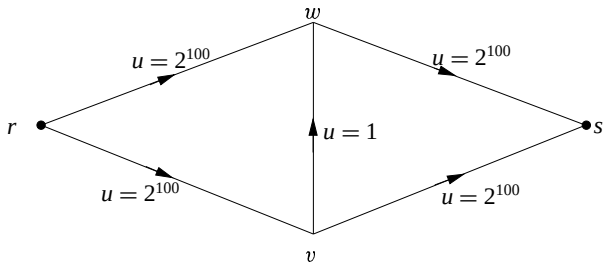

FIGURE 1.6. *A simple max flow problem*

According to the preceding Remark, an algorithm based on the construction of augmenting paths may be extremely inefficient. Fortunately, the way we search for augmenting paths (*cf.* Ex. 1.15 or the proof of Lemma 1.5) actually provides us with *shortest* augmenting paths in each step. Using shortest augmenting paths, we would find a maximum flow for the network in Figure 1.6 after only two(!) augmentations. In general, augmentation along shortest paths results in an efficient algorithm.

Shortest Augmenting Path Algorithm

INIT: $\mathbf{x} = \mathbf{0}$

ITER: WHILE augmenting paths exist DO
Augment $\mathbf{x}$ along a shortest augmenting path

The following result together with Lemma 1.5 finally establishes the Max Flow–Min Cut Theorem 1.6.

THEOREM 1.7. *The shortest augmenting path algorithm computes a maximum flow (for arbitrary real capacities) after at most nm augmentations, where $n = |V|$ and $m = |E|$.*

Proof. The crucial point is to analyze how shortest augmenting paths change from one iteration to the next. Assume that $\mathbf{x} \in \mathbb{R}^E$ is the current feasible flow with a shortest augmenting path P of length $|P| = k$. We will show that after at most m augmenting steps the length of shortest augmenting paths must increase. Since a shortest augmenting path has length at most $n - 1$, the Theorem then follows.

Consider the current flow $\mathbf{x}$. Any augmenting path with respect to $\mathbf{x}$ corresponds to a directed $r - s$ path $P^{\rightarrow}$ in

$$E^{\rightarrow} = E^{\rightarrow}(\mathbf{x}) = E^{+} \cup \{(w, v) \mid (v, w) \in E^{-}\}.$$

Let $F^{\rightarrow} \subseteq E^{\rightarrow}$ be the union of all (currently) shortest $r - s$ paths $P^{\rightarrow} \subseteq E^{\rightarrow}$ (of length k). When we augment $\mathbf{x}$ along $P^{\rightarrow} \subseteq F^{\rightarrow}$, at least one edge $e \in P$ becomes

tight and leaves $F^{\rightarrow}$. As the flow changes along P, new edges (antiparallel to edges in $P^{\rightarrow}$) may enter E^+ or E^-. None of these, however, enters $F^{\rightarrow}$ (*cf.* Lemma 1.3). Hence $|F^{\rightarrow}|$ decreases by at least one in each step until $E^{\rightarrow}$ contains no longer any directed $r-s$ path of length k (or less).

◇

1.3.1. Applications. To illustrate the scope of the network flow model, we give two examples of combinatorial problems that can be solved with the augmenting flow algorithm.

Maximum Bipartite Matching. Given two disjoint (node) sets U and W and a set E of (undirected) edges joining elements in U with elements in W, we are to find the maximum number of pairs $(u, w) \in E$ that can be formed in such a way that each $u \in U$ and each $w \in W$ occurs in at most one pair. We can formulate this problem as a max flow problem as follows. First, orient all edges from U towards W. Then add a source r and sink s as indicated below. All edges have capacity 1.

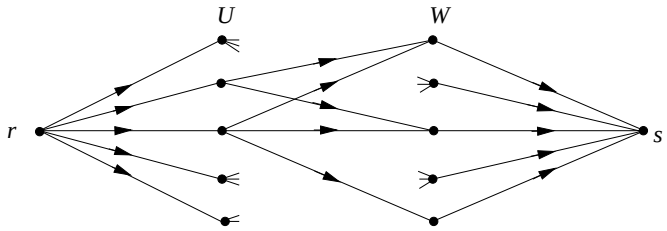

FIGURE 1.7. *The corresponding network*

EX. 1.16. *Show that the value of a maximum flow in the network described above equals the maximum number of pairs we are looking for. (Use the integrality of the solution guaranteed in Theorem 1.6.)*

Scores in Sports Competitions. Consider the following problem (a simplified version of which is mentioned in [12]). There are sports teams $T_0, \ldots, T_n$ playing matches against each other during one season according to a fixed schedule. After each match, two scores are distributed among the participants; 2 : 0 scores for the winner and 1 : 1 in case of a tie. At a certain point in time during the season, one may ask whether a team, say T_0, has still a chance of ending up first. More precisely, assume that T_i has current score $s_i \in \mathbb{Z}_+$ $(i = 0, \ldots, n)$. Assume that T_0 wins all his remaining matches, resulting in a final score $\overline{s}_0 \in \mathbb{Z}_+$ (and current score s_i for T_i, $i \neq 0$). We then ask whether the other teams $T_1, \ldots, T_n$ can possibly end up with all having final scores $\overline{s}_i \leq \overline{s}_0$ $(i = 1, \ldots, n)$, *i.e.*, each T_i collects at most $u_i = \overline{s}_0 - s_i$ additional scores in the remaining matches, say $M_1, \ldots, M_k$.

We obtain a max flow model as follows. Each team T_i $(i = 1, \ldots, n)$ and each of the remaining matches M_j is represented by a node. Each node corresponding to

a match has two entering arcs coming from the two teams playing against each other in this match. Finally, we add a source and sink.

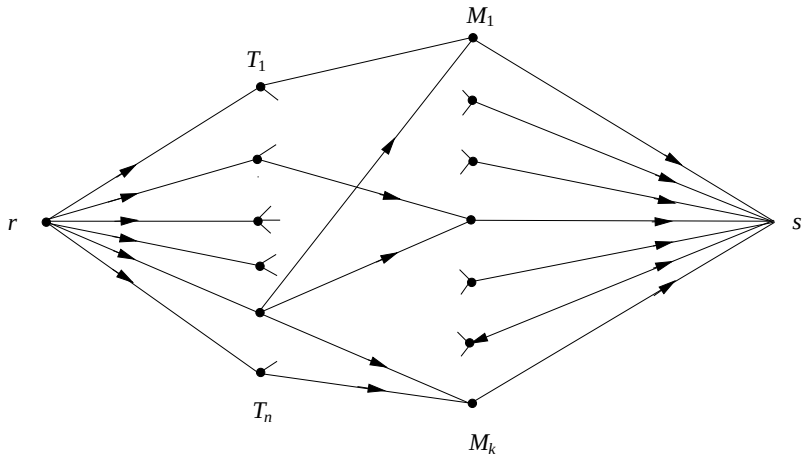

FIGURE 1.8. *The corresponding network*

Each edge from r to T_i has capacity u_i. All other edges have capacity 2.

EX. 1.17. *Show that there is a possibility for the teams $T_1, \ldots, T_n$ to finish the season so that each T_i collects at most u_i additional score points if and only if the above network has a max flow of value $2k$. (As in Ex. 1.16, use the integrality property in Theorem 1.6).*

REMARK. This approach fails for soccer competitions under the FIFA rules allowing 3 : 0 for the winner and 1 : 1 in case of a tie. Indeed, passing from 2 : 0 to 3 : 0 makes the problem *NP*-hard (*cf.* [48]).

1.4. Min Cost Flows

We will now extend our network model by allowing a cost vector $\mathbf{c} \in \mathbb{R}^E$, where the *cost* c_e of an edge $e \in E$ is interpreted as the cost of sending one unit of flow through this edge. (Negative edge costs are permitted). The *Min Cost Flow Problem* asks for an $r-s$ flow $\mathbf{x}$ of prescribed value $f = f_{\mathbf{x}}(r) \geq 0$ that minimizes the *total cost*

$$\sum_{e \in E} c_e \cdot x_e = \mathbf{c}^T \mathbf{x} \ .$$

With $\mathbf{b} \in \mathbb{R}^V$ defined by $b_r = -f$, $b_s = +f$ and $b_v = 0$ else, we can write the Min Cost Flow Problem as

$$(P) \qquad \min \ \mathbf{c}^T \mathbf{x} \quad \text{s.t.} \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \quad \mathbf{0} \leq \mathbf{x} \leq \mathbf{u} \ .$$

REMARK. Note the similarity with the min cost path problem (P) on p. 12. Indeed, for $f = 1$ (which we can achieve by scaling $\mathbf{x}$ and $\mathbf{u}$ if we want), problem (P) here can be viewed as a "min cost path problem with capacity bounds".

We start our investigations with some simplifying assumptions. First note that we may remove all edges e with capacity $u_e = 0$. Furthermore, let us assume (*cf.* Ex. 1.18) that every $v \in V$ can be reached from r along a directed path. So we assume that our network contains a directed spanning tree $T \subseteq E$ rooted at r and that $u_e > 0$ holds for all $e \in E$.

Ex. 1.18. *Let $S \subseteq V \setminus \{r\}$ be the set of vertices that cannot be reached from r along a directed path. Hence $\delta^-(S) = \emptyset$. Show that the min cost flow problem decomposes into a min cost flow problem on $D[V \setminus S]$ with value f and min cost flow problems on the strongly connected components in $D[S]$ with value 0 (with sources and sinks chosen arbitrarily in each component).*

Secondly, note that (P) may be infeasible (in case the maximum $r-s$ flow value is strictly less than f). We circumvent this problem by applying the following simple trick: We introduce an *artificial* edge from r to s with capacity f and large cost $c > 0$, large enough to make sure that a min cost flow of value f will use this artificial edge only if necessary, *i.e.*, only if the original network has maximum $r-s$ flow less that f. For example, it suffices to take

$$c > \sum_{e \in E} |c_e| \ .$$

(The artificial edge may be parallel to some existing edge.) In the following, we will assume that such an (artificial) edge is present in E.

Our next step characterizes basic feasible solutions. We shall see that these are exactly the tree solutions, where a feasible solution $\mathbf{x} \in \mathbb{R}^E$ of (P), *i.e.*, a feasible flow $\mathbf{x}$ of value f, is a *tree solution* if there exists a (not necessarily directed) spanning tree $T \subseteq E$ such that $x_e = 0$ or $x_e = u_e$ for each non-tree edge $e \notin T$. Equivalently, the set $F = \{e \in E \mid 0 < x_e < u_e\} \subseteq E$ is circuit free (and can thus be extended to a spanning tree).

For example, setting $x_e = f$ on the artificial edge and $x_e = 0$ on all other edges yields a tree solution $\mathbf{x} \in \mathbb{R}^E$ that equals zero on all edges of the directed spanning tree T rooted at r (whose existence we assume) and equals its lower or upper bounds outside T. This tree solution will be used as the initial solution in the algorithm described later.

Lemma 1.6. *A feasible solution $\mathbf{x}$ of the linear program (P) is a vertex (basic) solution if and only if $\mathbf{x}$ is a tree solution.*

Proof. Recall that the vector $\mathbf{x} \in \mathbb{R}^E$ is a vertex solution (*cf.* p. **??**) of the system

$$\mathbf{A}\mathbf{x} = \mathbf{b},\ -\mathbf{x} \leq \mathbf{0},\ \mathbf{x} \leq \mathbf{u}$$

if and only if the rows $\mathbf{A}_{v\cdot}$, $v \in V$, and the unit vectors $\pm\mathbf{e}_i \in \mathbb{R}^E$, $i \in I \subseteq E$, that correspond to relations satisfied with equality contain a basis of $\mathbb{R}^E$. Since $\operatorname{rank} \mathbf{A} = |V| - 1$ (*cf.* Ex. 1.7), this is equivalent with saying that

$$x_i = 0 \quad \text{or} \quad x_i = u_i \quad \text{for } i \in I \text{ , with } |I| = |E| - |V| + 1$$

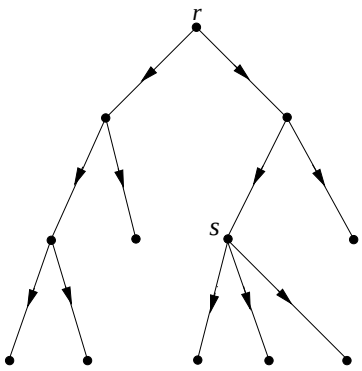

FIGURE 1.9. *The initial tree T*

and the columns $\mathbf{A}_{\cdot i}$ with $i \notin I$ form a column basis of $\mathbf{A}$. By Theorem 1.4, the latter is equivalent with the fact that the $\mathbf{A}_{\cdot i}$, $i \notin I$, correspond to a spanning tree.

◇

EX. 1.19. *Suppose* $\mathbf{x} \in \mathbb{R}^E$ *is a feasible solution of* (P) *but not a tree solution. So there exists a circuit* $C \subseteq \{e \in E \mid 0 < x_e < u_e\}$ *with incidence vector, say,* $\mathbf{z} \in \mathbb{R}^E$. *Show that* $\mathbf{x}$ *is a proper convex combination of the feasible solutions* $\mathbf{x} + \varepsilon\mathbf{z}$ *resp.* $\mathbf{x} - \varepsilon\mathbf{z}$, $\varepsilon > 0$ *(and consequently,* $\mathbf{x}$ *is not a vertex).*

Next we characterize optimal tree solutions. As in the case of min cost paths, we consider the dual program and node potentials. Here, our dual pair of programs is

$$\begin{array}{llrcl} (P) & \min & \mathbf{c}^T\mathbf{x} & & \\ & s.t. & \mathbf{A}\mathbf{x} & = & \mathbf{b} \\ & & \mathbf{x} & \leq & \mathbf{u} \\ & & \mathbf{x} & \geq & \mathbf{0} \end{array} \qquad \text{and} \qquad \begin{array}{llrcl} (D) & \max & \mathbf{y}^T\mathbf{b} - \mathbf{z}^T\mathbf{u} & & \\ & s.t. & \mathbf{y}^T\mathbf{A} - \mathbf{z}^T & \leq & \mathbf{c}^T \\ & & \mathbf{z} & \geq & \mathbf{0}. \end{array}$$

Apart from the additional edge variables z_e, corresponding to the capacity constraints $x_e \leq u_e$, the dual problem (D) looks like the dual of the min cost path problem (which is not surprising). In some sense, the new dual variables z_e play only a secondary role: If $\mathbf{y} \in \mathbb{R}^V$ is any given node potential, there is a unique "optimal" choice for a corresponding $\mathbf{z} \in \mathbb{R}^E$ such that $(\mathbf{y}, \mathbf{z})$ is dually feasible. Since $\mathbf{u} > \mathbf{0}$, we want to choose each $z_e \geq 0$ as small as possible so as to maximize the dual objective function. In view of the constraints $\mathbf{y}^T\mathbf{A} - \mathbf{z} \leq \mathbf{c}^T$ we therefore choose $\mathbf{z} \in \mathbb{R}^E$ according to

$$z_e = \begin{cases} 0 & \text{if } \mathbf{y}^T\mathbf{A}_{\cdot e} \leq c_e \\ \mathbf{y}^T\mathbf{A}_{\cdot e} - c_e & \text{otherwise.} \end{cases} \tag{1.5}$$

As in the case of min cost paths, we may always restrict ourselves to *normalized* node potentials $\mathbf{y} \in \mathbb{R}^V$, satisfying $y_r = 0$.

Let now $\mathbf{x} \in \mathbb{R}^E$ be a feasible tree solution of (P). The corresponding tree $T \subseteq E$ uniquely determines a normalized potential $\mathbf{y} \in \mathbb{R}^V$ satisfying $\mathbf{y}^T \mathbf{A}_{\cdot e} = c_e$ for all $e \in T$, *i.e.*,

$$y_w - y_v = c_e \qquad \text{for all } e = (v, w) \in T. \tag{1.6}$$

As before, we can easily compute $\mathbf{y}$ along paths in T, starting with the root r and $y_r = 0$. Note that these paths are in general not directed since T need not be directed. If $v \in V$ and $P_v \subseteq T$ is the unique $r - v$ path in T, then $P_v = P_v^+ \cup P_v^-$, where P_v^+ is the set of edges directed from r towards v and P^- is the set of oppositely directed edges. Computing the $\mathbf{y}$-values along P_v, we find

$$y_v = c(P_v^+) - c(P_v^-).$$

The node potentials y_v can be interpreted as follows. Suppose our current T and corresponding tree solution $\mathbf{x}$ are such that we could (still) send additional flow from r to each node $v \in V$ along the unique $r - v$ path $P_v \subseteq T$ without violating the capacity constraints. In other words, suppose

$$\mathbf{x} < \mathbf{u} \quad \text{on} \quad T^+ \qquad \text{and} \qquad \mathbf{x} > \mathbf{0} \quad \text{on} \quad T^-.$$

In this case, T and $\mathbf{x}$ are called *strongly feasible*. (Note that our initial T and $\mathbf{x}$ are strongly feasible since $T = T^+$ and $\mathbf{x} = \mathbf{0}$ on T (*cf.* Figure 1.9)). To send additional flow from r to v along P_v, we would increase $\mathbf{x}$ on P^+ and decrease $\mathbf{x}$ on P^-. Sending one unit of flow from r to v this way would therefore increase the total cost by $y_v = c(P_v^+) - c(P_v^-)$. In this sense, y_v can be interpreted as the "current cost of one unit flow" at $v \in V$.

Having computed the potential $\mathbf{y} \in \mathbb{R}^V$ corresponding to T *via* (1.6), we determine the corresponding $\mathbf{z} \in \mathbb{R}^E$ according to (1.5). Since $\mathbf{x}$ and $(\mathbf{y}, \mathbf{z})$ are primally resp. dually feasible, we obtain the following weak duality inequality:

$$\mathbf{c}^T\mathbf{x} \;\geq\; (\mathbf{y}^T\mathbf{A} - \mathbf{z}^T)\mathbf{x} \;=\; \mathbf{y}^T\mathbf{A}\mathbf{x} - \mathbf{z}^T\mathbf{x} \;\geq\; \mathbf{y}^T\mathbf{b} - \mathbf{z}^T\mathbf{u} \tag{1.7}$$

with equality if and only if $\mathbf{x}$ and $(\mathbf{y}, \mathbf{z})$ are optimal.

Consider (1.7) in more detail. The second inequality in (1.7) can only be strict if we have $z_e > 0$ and $x_e < u_e$ for some $e \in E$. Since $\mathbf{z} = \mathbf{0}$ holds on T (by definition of $\mathbf{y}$ and $\mathbf{z}$), this can only happen for $e \in E \backslash T$. Furthermore, $z_e > 0$ is equivalent to $\mathbf{y}^T\mathbf{A}_{\cdot e} > c_e$. Summarizing, the second inequality in (1.7) is strict if and only if there is some $e = (v, w) \in E \backslash T$ with

$$y_w > y_v + c_e \qquad \text{and} \qquad x_e < u_e.$$

The first inequality in (1.7) is strict if and only if $x_e > 0$ and $c_e > \mathbf{y}^T\mathbf{A}_{\cdot e} - z_e$ holds for some $e \in E$. In view of (1.5) and (1.6), the latter can occur only if $e \in E \setminus T$ and $z_e = 0$. So the first inequality in (1.7) is strict if and only if there is some $e = (v, w) \in E \backslash T$ with

$$y_w < y_v + c_e \qquad \text{and} \qquad x_e > 0.$$

Summarizing, we have deduced the following characterization of optimal solutions.

THEOREM 1.8. *Let* $\mathbf{x}$ *be a feasible tree solution corresponding to* T *and let* $(\mathbf{y}, \mathbf{z})$ *be defined by (1.5) and (1.6). Then* $\mathbf{x}$ *and* $(\mathbf{y}, \mathbf{z})$ *are optimal for (P) and (D) if and only if each non-tree edge* $e = (v, w) \in E \backslash T$ *satisfies the following two conditions:*

(i) $y_w > y_v + c_e \Rightarrow x_e = u_e$
(ii) $y_w < y_v + c_e \Rightarrow x_e = 0.$

◇

These optimality conditions fit our interpretation of y_v as the cost of one unit of flow at v nicely: If flow is more expensive at w as compared to the cost at v plus the cost of transporting it from v to w along e, then we should send as much flow as possible through e. In the opposite case we should send the least possible amount (namely 0) through e.

The same intuition also tells us what to do in case $\mathbf{x}$ and $(\mathbf{y}, \mathbf{z})$ are not yet optimal. Suppose $e \in E \backslash T$ violates either (i) or (ii) of Theorem 1.8. We then call e an *infeasible* edge of TYPE 1 resp. TYPE 2. So an infeasible edge $e = (v, w) \in E \backslash T$ yields one of the following (recall that $\mathbf{x} = \mathbf{u}$ or $\mathbf{x} = \mathbf{0}$ outside T):

TYPE 1: $y_w > y_v + c_e$ but $x_e = 0$
TYPE 2: $y_w < y_v + c_e$ but $x_e = u_e$.

In the first case, we try to increase x_e. In the second case, we try to decrease x_e. Let $C \subseteq T \cup e$ be the unique circuit oriented such that $e \in C^+$ if e is of TYPE 1 and $e \in C^-$ if e is of TYPE 2. *Increasing* $\mathbf{x}$ *along* C means to increase $\mathbf{x}$ on C^+ and decrease $\mathbf{x}$ on C^- by some amount $\varepsilon \geq 0$ until some edge $h \in C$ becomes tight, *i.e.*, $\mathbf{x}$ reaches its upper capacity bound $x_h = u_h$ or lower capacity bound $x_h = 0$ on h. This adjustment results in a new tree solution $\mathbf{x}'$ with tree $T' = (T \cup e) \backslash h$.

Passing from a basic feasible solution $\mathbf{x}$ corresponding to the tree T to a "neighboring" basic feasible solution $\mathbf{x}'$ corresponding to $T' = (T \cup e) \backslash h$ is a step in the simplex algorithm. Therefore, the algorithm proceeding this way is called the *Network Simplex Algorithm* for min cost flow. We summarize it as follows:

Network Simplex Algorithm

INIT: Start with the initial tree $T = T^+$ and corresponding tree solution $\mathbf{x}$ (with $\mathbf{x} = \mathbf{0}$ on T, *cf.* Figure 1.9).
Compute the potential $\mathbf{y}$ corresponding to T.

ITER: WHILE an infeasible $e \in E \backslash T$ exists DO
BEGIN
Increase $\mathbf{x}$ on $C \subseteq T \cup e$ (by an amount $\varepsilon \geq 0$)
until some $h \in C$ becomes tight.
Replace T by $(T \cup e) \backslash h$ and update $\mathbf{y}$.
END

Cycling and Finiteness. As in the case of min cost path (or general linear programming) problems, also a network simplex step can be *degenerate* in the sense that it does not necessarily change the current primal solution $\mathbf{x}$. This situation occurs when some edges of C are already tight for $\varepsilon = 0$. Such edges are called *blocking* edges as they prevent us from increasing $\mathbf{x}$ along C by a positive amount.

In practice, degenerate steps occur quite often when solving min cost flow problems. Although almost never observed in practice, it is theoretically not impossible that we return to the same tree $T \subseteq E$ after a sequence of degenerate simplex steps. In such a case of *cycling*, the Network Simplex Algorithm would iterate indefinitely. (Hence it is not an "algorithm" in the strict sense!)

Theoretically, cycling could be avoided, for example, with the lexicographic pivot rule (*cf.* Chapter **??**). Alternatively, we may choose the leaving edge $h \in C$ in each iteration according to the so-called *leaving arc rule* we discuss below. This simple rule not only prevents cycling but also ensures that strong feasibility of $\mathbf{x}$ and T is maintained in each step. To derive this rule, let us look at an iteration $T \to (T \cup e) \backslash h$ as indicated in Figure 1.10 below.

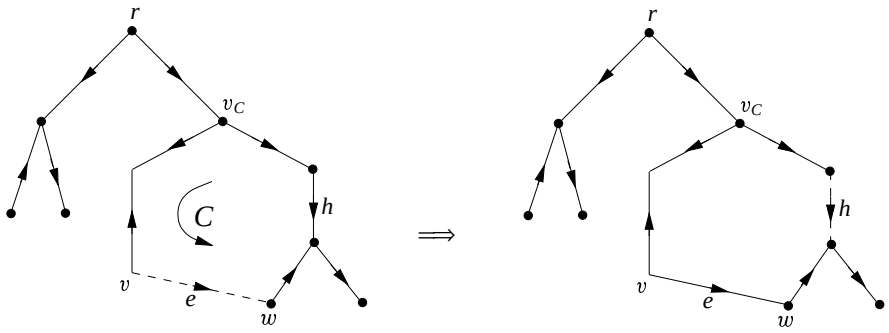

FIGURE 1.10. TYPE 1 *infeasibility* ($y_w > y_v + c_e, x_e = 0$)

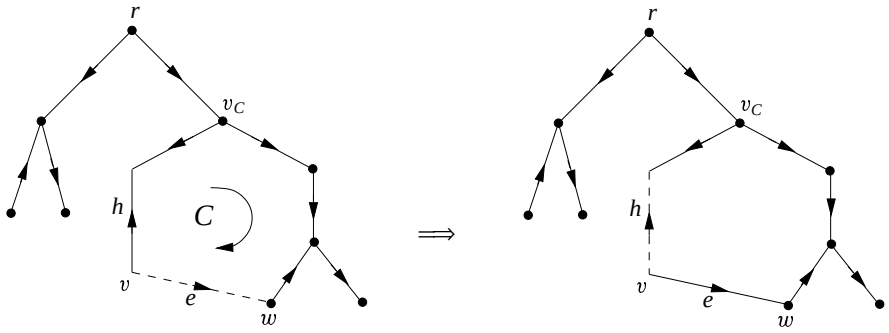

FIGURE 1.11. TYPE 2 *infeasibility* ($y_w < y_v + c_e$, $x_e = u_e$)

Let v_C denote the vertex of C in which the two paths from r to the endpoints of e split (*cf.* Figure 1.10). When we try to increase $\mathbf{x}$ along C, a number of edges on C may become tight at the same time. So the question arises which of these we should chose as the leaving edge $h \in C$.

A moment's thought reveals that if we want to maintain strong feasibility of $\mathbf{x}$ and T, then there is a unique choice for the tight edge $h \in C$ leaving T:

- We must choose h to be the first tight edge we encounter when traversing C according to its orientation, starting in v_C.

This is exactly the *leaving arc rule*.

EX. 1.20. *Show that the Network Simplex Algorithm with the leaving arc rule maintains strong feasibility in each step. (Note that when passing from T to T', some edges in C that are forward edges in T become backward edges in T' and some backward edges in T may become forward edges in T' (which ones?))*

To show that the leaving arc rule also prevents cycling, consider a degenerate step $T \to T' = (T \cup e)\backslash h$. Let P denote the path we traverse when we follow C according to its orientation, starting in v_C until we arrive at e. Since the current flow $\mathbf{x}$ and T are strongly feasible, none of the edges on P can be blocking. So h is the first blocking edge we encounter on C *after* traversing e. (In the situation indicated in Figure 1.10 we would have $x_h = 0$.) In particular, the new tree $T' = (T \cup e)\backslash h$ contains P.

The node potential $\mathbf{y}'$ corresponding to T' can be obtained by updating $\mathbf{y}$ as follows. Removing $e = (v, w)$ splits T' into two components T'_1 and T'_2. One of them, say T'_1, contains the root and the other does not. In view of (1.6), we find that $\mathbf{y}'$ is obtained by increasing or decreasing $\mathbf{y}$ simultaneously on all nodes of T'_2 until the resulting $\mathbf{y}'$ satisfies $y'_w = y'_v + c_e$. Since $P \subseteq T'_1$, we see that $v \in T'_1$ and $w \in T'_2$ in case e is a TYPE 1 infeasible edge and $w \in T'_1$, $v \in T'_2$ in case e is a TYPE 2 infeasible edge (*cf.* Figures 1.10 and 1.11). Hence in both cases, $\mathbf{y}'$ is obtained by *decreasing* the $\mathbf{y}$-values on T'_2 by $|y_w - y_v - c_e| > 0$. So each degenerate step decreases some node potentials. Therefore cycling cannot occur.

As a consequence of the finiteness of the Network Simplex Algorithm we are lead to the following integrality result.

THEOREM 1.9. *Consider the min cost flow problem*

$$(P) \qquad \min \ \mathbf{c}^T\mathbf{x} \quad s.t. \quad \mathbf{Ax} = \mathbf{b},\ \mathbf{0} \leq \mathbf{x} \leq \mathbf{u}\ .$$

If (P) *is feasible and* $\mathbf{u}$ *and* $\mathbf{b}$ *(i.e., the prescribed flow value* f*) are integral, then* (P) *has an integral optimal solution* $\mathbf{x} \in \mathbb{Z}^E$.

Proof. The initial solution is integral and the Network Simplex Algorithm maintains integrality if all capacity bounds are integral.

◇

REMARK. The Network Simplex Algorithm is perhaps the most popular algorithm for solving min cost flow problems in practice. We should point out, however, that it is not theoretically efficient (in the sense of "polynomiality" within the complexity models we discuss in Chapter **??**). Several special polynomial time algorithms for min cost flow problems have been designed, but their treatment falls outside the scope of this book. The interested reader may consult [2] or [12].

The Assignment Problem. The best-known application of the min cost flow model is the *assignment problem*. Consider two disjoint sets I and J consisting of n elements each. We are to assign the elements in I to those in J in such a way that each $i \in I$ is assigned to exactly one $j \in J$ and each $j \in J$ is assigned to exactly one $i \in I$. Assigning $i \in I$ to $j \in J$ costs a certain amount c_{ij}. We want an assignment that minimizes the total cost. Such an assignment can be found as a min cost flow as follows. We introduce directed edges (i, j) for $i \in I$ and $j \in J$. Then we add a root r and sink s as we did for the maximum bipartite matching problem.

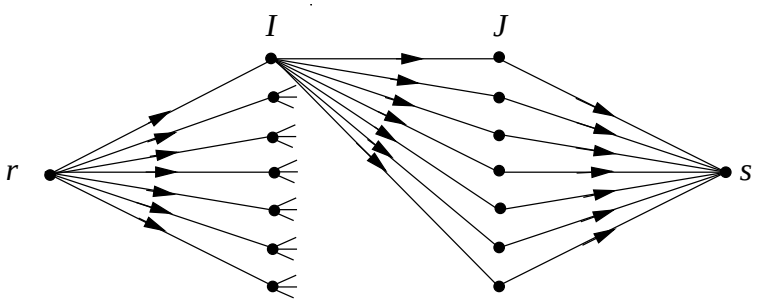

FIGURE 1.12. *The corresponding network*

The edges from I to J have edge costs c_{ij}. All other edges have cost zero. Each edge has upper capacity $u = 1$.

The Network Simplex Algorithm can then be used to compute a cost minimal flow of value $n = |I| = |J|$. The solution $\mathbf{x}$ will be integral (*cf.* Theorem 1.9), *i.e.*,

each edge has flow $x_e = 0$ or 1. Those edges from I to J which carry a flow of 1 correspond to an optimal assignment.

List of frequently used Symbols

$\mathbb{R}$	set of real numbers
$\mathbb{N}, \mathbb{Z}, \mathbb{Q}$	set of natural, integer, rational numbers
$\mathbb{R}^n$	Euclidean n-space
$\mathbb{R}^n_+$	set of non-negative vectors in $\mathbb{R}^n$
$\mathbb{R}^E$	set of real vectors indexed by the set E
$\mathbb{R}^{m\times n}$	set of real $m \times n$ matrices
$\mathbb{S}^{n\times n}$	set of real symmetric $n \times n$ matrices
$\mathbf{x} \in \mathbb{R}^n$	column vector with components $x_1, \ldots, x_n$
$\|\mathbf{x}\|$	Euclidean norm
$U_\varepsilon(\mathbf{x})$	ε-neighborhood of $\mathbf{x}$
$\mathbf{e}_1, \ldots, \mathbf{e}_n$	standard unit vectors in $\mathbb{R}^n$
$\langle \mathbf{x}\|\mathbf{y}\rangle$	inner product
$\mathbf{x}^T\mathbf{y}$	standard inner product in $\mathbb{R}^n$
$\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m\times n}$	real $(m \times n)$ matrix
$\mathbf{A}^T$	transpose of $\mathbf{A}$
$\mathbf{A}_{i\cdot}$, $\mathbf{A}_{\cdot j}$	row vectors, column vectors of $\mathbf{A}$
$\mathbf{A} \circ \mathbf{B} = \sum_i \sum_j a_{ij}b_{ij}$	"inner product" of matrices
$\|\mathbf{A}\|_F = \sqrt{\mathbf{A} \circ \mathbf{A}}$	Frobenius norm of the matrix $\mathbf{A}$.
$\mathbf{A} \succeq \mathbf{0}$	positive semidefinite matrix (p.s.d.)
$\mathbf{A} \succ \mathbf{0}$	positive definite matrix
$\mathbf{A} \succeq \mathbf{B}$	$\mathbf{A} - \mathbf{B}$ is positive semidefinite
$\mathbf{I}$	unit matrix
diag $(d_1, \ldots, d_n)$	diagonal matrix
$\boldsymbol{\alpha}$	vector with all components equal to $\alpha \in \mathbb{R}$
span A	linear hull (span) of a set A
aff A	affine hull (affine span) of a set A
cone A	convex cone of a set A
conv A	convex hull of a set A
cl C	closure of a set C
int C	interior of a set C
$L^\perp$	orthogonal complement of L
C^0	dual cone of C
P^{pol}	polar of P

$[\mathbf{x}, \mathbf{y}]$	line segment between $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$
$P(\mathbf{A}, \mathbf{b})$	polyhedron of solutions of $\mathbf{Ax} \leq \mathbf{b}$
$\ker \mathbf{A}$	kernel of the matrix $\mathbf{A}$
$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_n}\right)$	gradient of $f : \mathbb{R}^n \to \mathbb{R}$ at $\mathbf{x}$ (row vector)
$[\nabla f(\mathbf{x})]^T$ or $\nabla^T f(\mathbf{x})$	transpose of the gradient (column vector)
$\nabla f(\mathbf{x}) = \left(\frac{\partial f_i(\mathbf{x})}{\partial x_j}\right)$	Jacobian of $f : \mathbb{R}^n \to \mathbb{R}^m$ at $\mathbf{x}$
$[\nabla f(\mathbf{x})]^T$ or $\nabla^T f(\mathbf{x})$	transpose of the Jacobian
$\nabla_{\mathbf{x}} g(\mathbf{x}, \mathbf{y})$	partial derivative with respect to the $\mathbf{x}$-variable
$\nabla^2 f(\mathbf{x})$	Hessian of $f : \mathbb{R}^n \to \mathbb{R}$ at $\mathbf{x}$
$\partial f(\mathbf{x})$	subdifferential of $f : \mathbb{R}^n \to \mathbb{R}$ at $\mathbf{x}$
log	logarithm to base 2
ln	natural logarithm
$\langle q \rangle$	size of $q \in \mathbb{Q}$
$\langle \mathbf{x} \rangle$	size of $\mathbf{x} \in \mathbb{Q}^n$
$\langle I \rangle$	size of a problem instance I
$[\alpha]$	nearest integer
$\lceil \alpha \rceil$, $\lfloor \alpha \rfloor$	smallest integer $\geq \alpha$ resp. largest integer $\leq \alpha$
$w.l.o.g.$	without loss of generality
$\subseteq$	containment
$\subset$	proper containment

Bibliography

[1] A.V. Aho, J.E. Hopcraft and J.D. Ullman, The design and Analysis of Computer Algorithms, Addison-Wesley, Reading, (1974).

[2] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, Network Flows, Prentice Hall, New Jersey, (1993).

[3] Atkinson K.E., Numerical Analysis, Wiley, New York, (1988).

[4] A. Bazaraa, H. Sherali and C. Shetty, Nonlinear Programming, John Wiley, New York, (1993).

[5] R.E. Bellman, *On a routing problem*, Quarterly of Applied Mathematics, 16, 87-90, (1958).

[6] R.C. Bland, *New fi nite pivoting rules for the simplex method*, Mathematics of Operations Research, 2, 103-107, (1977).

[7] L. Blum, F. Lucker, M. Shub and S. Smale, Complexity and Real Computation, Springer, (1997).

[8] Th. Bröcker, Differentiable germs and catastrophes, London Math. Soc. Lect. Notes Series 17, Cambridge University Press, (1975).

[9] V. Chvatal, *Edmonds polytopes and a hierarchy of combinatorial problems*, Discrete Mathematics 4, 305-337, (1973).

[10] A. Cohen, *Rate of convergence of several conjugate gradients algorithms*, SIAM Journal on Numerical Analysis, 9, 248-259, (1972).

[11] S. Cook, *The complexity of theorem-proving procedures*, Proc. 3rd Ann ACM Symp. on Theory of Computing, ACM, New York, (1971).

[12] W. Cook , W. Cunningham W. Pulleyblank and A. Schrijver, Combinatorial Optimization, John Wiley & Sons, New York, (1998).

[13] R.W. Cottle et al., The Linear Complementarity Problem, Academic Press, Boston, (1992).

[14] H.G. Daellenbach, Systems and Decision Making. John Wiley & Sons, Chichester (1994).

[15] M. Davis, Computability and Unsolvability, MacGraw-Hill, New York, (1958).

[16] J.E. Dennis and R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice Hall, London, (1983).

[17] E. Dijkstra, *A note on two problems in connection with graphs*, Numerische Mathematik 1, 269-271, (1959).

[18] J. Edmonds, *Systems of Distinct Representation and Linear Algebra*, Journal of Research of the National Bureau of Standards, 71B,(4), (1967)

[19] U. Faigle, M. Hunting, W. Kern, R. Prakash and K.J. Supowit, *Simplices by point-sliding and the Yamnitsky-Levin algorithm*, Math. Methods of Operations Research 46, 131-142, (1997).

[20] J. Farkas, *Über die Theorie der einfachen Ungleichungen*, J. für die Reine und Angewandte Mathematik, 124, 1-27, (1902).

[21] W. Feller, *An Introduction to Probability Theory and Its Applications*, John Wiley, New York, 3rd. rev. ed., (1970).

[22] R. Fletcher, Practical Methods of Optimization, John Wiley & Sons, Chichester, (1987).

[23] L.R. Ford, Jr., *Network flow theory*, Paper P-923, RAND Corporation, Santa Monica, (1956).

[24] L.R. Ford and D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, (1962).
[25] L.R. Ford and D.R. Fulkerson, *Maximal flow trough a network*, Canadian Hournal of Math.,8, 399–404, (1956).
[26] J.B. Fourier, Solution d'une question particuliere du calcul des inégalités, Oeuvres II, 317-328, (1826).
[27] B. Ganter and R. Wille, *Formal Concept Analysis*, Springer-Verlag, Berlin, (1999).
[28] M.R. Garey and D.S. Johnson, Computers and Intractibility: A Guided Tour to the Theory of NP-completeness, Freeman, San Francisco, (1979).
[29] A.M. Geoffrion, *Lagrangian relaxation for integer programming*, Mathematical Programming Study 2, 82-114, (1974).
[30] Ph.E. Gill, W. Murray, M.A. Saunders and M.A. Wright, *Inertia controlling methods for general quadratic problems*, SIAM Review 33, 1-36, (1991).
[31] M.X. Goemans and D.P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. Assoc. for Comp. Machinery 42, No. 6, 1115-1145, (1995).
[32] D. Goldfarb and A. Idnani, *A numerical stable dual method for solving strictly convex quadratic programs*, Math. Prog. 27, 1-33, (1983).
[33] G.H. Golub and C.F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press 3rd. ed., (1996).
[34] R.F. Gomory, *An algorithm for integer solutions to linear problems*, in: Recent Advances in Mathematical Programming, (R.L. Graves and P. Wolfe, eds.), McGraw-Hill, New York, 262-302, (1963).
[35] P. Gordan, *Über die Auflösung linearer Gleichungen mit reellen Coefficienten*, Math. Ann. 6, 23-28, (1873).
[36] M. Grötschel, L. Lovasz and A. Schrijver, Geometric Algorithms and Combinatorial Optimization, 2nd edition, Springer, (1993)
[37] J. Gruska, Quantum Computing, McGraw Hill, London, (1999).
[38] Handbook of Semidefinite Programming: Theory, Algorithms and Applications, eds. Wolkowicz, Saigal and Vandenberghe, Kluwer, Boston, (2000).
[39] M. Held and R.M. Karp, *The travelling salesman problem and minimum spanning trees*, Operations Research 18, 1138-1162, (1970).
[40] R. Hettich and K. Kortanek, Semi-infinite programming: Theory, methods and applications, SIAM Review, vol 35, No.3, 380-429, (1993).
[41] R. Horst and H. Tuy, Global Optimization, Springer, Berlin, (1996).
[42] P. Lancaster and M. Tismenetsky, The Theory of Matrices, Academic Press, Boston, (1985).
[43] M. Hunting, U. Faigle and W. Kern, *A Lagrangian relaxation approach to the edge weighted clique problem*, European J. of Operational Research, 131, 119-131, (2001).
[44] R.G. Jeroslow, *There cannot be any algorithm for integer programming with quadratic constraints*, Operations Research 21,221-224, (1973).
[45] F. John, *Extremum problems with inequalities as subsidiaries conditions*. Studies and Essays, Presented to R. Courant on his 60th Birthday January 8, 1948, Interscience, New York, 187-204, (1948).
[46] N. Karmarkar, *A new polynomial time algorithm for linear programming*, Combinatorics, 4, 373-395, (1984).
[47] R.M. Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations (R.E. Miller and J.W. Thatcher, eds.), Plenum Press, New York, (1972).
[48] W. Kern and D. Paulusma, *The new FIFA rules are hard: Complexity aspects of sports competitions*, Discr. Appl. Math. 108, 317-323, (2001).

[49] L.G. Khachian, *Polynomial algorithms in linear programming* (in Russian), Doklady Akademii Nauk SSSR 244, 1093-1096. (English translation: Soviet Mathematics Doklady 20, 191-194), (1979).
[50] V. Klee and G.J. Minty, *How good is the simplex algorithm?*, in Inequalities III, ed. by O. Shisha, Acad. Press, New York, (1972).
[51] H.W. Kuhn, *Nonlinear programming: A historical note*. In: History of Mathematical Programming, J.K. Lenstra *et al.* eds., CWI, Amsterdam, 82-96, (1991).
[52] C. Lemaréchal and F. Oustry, *Semi-defi nite relaxations and Lagrangian duality with applications to combinatorical optimization*, Rapport de Recherche 3710, INRIA, (1999).
[53] G. Lekkerkerker, Geometry of Numbers. North-Holland, Amsterdam, (1969).
[54] L. Lovasz and M. Plummer, Matching Theory, North Holland, Amsterdam, (1986).
[55] D.G. Luenberger, Optimization by Vector Space Methods, John Wiley & Sons, New York, (1969).
[56] D.G. Luenberger, Introduction to Linear and Nonlinear Programming, Addison-Wesley, (1980).
[57] E.M. Macambira and C.C. de Souza, *The edge weighted clique problem: Valid inequalities, facets and polyhedral computations*, European Journal of Operational Research 123, 346-371, (1999).
[58] N. Maratos, *Exact penalty function algorithms for fi nite dimensional and control optimization problems*, Ph.D. Thesis, Imperial College Sci. Tech., University of London, (1978).
[59] R. Motwani and P. Raghavan, Randomized Algorithms, Cambridge Univ. Press, (1995).
[60] T.S. Motzkin, *Beiträge zur Theorie der Linearen Ungleichungen*, Dissertation, University of Basel, Jerusalem, (1936).
[61] J. von Neumann, *Zur Theorie der Gesellschaftsspiele*, Mathematische Annalen 100, 295-320, (1928).
[62] J. von Neumann, *A certain zero-sum game equivalent to the optimal assignment problem*. In: Contributions to the Theory of Games I. H.W. Kuhn and A.W. Tucker, eds., Annals of Mathematics Studies 28, 5-12, (1953).
[63] J. Nocedal and S.J. Wright, Numerical Optimization, Springer (1999).
[64] M. Padberg, *The boolean quadric polytope: some characteristics, facets and relatives*, Math. Progr. 45, 139-172, (1989).
[65] B.T. Polyak, *A general method for solving extremum problems*, Soviet Math. No 8, 593-597, (1966).
[66] V. Pratt, *Every prime has a succinct certifi cate*, SIAM Journal of Computing, 4, 214–220, (1975).
[67] C.R. Rao, Linear Statistical Inference and Its Applications, Wiley, New York, (1973).
[68] C. Roos, T. Terlaky and J.-Ph. Vial , Theory and Algorithms for Linear Optimization, John Wiley & Sons, Chichester, (1997).
[69] W. Rudin, *Principles of Mathematical Analysis*, (third edition), McGraw-Hill, (1976).
[70] A. Schrijver, Theory of Linear and Integer programming, John Wiley, New York, (1986).
[71] P. Spelucci, *Numerische Verfahren der nichtlinearen Optimierung*, Birkhäuser Verlag, Boston, (1993).
[72] D.M. Topkis and A.F. Veinott, *On the convergence of some feasible direction algorithms for nonlinear programming*, SIAM J. Control 5, 268-279, (1967).
[73] A. Turing, *On computable numbers, with application to the Entscheidungsproblem*, Proc. London Math. Soc. Ser. 2, 42, 230-265 and 43, 544-546, (1936).
[74] H. Tuy, *Convex Analysis and Global Optimization*, Kluwer, Dordrecht, (1998).
[75] S.A. Vavasis, Nonlinear Optimization (Complexity issues), Oxford University Press, New York, (1991).
[76] D. Welsh, *Codes and Cryptography*, Clardendon Press, Oxford, (1988).

[77] C.P. Williams and S.H. Clearwater, Explorations in Quantum Computing. Springer-Verlag, Heidelberg, (1998).
[78] P. Wolfe, *On the convergence of gradient methods under constraints*, IBM J. Research and Development, 16, 407-411, (1972).
[79] G.M. Ziegler, Lectures on Polytopes, Springer-Verlag, New York, (1999).
[80] G. Zoutendijk, *Methods of Feasible Directions*, Elsevier, Amsterdam, (1960).

Index