To appear in: Approximation Algorithms, D. Hochbaum, ed.

CHAPTER

4

THE PRIMAL-DUAL METHOD FOR APPROXIMATION ALGORITHMS AND ITS APPLICATION TO NETWORK DESIGN PROBLEMS

Michel X. Goemans   David P. Williamson

Dedicated to the memory of Albert W. Tucker

The primal-dual method is a standard tool in the design of algorithms for combinatorial optimization problems. This chapter shows how the primal-dual method can be modified to provide good approximation algorithms for a wide variety of NP-hard problems. We concentrate on results from recent research applying the primal-dual method to problems in network design.

4.1 INTRODUCTION

In the last four decades, combinatorial optimization has been strongly influenced by linear programming. With the mathematical and algorithmic understanding of linear programs came a whole host of ideas and tools that were then applied to combinatorial optimization. Many of these ideas and tools are still in use today, and form the bedrock of our understanding of combinatorial optimization.

One of these tools is the *primal-dual method*. It was proposed by Dantzig, Ford, and Fulkerson [DFF56] as another means of solving linear programs. Ironically, their inspiration came from combinatorial optimization. In the early 1930s, Egerváry [Ege31] proved a min-max relation for the assignment problem (or the minimum-cost bipartite perfect matching problem) by reducing it to a known min-max result for maximum cardinality matchings. This lead Kuhn to propose his primal-dual "Hungarian Method" for solving the assignment problem [Kuh55], which then inspired Dantzig, Ford, and Fulkerson. Although the primal-dual method in its original form has not survived as an algorithm for linear programming, it has found widespread use as a means of devising algorithms for problems in combinatorial optimization. The main feature of the primal-dual method is that it allows a weighted optimization problem to be reduced to a purely combinatorial, unweighted problem. Most of the fundamental algorithms in combinatorial optimization either use the method or can be understood in terms of it, including Dijkstra's shortest path algorithm [Dij59], Ford and Fulkerson's network flow algorithm [FF56], Edmonds' non-bipartite matching algorithm [Edm65] and, of course, Kuhn's assignment algorithm.

The primal-dual method as described above has been used to solve problems that can be modelled as linear programs; the method simply leads to efficient polynomial-time algorithms for solving these problems. Since *NP*-hard problems cannot be modelled as polynomially-sized linear programs unless $P = NP$, the primal-dual method does not generalize straightforwardly to generate algorithms for the *NP*-hard optimization problems that are the interest of this book. Nevertheless, with modifications the primal-dual method leads to approximation algorithms for a wide variety of *NP*-hard problems. In this chapter we will explain the current state of knowledge about how the primal-dual method can be used to devise approximation algorithms.

One of the benefits of the primal-dual method is that it leads to a very general methodology for the design of approximation algorithms for *NP*-hard problems. Until quite recently, whenever one wanted to design an approximation algorithm, one usually had to tailor-make an algorithm using the particular structure of the problem at hand. However, in the past few years, several general methods for designing approximation algorithms have arisen. The primal-dual method is one of these, and we will see in this chapter that it leads to approximation algorithms for a large number of problems.

Linear programming has long been used to design and analyze approximation algorithms for *NP*-hard problems, particularly for problems which can be naturally formulated as integer programs. Several approximation algorithms from the seventies use linear programming (LP) in their analysis (see [Chv79, Lov75, CFN77], for example). A 1980 paper by Wolsey [Wol80] highlighted the use of linear programming, and showed that several previously known approximation algorithms could be analyzed using linear programming, including Christofides' algorithm for the TSP [Chr76] and Johnson et al.'s bin packing algorithms [JDU$^+$74]. In the eighties, several papers appeared which used the optimum solution of a linear program to derive an integer solution; the most common technique given rounds fractional solutions to integer solutions. The reader can find examples of deterministic rounding and other techniques (as in [Hoc82]) in Chapter 3 of this book, while randomized rounding [RT87] is presented in Chapter 11. In the primal-dual method for approximation algorithms, an approximate solution to the problem and a feasible solution to the dual of an LP relaxation are constructed simultaneously; the performance guarantee is proved by comparing the values of both solutions. Many of the approximation algorithms with an LP-based analysis can be viewed as primal-dual, but the first truly primal-dual approximation algorithm in which the integer primal and the dual solutions are constructed at the same time is the algorithm of Bar-Yehuda and Even [BYE81] for the vertex cover problem. In the past few years, the power of the primal-dual method has become apparent through a sequence of papers developing this technique for *network design problems* [AKR95, GW95a, SVY92, KR93, WGMV95, GGW93, AG94, GGP+94, GW94a, RW95]. This line of research started with a paper by Agrawal, Klein, and Ravi [AKR95], who introduced a powerful modification of the basic method. Our survey will focus mostly on these problems and results.

In basic versions of network design problems we are given a graph $G = (V, E)$ (undirected or directed) and a cost c_e for each edge $e \in E$ (or for each arc in the directed case), and we would like to find a minimum-cost subset E' of the edges E that meets some design criteria. For example, we may wish to find the minimum-cost set of arcs in a directed graph such that every vertex can reach every other vertex; that is, we wish to find the minimum-cost strongly connected subgraph. Network design problems arise from many sources, including the design of various transportation systems (such as highways and mass-transit systems), as well as telephone and computer networks. We direct the reader to the book edited by Ball et al. [BMMN94] for a broad overview of network design problems, models, and algorithms. For the most part, our survey will concentrate on network design problems on undirected graphs $G = (V, E)$ with nonnegative edge costs c_e.

We will present the primal-dual method as developed for network design problems in a somewhat different fashion than in the original references. We isolate the essential ideas or *design rules* present in all these approximation results and develop generic primal-dual algorithms together with generic proofs of their performance guarantees. Once this is in place, it becomes quite simple to apply these algorithms and proofs to a variety of problems, such as the vertex cover problem [BYE81], the edge covering problem [GW94a], the minimum-weight perfect matching problem [GW95a], the survivable network design problem [AKR95, WGMV95], the prize-collecting traveling salesman problem [GW95a], and the minimum multicut problem in trees [GVY93b]. We show that each of these design rules is implicit in several long-known primal-dual algorithms that solve network design problems exactly, namely Dijkstra's shortest s-t path algorithm [Dij59], Edmonds' minimum-cost branching algorithm [Edm67], and Kruskal's minimum spanning tree algorithm [Kru56]. The generic algorithms reduce to these exact algorithms for these problems.

The survey is structured as follows. In the next section, we review the classical primal-dual method for solving linear programs and optimization problems that can be modelled as linear programs. In Section 4.3, we gradually develop a primal-dual method for the design of approximation algorithm by modifying the classical method and introducing a sequence of design rules. This yields our generic primal-dual algorithm and generic theorems for proving good performance guarantees of the algorithm. We then apply the algorithm and theorems to a number network design problems in following sections. The general model of network design problems that we consider is given in Section 4.4. We introduce a number of network design problems in Sections 4.5 through 4.7, and show that the generic algorithm yields near optimal results. In Section 4.8 we show that the primal-dual method can even be applied to other problems that do not fit in our model, and we conclude in Section 4.9.

4.2 THE CLASSICAL PRIMAL-DUAL METHOD

Before we begin to outline the primal-dual method for approximation algorithms, we first review the classical primal-dual method as applied to linear programs and polynomial-time solvable optimization problems. We refer the reader unfamiliar with the basic theorems and terminology of linear programming to introductions in Chvátal [Chv83] or Strang [Str88, Ch. 8]. For a more detailed description of the primal-dual method for polynomial-time combinatorial optimization problems, see Papadimitriou and Steiglitz [PS82].

Consider the linear program

$$\begin{aligned} \text{Min} \quad & c^T x \\ \text{subject to:} \quad & \\ & Ax \geq b \\ & x \geq 0 \end{aligned}$$

and its dual

$$\begin{aligned} \text{Max} \quad & b^T y \\ \text{subject to:} \quad & \\ & A^T y \leq c \\ & y \geq 0, \end{aligned}$$

where $A \in \mathbf{Q}^{m\times n}$, $c, x \in \mathbf{Q}^n$, $b, y \in \mathbf{Q}^m$, and ^{T} denotes the transpose. For ease of presentation we assume that $c \geq 0$. In the primal-dual method of Dantzig, Ford, and Fulkerson, we assume that we have a feasible solution y to the dual; initially we can set $y = 0$. In the primal-dual method, either we will be able to find a primal solution x that obeys the complementary slackness conditions with respect to y, thus proving that both x and y are optimal, or we will be able to find a new feasible dual solution with a greater objective function value.

First consider what it means for x to be complementary slack to y. Let A_i denote the ith row of A and A^j the jth column of A (written as a row vector to avoid the use of transpose). For the linear program and dual given above, there are two types of complementary slackness conditions. First, there are *primal complementary slackness conditions*, corresponding to the primal variables, namely

$$x_j > 0 \Rightarrow A^j y = c_j.$$

Let $J = \{j | A^j y = c_j\}$. Second, there are *dual complementary slackness conditions*, corresponding to the dual variables, namely

$$y_i > 0 \Rightarrow A_i x = b_i.$$

Let $I = \{i | y_i = 0\}$.

Given a feasible dual solution y we can state the problem of finding a primal feasible x that obeys the complementary slackness conditions as another optimization problem: find a solution x which minimizes the "violation" of the primal constraints and of the complementary slackness conditions. The notion of violation can be formalized in several ways leading to different *restricted primal* problems. For example, the following restricted linear program performs the required role:

$$\begin{aligned} z_{INF} \quad = \quad \text{Min} \quad & \sum_{i \notin I} s_i + \sum_{j \notin J} x_j \\ \text{subject to:} \quad & \\ & A_i x \geq b_i \qquad\qquad i \in I \\ & A_i x - s_i = b_i \qquad i \notin I \\ & x \geq 0 \\ & s \geq 0. \end{aligned}$$

(To ensure feasibility of the restricted primal, we are implicitly assuming the existence of an $x \geq 0$ satisfying $Ax \geq b$.) If this linear program has a solution (x, s) such that the objective function value z_{INF} is 0 then we will have found a primal solution x that obeys the complementary slackness conditions for our dual solution y. Thus x and y are optimal primal and dual solutions, respectively. However, suppose that the optimal solution to the restricted primal has $z_{INF} > 0$. Consider now the dual of the restricted primal:

$$\begin{aligned} \text{Max} \quad & b^T y' \\ \text{subject to:} \quad & \\ & A^j y' \leq 0 \qquad j \in J \\ & A^j y' \leq 1 \qquad j \notin J \\ & y'_i \geq -1 \qquad i \notin I \\ & y'_i \geq 0 \qquad\; i \in I. \end{aligned}$$

Since the optimal solution to its primal has value greater than 0, we know that this program has a solution y' such that $b^T y' > 0$. We will now show that we can find an $\epsilon > 0$ such that $y'' = y + \epsilon y'$ is a feasible dual solution. Thus if we cannot find an x that obeys the complementary slackness conditions, we can find a feasible y'' such that $b^T y'' = b^T y + \epsilon b^T y' > b^T y$; that is, we can find a new dual solution with greater objective function value. Observe that, by definition of I, $y'' \geq 0$ provided that $\epsilon \leq \min_{i \notin I: y'_i < 0}(-y_i/y'_i)$ while, by definition of J, $A^T y'' \leq c$ provided that $\epsilon \leq \min_{j \notin J: A^j y' > 0} \frac{c_j - A^j y}{A^j y'}$. Choosing the smaller upper bound on ϵ, we obtain a new dual feasible solution of greater value, and we can reapply the procedure. Whenever no primal feasible solution obeys the complementary slackness conditions with y, the above restricted primal outputs the least infeasible solution, and this can be used to trace the progress of the algorithm towards finding a primal feasible solution.

Since the method outlined above reduces the solution of a linear program to the solution of a series of linear programs, it does not seem that we have made much progress. Notice, however, that the vector c has disappeared in the restricted primal and its dual. In network design problems, this vector corresponds to the edge-costs. The classical primal-dual method thus reduces weighted problems to their unweighted counterparts, which are often much easier to solve. Furthermore, for combinatorial optimization problems (such as network design problems), these unweighted problems can usually be solved combinatorially, rather than with linear programming. That is, we can use combinatorial algorithms to find an x that obeys the complementary slackness conditions, or failing that, to find a new feasible dual with greater dual objective value. In this way, the method leads to efficient algorithms for these optimization problems.

As an example, we quickly sketch the primal-dual method as it is applies to the assignment problem, also known as the minimum-weight perfect matching problem in bipartite graphs. Suppose we have a bipartite graph $G = (A, B, E)$, with $|A| = |B| = n$, and each edge $e = (a, b)$ has $a \in A$, $b \in B$. We assume that a perfect matching exists in E. Let $c_e \geq 0$ denote the cost of edge e; throughout this section we will use c_e and c_{ab} interchangeably for an edge $e = (a, b)$. We would like to find the minimum-cost set of edges such that each vertex is adjacent to exactly one edge. This problem can be formulated as the following integer program:

$$
\begin{array}{lll}
\text{Min} & \displaystyle\sum_{e \in E} c_e x_e & \\
\text{subject to:} & & \\
& \displaystyle\sum_{b:(a,b) \in E} x_{ab} = 1 & a \in A \\
& \displaystyle\sum_{a:(a,b) \in E} x_{ab} = 1 & b \in B \\
& x_e \in \{0, 1\} & e \in E.
\end{array}
$$

It is well-known that the LP relaxation of this integer program has integer solutions as extreme points (Birkhoff [Bir46], von Neumann [vN53]), so we can drop the integrality constraints and replace them with $x_e \geq 0$. The dual of this LP relaxation is

$$
\begin{array}{ll}
\text{Max} & \displaystyle\sum_{a \in A} u_a + \sum_{b \in B} v_b \\
\text{subject to:} &
\end{array}
$$

$$u_a + v_b \leq c_{ab} \qquad (a,b) \in E.$$

The primal-dual method specifies that we start with a dual feasible solution, in this case $u = v = 0$. Given our current feasible dual solution, we look for a primal feasible solution that obeys the complementary slackness conditions. In this case, we only have primal complementary slackness conditions. Let $J = \{(a,b) \in E : u_a + v_b = c_{ab}\}$. Then the restricted primal is

$$\text{Min} \quad \sum_{a \in A} s_a + \sum_{b \in B} s_b$$

subject to:

$$\begin{array}{ll}
\displaystyle\sum_{b:(a,b)\in E} x_{ab} + s_a = 1 & a \in A \\
\displaystyle\sum_{a:(a,b)\in E} x_{ab} + s_b = 1 & b \in B \\
x_e = 0 & e \in (E - J) \\
x_e \geq 0 & e \in J \\
s \geq 0. &
\end{array}$$

As with the original primal, every basic feasible solution to the restricted primal has every component equal to 0 or 1. This implies that solving the restricted primal reduces to the problem of finding the largest cardinality matching in the bipartite graph $G' = (A, B, J)$. Efficient algorithms are known for finding maximum matchings in bipartite graphs. If we find a perfect matching in G', then we have found an x that obeys the complementary slackness conditions with respect to (u, v), and x and (u, v) must be optimal solutions. Initially, J is likely to be empty and, as a result, our initial primal infeasible solution is $x = 0$. One can show that the infeasibility of x gradually decreases during the course of the algorithm.

The dual of the restricted primal is

$$\text{Max} \quad \sum_{a \in A} u'_a + \sum_{b \in B} v'_b$$

subject to:

$$\begin{array}{ll}
u'_a + v'_b \leq 0 & (a,b) \in J \\
u'_a \leq 1 & a \in A \\
v'_b \leq 1 & b \in B.
\end{array}$$

It can easily be seen that every basic solution (u', v') has all its components equal to ± 1. Given the maximum matching, there is a straightforward combinatorial algorithm to find an optimum solution to this dual. If the optimum value of the restricted primal is not zero then an improved dual solution can be obtained by considering $u'' = u + \epsilon u'$ and $v'' = v + \epsilon v'$, for $\epsilon = \min_{(a,b)\in E-J}(c_{ab} - u_a - v_b)$. It is not hard to see that this choice of ϵ maintains dual feasibility, and it can be shown that only $O(n^2)$ dual updates are necessary before a perfect matching is found in G'. At this point we will have found a feasible x that obeys the complementary slackness conditions with a feasible dual u, v, and thus these solutions must be optimal.

EXERCISE 4.1 Show how to formulate a restricted primal by using only one new variable. Make sure that your restricted primal is always feasible.

4.3 THE PRIMAL-DUAL METHOD FOR APPROXIMATION ALGORITHMS

Most combinatorial optimization problems have natural integer programming formulations. However, unlike the case of the assignment problem, the LP relaxations typically have extreme points which do not correspond to solutions of the combinatorial optimization problem. Therefore, we cannot use the classical primal-dual method to find an optimum integer solution. In this section, however, we will show that a suitable modification of the method is very useful for finding approximate integer solutions. In addition, we will show a sequence of design rules that leads to good approximation algorithms for network design problems.

The central modification made to the primal-dual method is to relax the complementary slackness conditions. In the classical setting described in the previous section, we imposed both primal and dual complementary slackness conditions, and we used the dual of the restricted primal problem to find a direction to improve the dual solution if the complementary conditions were not satisfied. For the design of approximation algorithms, we will impose the primal complementary slackness conditions, but relax the dual complementary slackness conditions. Furthermore, given these conditions, if the current primal solution is not feasible, we will be able to increase the value of the dual.

To illustrate this modification of the method, we will examine a specific combinatorial optimization problem, the *hitting set problem*. The hitting set problem is defined as follows: Given subsets $T_1, \ldots, T_p$ of a ground set E and given a nonnegative cost c_e for every element $e \in E$, find a minimum-cost subset $A \subseteq E$ such that $A \cap T_i \neq \emptyset$ for every $i = 1, \ldots, p$ (i.e. A "hits" every T_i). The problem is equivalent to the more well-known set cover problem in which the goal is to cover the entire ground set with the minimum-cost collection of sets (see Chapter 3).

As we proceed to construct piece by piece a powerful version of the primal-dual method for approximation algorithms, we will along the way "rediscover" many classical (exact or approximation) algorithms for problems that are special cases of the hitting set problem. From these classical algorithms, we will infer design rules for approximation algorithms which we will later show lead to good approximation algorithms for other problems. The particular special cases of the hitting set problem we study are as follows. The *undirected $s-t$ shortest path problem* with nonnegative lengths can be formulated as a hitting set problem by noticing that any $s-t$ path must intersect every $s-t$ cut $\delta(S)$, where

$\delta(S) = \{e = (i,j) \in E : i \in S, j \notin S\}$ and $s \in S$ and $t \notin S$. Thus, we can let E be the edge set of the undirected graph $G = (V,E)$, c_e be the length of the edge e, and $T_1, \ldots, T_p$ be the collection of all $s-t$ cuts, i.e. $T_i = \delta(S_i)$ where S_i runs over all sets containing s but not t. Observe that the feasible solutions consist of subgraphs in which s and t are connected; only *minimal* solutions (i.e. solutions for which no edge can be removed without destroying feasibility) will correspond to $s-t$ paths. The directed $s-t$ path problem can be similarly formulated. The *minimum spanning tree problem* is also a special case of the hitting set problem; here we would like to cover all cuts $\delta(S)$ with no restriction on S. The *vertex cover problem* (see Chapter 3) is the problem of finding a minimum (cardinality or cost) set of vertices in an undirected graph such that every edge has at least one endpoint in the set. The vertex cover is a hitting set problem in which the ground set E is now the set of vertices and T_i corresponds to the endpoints of edge i. In the *minimum-cost arborescence problem*, we are given a directed graph $G = (V,E)$ with nonnegative arc costs and a special root vertex r and we would like to find a spanning tree directed out of r of minimum cost. Here the sets to hit are all r-directed cuts, i.e. sets of arcs of the form $\delta^-(S) = \{(i,j) \in E : i \notin S, j \in S\}$ where $S \subseteq V - \{r\}$. All these special cases, except for the vertex cover problem, are known to be polynomially solvable. Dijkstra's algorithm [Dij59] solves the shortest path problem, Edmonds' algorithm [Edm67] solves the minimum-cost arborescence problem, while Kruskal's greedy algorithm [Kru56] solves the minimum spanning tree problem. For many special cases (again excluding the vertex cover problem), the number of sets to hit is exponential in the size of the instance; we will see shortly that this does not lead to any difficulties.

The hitting set problem can be formulated as an integer program as follows:

$$\begin{aligned} \text{Min} \quad & \sum_{e \in E} c_e x_e \\ \text{subject to:} \quad & \\ & \sum_{e \in T_i} x_e \geq 1 \qquad && i = 1, \ldots, p \\ & x_e \in \{0,1\} && e \in E, \end{aligned}$$

where x represents the incidence (or characteristic) vector of the selected set A, i.e. $x_e = 1$ if $e \in A$ and 0 otherwise. Its LP relaxation and the corresponding dual are the following:

$$\begin{aligned} \text{Min} \quad & \sum_{e \in E} c_e x_e \\ \text{subject to:} \quad & \\ & \sum_{e \in T_i} x_e \geq 1 \qquad && i = 1, \ldots, p \\ & x_e \geq 0 && e \in E, \end{aligned}$$

and

$$\text{Max} \quad \sum_{i=1}^{p} y_i$$

$y \leftarrow 0$
$A \leftarrow \emptyset$
While $\exists k : A \cap T_k = \emptyset$
Increase y_k until $\exists e \in T_k : \sum_{i:e \in T_i} y_i = c_e$
$A \leftarrow A \cup \{e\}$
Output A (and y)

FIGURE 4.1

The basic primal-dual algorithm.

subject to:

$$\sum_{i:e\in T_i} y_i \le c_e \qquad e \in E$$

$$y_i \ge 0 \qquad i = 1, \ldots, p.$$

For the incidence vector x of a set A and a dual feasible solution y, the primal complementary slackness conditions are

$$e \in A \Rightarrow \sum_{i:e\in T_i} y_i = c_e \tag{4.1}$$

while the dual complementary slackness conditions are

$$y_i > 0 \Rightarrow |A \cap T_i| = 1. \tag{4.2}$$

As we said earlier, the central modification made to the primal-dual method is to enforce the primal complementary slackness conditions and relax the dual conditions. Given a dual feasible solution y, consider the set $A = \{e : \sum_{i:e\in T_i} y_i = c_e\}$. Clearly, if A is infeasible then no feasible set can satisfy the primal complementary slackness conditions (4.1) corresponding to the dual solution y. As in the classical primal-dual method, if we cannot find a feasible primal solution given the complementary slackness conditions, then there is a way to increase the dual solution. Here, the infeasibility of A means that there exists k such that $A \cap T_k = \emptyset$. The set T_k is said to be *violated*. By increasing y_k, the value of the dual solution will improve; the maximum value y_k can take without violating dual feasibility is

$$y_k = \min_{e \in T_k} \left\{ c_e - \sum_{i \neq k : e \in T_i} y_i \right\}. \tag{4.3}$$

Observe that $y_k > 0$ since no element e in T_k is also in A. For this value of y_k, at least one element e (the argmin in (4.3)) will be added to A since now $\sum_{i:e\in T_i} y_i = c_e$. We can repeat the procedure until A is a feasible primal solution.

This basic version of the primal-dual method is formalized in Figure 4.1. In the description of the algorithm in the figure, we are adding only one element e at a time to A, although other elements f could satisfy $\sum_{i:f\in T_i} y_i = c_f$. This means that in a later stage such an element f could be added while the corresponding increase of y_l for some $T_l \ni f$ would be 0. This does not affect the algorithm.

The primal-dual method as described is also referred to as a *dual-ascent algorithm*. See for example the work of Erlenkotter [Erl78] for the facility location problem, Wong [Won84] for the Steiner tree problem, Balakrishnan, Magnanti and Wong [BMW89] for the fixed-charge network design problem, or the recent Ph.D. thesis of Raghavan [Rag94].

The main question now is whether the simple primal-dual algorithm described in Figure 4.1 produces a solution of small cost. The cost of the solution is $c(A) = \sum_{e \in A} c_e$ and since e was added to A only if the corresponding dual constraint was tight, we can rewrite the cost as $\sum_{e \in A} \sum_{i: e \in T_i} y_i$. By exchanging the two summations, we get

$$c(A) = \sum_{i=1}^{p} |A \cap T_i| y_i.$$

Since y is a dual feasible solution, its value $\sum_{i=1}^{p} y_i$ is a lower bound on the optimum value z_{OPT} of the hitting set problem. If we can guarantee that

$$|A \cap T_i| \leq \alpha \text{ whenever } y_i > 0 \tag{4.4}$$

then this would immediately imply that $c(A) \leq \alpha z_{OPT}$, i.e. the algorithm is an α-approximation algorithm. In particular, if α can be guaranteed to be 1, then the solution given by the algorithm must certainly be optimal, and equation (4.4) together with primal feasibility imply the dual complementary slackness conditions (4.2). Conditions (4.4) certainly hold if we choose α to be the largest cardinality of any set T_i: $\alpha = \max_{i=1}^{p} |T_i|$. This α-approximation algorithm for the general hitting set problem was discovered by Bar-Yehuda and Even [BYE81]; the analysis appeared previously in a paper of Hochbaum [Hoc82], who gave an α-approximation algorithm using an optimal dual solution. In the special case of the vertex cover problem, every T_i has cardinality two and therefore the algorithm is a 2-approximation algorithm. We refer the reader to the Chapter 3 for the history of these results, as well as additional results on the vertex cover problem and the general set cover problem. The algorithm above is functionally equivalent to the "dual feasible" algorithm of Chapter 3.

Before refining the basic algorithm, we discuss some implementation and efficiency issues. First, since A has at most $|E|$ elements, the algorithm performs at most $|E|$ iterations and outputs a dual feasible solution y with at most $|E|$ nonzero values. This observation is particularly important when there are exponentially many sets T_i (and these sets are given implicitly) as in the case of the $s-t$ shortest path problem or the minimum-cost arborescence problem. In such cases, the algorithm does not keep track of every y_i but only of the nonzero components of y. Also, the algorithm must be able to find a set T_k not intersecting A. If there are many sets to hit, we must have a *violation* oracle: given A the oracle must be able to decide if $A \cap T_i \neq \emptyset$ for all i and, if not, must output a set T_k for which $A \cap T_k = \emptyset$.

For the shortest path problem, the minimum-cost arborescence problem or the network design problems we will be considering, the sets T_i to be hit are naturally associated to vertex sets S_i ($T_i = \delta(S_i)$, or for the minimum-cost arborescence problem, $T_i = \delta^-(S_i)$). For simplicity, we shall often refer to these vertex sets instead of the corresponding cuts; for example, we will say that the set S_i is violated, rather than $T_i = \delta(S_i)$ is violated. Also, we shall denote the dual variable corresponding to the cut induced by S as y_S.

We obtain our first design rule by considering a violation oracle for the $s-t$ shortest path problem. For this problem, the oracle simply computes the connected components of (V, A) and check if s and t belong to the same component; if not, the component containing s (or the one containing t, or the union of components containing s or t) is a violated set. This comment raises the issue of which violated set to select in the basic primal-dual algorithm, when there are several sets which are not hit by A. For network design problems in which the T_i's naturally correspond to vertex sets, a good selection rule is to take among all violated edge sets T one for which the corresponding vertex set S is (inclusion-wise) minimal, i.e. there is no violated S' with $S' \subset S$. We refer to this rule as the *minimal violated set rule*. In the case of the undirected shortest path problem, this rule consists of selecting the connected component containing s, provided that this component does not contain t. Here there is a *unique* minimal violated set, although this is not always the case.

Let us consider the resulting primal-dual algorithm for the shortest path problem in greater detail. Initially, all y_S are 0, $A = \emptyset$, and the minimal violated set is simply $S = \{s\}$. As y_S is increased, the shortest edge (s, i) out of s is selected and added to A. In a later stage, if S denotes the current minimal violated set, an edge (i, j) with $i \in S$ and $j \notin S$ is added to A and the minimal violated set becomes $S \cup \{j\}$ (unless $j = t$ in which case there are no more violated sets). Thus, A is a forest consisting of a single non-trivial component containing s. To see which edges get added to A, it is useful to keep track of a notion of *time*. Initially, time is 0 and is incremented by ϵ whenever a dual variable is increased by ϵ. For every edge e, let $a(e)$ denote the time at which e would be added to A *if* the minimal violated sets were not to change. We refer to $a(e)$ as the *addition time* of edge e. Similarly, let $l(j)$ be the time at which a vertex j would be added to S. Clearly, $l(j)$ is simply the smallest $a(e)$ over all edges e incident to j. The next vertex to be added to S is thus the vertex attaining the minimum in $\min_{j \notin S} l(j)$. As j is added to S, we need to update the $a(.)$ and $l(.)$ values. Only the $a(.)$ values of the edges incident to j will be affected; this makes their update easy. Also, for $k \notin S$, $l(k)$ simply becomes $\min(l(k), l(j) + c_{jk})$. By now, the reader must have realized that the $l(.)$ values are simply the labels in Dijkstra's algorithm [Dij59] for the shortest path problem. Keeping track of the $a(.)$ values is thus not necessary in this case, but will be useful in more sophisticated uses of the primal-dual method.

The primal-dual algorithm with minimal violated set rule thus reduces to Dijkstra's algorithm in the case of the shortest path. Or not quite, since the set A output by the algorithm is not simply an $s-t$ path but is a shortest path forest out of s. The cost of this forest is likely to be higher than the cost of the shortest $s-t$ path. In fact, if we try to evaluate the parameter α as defined in (4.4), we observe that α could be as high as $|V| - 1$, if all edges incident to s have been selected. We should therefore eliminate all the unnecessary edges from the solution. More precisely, we add a *delete step* at the end of the primal-dual algorithm which discards as many elements as possible from A without losing

$y \leftarrow 0$
$A \leftarrow \emptyset$
$l \leftarrow 0$
While $\exists k : A \cap T_k = \emptyset$
  $l \leftarrow l + 1$
  Increase y_k until $\exists e_l \in T_k : \sum_{i: e_l \in T_i} y_i = c_{e_l}$
  $A \leftarrow A \cup \{e_l\}$
For $j \leftarrow l$ downto 1
  if $A - \{e_j\}$ is feasible then $A \leftarrow A - \{e_j\}$
Output A (and y)

FIGURE 4.2

Primal-dual algorithm with reverse delete step.

feasibility. Observe that, in general, different sets could be output depending on the order in which edges are deleted; in this case, we simply keep only the path from s to t in the shortest path forest. It is not difficult to show (this follows trivially from the forthcoming Theorem 4.1) that the resulting $s-t$ path P satisfies $|P \cap \delta(S)| = 1$ whenever $y_S > 0$, implying that the algorithm finds an optimal solution to the problem.

In some cases, however, the order of deletion of elements is crucial to the proof of a good performance guarantee; this leads to our next design rule. We adopt a *reverse delete step* in which elements are considered for removal in the *reverse* order they were added to A. This version of the primal-dual algorithm with the reverse delete step is formalized in Figure 4.2. We first analyze the performance guarantee of this algorithm in general, then show that it leads to Edmonds' algorithm for the minimum-cost arborescence problem.

To evaluate the performance guarantee of the algorithm, we need to compute an upper bound on α as given in (4.4). To avoid any confusion, let A_f be the set output by the algorithm of Figure 4.2. Fix an index i such that $y_i > 0$, and let e_j be the edge added when y_i was increased. Because of the reverse delete step, we know that when e_j is considered for removal, no element e_p with $p < j$ has been already removed. Let B denote the set of elements right after e_j is considered in the reverse delete step. This means that $B = A_f \cup \{e_1, \ldots, e_{j-1}\}$, and that B is a *minimal augmentation* of $\{e_1, \ldots, e_{j-1}\}$, i.e. B is feasible, $B \supseteq \{e_1, \ldots, e_{j-1}\}$ and for all $e \in B - \{e_1, \ldots e_{j-1}\}$ we have that $B - \{e\}$ is not feasible. Moreover, $|A_f \cap T_i| \leq |B \cap T_i|$ and this continues to hold if we maximize over *all* minimal augmentations B of $\{e_1, \ldots, e_{j-1}\}$. Thus, as an upper bound on α, we can choose

$$\beta = \max_{\left\{\substack{\text{infeasible} \\ A \subset E}\right\}} \max_{\left\{\substack{\text{minimal} \\ \text{augmentations } B \text{ of } A}\right\}} |B \cap T(A)|, \tag{4.5}$$

where $T(A)$ is the violated set selected by the primal-dual algorithm when confronted with the set A. We have therefore proved the following theorem:

THEOREM 4.1 The primal-dual algorithm described in Figure 4.2 delivers a feasible solution of cost at most $\beta \sum_{i=1}^{p} y_i \leq \beta z_{OPT}$, where β is given in (4.5).

The reverse delete step has thus allowed us to give a bound on the performance of the algorithm without looking at the entire run of the algorithm, but simply by considering any *minimal augmentation* of a set. As an exercise, the reader is invited to derive the optimality of the primal-dual algorithm for the shortest path problem from Theorem 4.1.

Consider now the minimum-cost arborescence problem. For any subset A of arcs, the violation oracle with minimal violated set rule can be implemented by first computing the strongly connected components and then checking if any such component not containing the root, say S, has no arc incoming to it (i.e. $\delta^-(S) \cap A = \emptyset$). If no such component exists then one can easily derive that A contains an arborescence. Otherwise, the algorithm would increase the dual variable corresponding to such a strongly connected component (observe that we have the choice of which component to select if there are several of them). Any minimal augmentation of A must have only *one* arc incoming to a strongly connected component S, since one such arc is sufficient to reach all vertices in S. Thus the parameter β is equal to 1, and the primal-dual algorithm delivers an optimum solution. This elegant algorithm is due to Edmonds [Edm67]. We should point out that in the case of the arborescence problem, deleting the edges *in reverse* is crucial (while this was not the case for the shortest path problem). The use of the reverse delete step will also be crucial in the design of approximation algorithms for network design problems described in the following sections; in this context, this idea was first used by Klein and Ravi [KR93] and Saran, Vazirani, and Young [SVY92].

Several variants of the primal-dual algorithm described in Figure 4.2 can be designed, without affecting the proof technique for the performance guarantee. One useful variant is to allow the algorithm to increase the dual variable of a set which does not need to be hit. More precisely, suppose we also add to the linear programming relaxation the constraints

$$\sum_{e \in T_i} x_e \geq 1$$

$i = p+1, \ldots, q$, for a collection $\{T_{p+1}, \ldots, T_q\}$ of sets. This clearly may affect the value of the relaxation. Assume we now use the primal-dual algorithm by increasing the dual variable corresponding to any set T_i, where i now runs from 1 to q. Thus, in step 4 of Figure 4.2, a solution A is considered feasible if it hits every set T_i for $i = 1, \ldots, q$. However, in the reverse delete step 9, A only needs to hit every T_i for $i = 1, \ldots, p$. Although the addition of sets T_i's has made the relaxation invalid, we can still use the dual solution we have constructed. Indeed, $\sum_{i=1}^{p} y_i$ is still a lower bound on the optimum value, and, as before, it can be compared to the cost $\sum_{i=1}^{q} |A \cap T_i| y_i$ of the output solution A. The proof technique we have developed for Theorem 4.1 still applies, provided we can guarantee that $A \cap T_i = \emptyset$ for $i = p+1, \ldots, q$. In this case, the performance guarantee will again be β as given by (4.5). As an application, assume that in the minimum-cost arborescence problem, we also include the constraints corresponding to sets S containing the root (this would constitute a formulation for the strongly connected subgraph problem). Then, as long as A does not induce a strongly connected graph, we increase the dual variable corresponding to any strongly connected component with no arc incoming to it (whether or not it contains r). This step is thus independent of the root. It is only in the reverse delete step that we use knowledge of the root. This algorithm still outputs the optimum arborescence (for any specific root r) since it is easy to see that any arc incoming to a strongly connected component containing r and selected by the algorithm will be deleted in the reverse delete step. The algorithm therefore constructs a *single* dual solution proving optimality for *any* root. This observation was made by Edmonds [Edm67]. Another application of this variant of the primal-dual algorithm will be discussed in Section 4.5.

Our final design rule comes from considering the minimum spanning tree problem and the associated greedy algorithm due to Kruskal [Kru56]. In the case of the minimum spanning tree problem, the violation oracle with minimal violated set rule can be implemented by first computing the connected components of (V, A) and, if there are k components where $k > 1$, by selecting any such component, say S. It is easy to see that any minimal augmentation of A must induce a spanning tree if we separately shrink every connected component of (V, A) to a supervertex. The resulting algorithm has a bad performance guarantee since a minimal augmentation of A could therefore have as many as $k-1$ edges incident to S. Recall that Kruskal's greedy algorithm repeatedly chooses the minimum-cost edge spanning two distinct connected components. This choice of edge is equivalent to *simultaneously* increasing the dual variables corresponding to *all* connected components of (V, A), until the dual constraint for an edge becomes tight.

To see this, consider the notion of *time* as introduced for the shortest path problem. As in that context, we let the addition time $a(e)$ of an edge e to be the time at which this edge would be added to A if the collection of minimal violated sets were not to change. Initially, the addition time of e is $c_e/2$ (since the duals are increased on both endpoints of e), and it will remain so as long as both ends are in different connected components of (V, A). The next edge to be added to A is the one with smallest addition time and is thus the minimum-cost edge between two components of (V, A). Thus, the algorithm mimics Kruskal's algorithm.

This suggests that we should revise our primal-dual algorithm and increase *simultaneously and at the same speed* the dual variables corresponding to several violated sets. We refer to this rule as the *uniform increase* rule. This is formalized in Figure 4.3, in which the oracle VIOLATION returns a collection of violated sets whose dual variables will be increased. In the case of network design problems, the study of the minimum spanning tree problem further suggests that the oracle VIOLATION should return *all* minimal violated sets. In the context of approximation algorithms for network design problems, this uniform increase rule on minimal violated sets was first used by Agrawal, Klein, and Ravi [AKR95] without reference to linear programming; its use was broadened and the linear programming made explicit in a paper of the authors [GW95a]. The algorithm of Agrawal et al. can be considered the first highly sophisticated use of the primal-dual method in the design of approximation algorithms.

$y \leftarrow 0$
$A \leftarrow \emptyset$
$l \leftarrow 0$
While A is not feasible
  $l \leftarrow l+1$
  $\mathcal{V} \leftarrow$ VIOLATION(A)
  Increase y_k uniformly for all $T_k \in \mathcal{V}$ until $\exists e_l \notin A : \sum_{i:e_l \in T_i} y_i = c_{e_l}$
  $A \leftarrow A \cup \{e_l\}$
For $j \leftarrow l$ downto 1
  if $A - \{e_j\}$ is feasible then $A \leftarrow A - \{e_j\}$
Output A (and y)

FIGURE 4.3

Primal-dual algorithm with uniform increase rule and reverse delete step.

The analysis of the performance guarantee can be done in a similar way as for the primal-dual algorithm of Figure 4.2. Remember we compared the cost of the solution output A_f, which can be written as $\sum_{i=1}^{p} |A_f \cap T_i| y_i$, to the value $\sum_{i=1}^{p} y_i$ of the dual solution. However, instead of comparing the two summations term by term, we may take advantage of the fact that several dual variables are being increased at the same time. Let $\mathcal{V}_j$ denote the collection of violated sets returned by the oracle VIOLATION in the jth iteration of our primal-dual algorithm of Figure 4.3 and let ϵ_j denote the increase of the dual variables corresponding to $\mathcal{V}_j$ in iteration j. Thus, $y_i = \sum_{j:T_i \in \mathcal{V}_j} \epsilon_j$. We can rewrite the value of the dual solution as

$$\sum_{i=1}^{p} y_i = \sum_{j=1}^{l} |\mathcal{V}_j| \epsilon_j,$$

and the cost of A_f as:

$$\sum_{i=1}^{p} |A_f \cap T_i| y_i = \sum_{i=1}^{p} |A_f \cap T_i| \sum_{j:T_i \in \mathcal{V}_j} \epsilon_j = \sum_{j=1}^{l} \left(\sum_{T_i \in \mathcal{V}_j} |A_f \cap T_i| \right) \epsilon_j.$$

From these expressions (comparing them term by term), it is clear that the cost of A_f is at most the value of the dual solution times γ if, for all $j = 1, \ldots, l$,

$$\sum_{T_i \in \mathcal{V}_j} |A_f \cap T_i| \le \gamma |\mathcal{V}_j|.$$

Again using the reverse delete step, we can replace A_f, which depends on the entire algorithm in an intricate fashion, by any minimal augmentation B of the infeasible solution at the start of iteration j. We have thus proved the following theorem.

THEOREM 4.2 The primal-dual algorithm described in Figure 4.3 delivers a feasible solution of cost at most $\gamma \sum_{i=1}^{p} y_i \le \gamma z_{OPT}$, if γ satisfies that for any infeasible set A and any minimal augmentation B of A

$$\sum_{T_i \in \mathcal{V}(A)} |B \cap T_i| \leq \gamma |\mathcal{V}(A)|,$$

where $\mathcal{V}(A)$ denotes the collection of violated sets output by VIOLATION on input A.

Let us consider again the minimum spanning tree problem. For any set A, $\mathcal{V}(A)$ denotes the set of connected components of A and we know that any minimal augmentation B of A must induce a spanning tree when shrinking all connected components. Therefore, $\sum_{T_i \in \mathcal{V}(A)} |B \cap T_i|$ corresponds to the sum of the degrees of a spanning tree on a graph with $k = |\mathcal{V}(A)|$ supervertices, and is thus equal to $2k-2$, independently of the spanning tree. The upper bound γ on the performance guarantee can thus be set to 2. Theorem 4.2 will be used repeatedly in the next sections to prove the performance guarantee of approximation algorithms for many network design problems.

The reader may be surprised that we did not prove optimality of the spanning tree produced since the algorithm reduces to Kruskal's greedy algorithm. The reason is simply that our linear programming formulation of the minimum spanning tree problem is not strong enough to prove optimality. Instead of increasing the dual variables corresponding to all sets $S \in \mathcal{V}$, we could also view the algorithm as increasing a single dual variable corresponding to the aggregation of the inequalities for every $S \in \mathcal{V}$. The resulting inequality $\sum_{S \in \mathcal{V}} \sum_{e \in \delta(S)} x_e \geq |\mathcal{V}|$ can in fact be strengthened to

$$\sum_{S \in \mathcal{V}} \sum_{e \in \delta(S)} x_e \geq 2|\mathcal{V}| - 2$$

since any connected graph on k vertices has at least $k-1$ edges. The value of the dual solution constructed this way is therefore greater, and with this stronger formulation, it is easy to see that the proof technique developed earlier will prove the optimality of the tree produced. The use of valid inequalities in this primal-dual framework is also considered in Bertsimas and Teo [BT95].

We would like to point out that the bound given in Theorem 4.2 is tight in the following sense. If there exists a set A and a minimal augmentation B of A for which $\sum_{T_i \in \mathcal{V}(A)} |B \cap T_i| = \gamma |\mathcal{V}(A)|$, then the algorithm can return solutions of value equal to γ times the value $\sum_{i=1}^{p} y_i$ of the dual solution constructed by the algorithm. For this, one simply needs to set the cost of all elements of A to 0 and to set appropriately the cost of the elements in $B - A$ so that they would all be added to A at the same time during the execution of the algorithm.

As a final remark, we could also allow the oracle VIOLATION to return sets which do not need to be hit, as we did in the case of the minimum-cost arborescence problem. The performance guarantee is given in the following theorem. Its proof is similar to the proof of Theorem 4.2 and is therefore omitted.

THEOREM 4.3 If the oracle VIOLATION may return sets which do not need to be hit then the performance guarantee of the primal-dual algorithm described in Figure 4.3 is γ, provided that for any infeasible set A and any minimal augmentation B of A

$$\sum_{T_i \in \mathcal{V}(A)} |B \cap T_i| \leq \gamma c,$$

where $\mathcal{V}(A)$ denotes the collection of sets output by VIOLATION, and c denotes the number of sets in $\mathcal{V}(A)$ which need to be hit.

EXERCISE 4.2 Prove the correctness of Dijkstra's algorithm by using Theorem 4.1.

EXERCISE 4.3 Find an instance of the minimum-cost arborescence problem where the use of a non-reverse delete step leads to a non-optimal solution.

EXERCISE 4.4 Consider the minimum spanning tree problem on a complete graph with all edge costs equal to 1. Given a set A of edges, write a restricted primal in the spirit of Section 4.2. Show that the unique optimum solution to its dual is to set the dual variables corresponding to all connected components of (V, A) to 0.5 and all other dual variables to 0.

EXERCISE 4.5 Prove Theorem 4.3.

4.4 A MODEL OF NETWORK DESIGN PROBLEMS

With a primal-dual method for approximation algorithms in place, we show how to apply it to various other network design problems. In this and following sections, we will discuss various problems and prove that the design principles listed above lead to good approximation algorithms for these problems.

Most of the network design problems we will discuss have as input an undirected graph $G = (V, E)$ with nonnegative edge costs c_e, and can be modelled by the following integer program:

$$\begin{array}{llll} & \text{Min} & \sum_{e \in E} c_e x_e & \\ & \text{subject to:} & & \\ (IP) & & \sum_{e \in \delta(S)} x_e \geq f(S) & \emptyset \neq S \subset V \\ & & x_e \in \{0, 1\} & e \in E. \end{array}$$

This integer program is a variation on some of the hitting set problems discussed above, parametrized by the function $f : 2^V \to \mathbf{N}$: here, our ground set is the set of edges E and a feasible solution must contain at least $f(S)$ edges of any cut $\delta(S)$. We will sometimes consider further variations of the problem in which the constraint $x_e \in \{0,1\}$ is replaced by $x_e \in \mathbf{N}$; that is, we are allowed to take any number of copies of an edge e in order to satisfy the constraints. If the function f has range $\{0,1\}$, then the integer program (IP) is a special case of the hitting set problem in which we must hit the sets $\delta(S)$ for which $f(S) = 1$.

We have already seen that (IP) can be used to model two classical network design problems. If we have two vertices s and t, and set $f(S) = 1$ when S contains s but not t, then edge-minimal solutions to (IP) model the undirected $s-t$ shortest path problem. If $f(S) = 1$ for all $\emptyset \neq S \subset V$, then (IP) models the minimum spanning tree problem.

The integer program (IP) can also be used to model many other problems, which we will discuss in subsequent sections. As an example, (IP) can be used to model the *survivable network design problem*, sometimes also called the *generalized Steiner problem*. In this problem we are given nonnegative integers r_{ij} for each pair of vertices i and j, and must find a minimum-cost subset of edges $E' \subset E$ such that there are at least r_{ij} edge-disjoint paths for each i, j pair in the graph (V, E'). This problem can be modelled by (IP) with the function $f(S) = \max_{i \in S, j \notin S} r_{ij}$; a min-cut/max-flow argument shows that it is necessary and sufficient to select $f(S)$ edges from $\delta(S)$ in order for the subgraph to have at least r_{ij} paths between i and j. The survivable network design problem is used to model a problem in the design of fiber-optic telephone networks [GMS94, Sto92]. It finds the minimum-cost network such that nodes i and j will still be connected even if $r_{ij} - 1$ edges of the network fail.

The reader may notice that the two network design problems mentioned above are special cases of the survivable network design problem: the undirected $s-t$ shortest path problem corresponds to the case in which $r_{st} = 1$ and $r_{ij} = 0$ for all other i, j, while the minimum spanning tree problem corresponds to the case $r_{ij} = 1$ for all pairs i, j. Other well-known problems are also special cases. In the *Steiner tree problem*, we are given a set of terminals $T \subseteq V$ and must find a minimum-cost set of edges such that all terminals are connected. This problem corresponds to the case in which $r_{ij} = 1$ if $i, j \in T$ and $r_{ij} = 0$ otherwise. In the *generalized Steiner tree problem*, we are given p sets of terminals $T_1, \ldots, T_p$, where $T_i \subseteq V$. We must find a minimum-cost set of edges such that for each i, all the vertices in T_i are connected. This problem corresponds to the survivable network design problem in which $r_{ij} = 1$ if there exists some k such that $i, j \in T_k$, and $r_{ij} = 0$ otherwise. We will show how the primal-dual method can be applied to these two special cases (and many others) in Section 4.6, and show how the method can be applied to the survivable network design problem in general in Section 4.7.

It is not known how to derive good approximation algorithms for (IP) for any given function f. Nevertheless, the primal-dual method can be used to derive good approximation algorithms for particular classes of functions that model interesting network design problems, such as those given above. In the following sections, we consider various classes of functions f, and prove that the primal-dual method with the design rules of the previous section gives good performance guarantees.

4.4.1 0-1 FUNCTIONS

We will first focus our attention on the case in which the function f has range $\{0,1\}$. We often refer to such functions as 0-1 functions. The shortest path, minimum spanning tree and (generalized) Steiner tree problems all fit in this case, as well as many other problems to be discussed in the coming sections. For functions with range $\{0,1\}$, the integer program (IP) reduces to

$$
\begin{array}{llll}
 & \text{Min} & \sum_{e\in E} c_e x_e & \\
 & \text{subject to:} & & \\
(IP) & & \sum_{e\in\delta(S)} x_e \geq 1 & S : f(S) = 1 \\
 & & x_e \in \{0,1\} & e \in E,
\end{array}
$$

and the dual of its LP relaxation is:

$$
\begin{array}{lll}
\text{Max} & \sum_{S:f(S)=1} y_S & \\
\text{subject to:} & & \\
 & \sum_{S:e\in\delta(S)} y_S \leq c_e & e \in E \\
 & y_S \geq 0 & S : f(S) = 1.
\end{array}
$$

Observe that the edge-minimal solutions of (IP) are forests since one can remove arbitrarily any edge from a cycle without destroying feasibility. In Figure 4.4, we have specialized the algorithm of Figure 4.3 to this case, assuming the oracle VIOLATION returns the minimal violated sets. As already mentioned in the previous section, we will often stretch our terminology to say that a vertex set S is violated, instead of saying that the associated cut $T = \delta(S)$ is violated. Let $\delta_A(S) = \delta(S) \cap A$. Then a set $S \subset V$ is violated when $\delta_A(S) = \emptyset$ and $f(S) = 1$. We can restate Theorem 4.2 as follows.

THEOREM 4.4 The primal-dual algorithm described in Figure 4.4 delivers a feasible solution of cost at most $\gamma \sum_{S:f(S)=1} y_S \leq \gamma z_{OPT}$, if γ satisfies that for any infeasible set A and any minimal augmentation B of A

$$\sum_{S\in\mathcal{V}(A)} |\delta_B(S)| \leq \gamma |\mathcal{V}(A)|,$$

where $\mathcal{V}(A)$ denotes the collection of minimal violated sets.

For general functions f with range $\{0,1\}$, there could be exponentially many sets S for which $f(S) = 1$. As a result, we assume that f is implicitly given through an oracle taking a set S as input and outputting its value $f(S)$. But, for arbitrary 0-1 functions , it might not be easy to check whether an edge set A is feasible, i.e. whether it hits all cuts $\delta(S)$ for which $f(S) = 1$. Also, the minimal violated sets might not have any nice structure as they do for the shortest path

$y \leftarrow 0$
$A \leftarrow \emptyset$
$l \leftarrow 0$
While A is not feasible
  $l \leftarrow l+1$
  $\mathcal{V} \leftarrow \{$minimal violated sets $S\}$
  Increase y_S uniformly for all $S \in \mathcal{V}$ until $\exists e_l \in \delta(T), T \in \mathcal{V} : \sum_{S: e_l \in \delta(S)} y_S = c_{e_l}$
  $A \leftarrow A \cup \{e_l\}$
For $j \leftarrow l$ downto 1
  if $A - \{e_j\}$ is feasible then $A \leftarrow A - \{e_j\}$
Output A (and y)

FIGURE 4.4

Primal-dual algorithm for (IP) with uniform increase rule on minimal violated sets and reverse delete step.

or minimum spanning tree problems. However, consider the class of functions satisfying the *maximality* property:

- [Maximality] If A and B are disjoint, then $f(A \cup B) \leq \max(f(A), f(B))$.

For functions with range $\{0,1\}$, this can also be expressed as:

- [Maximality] If A and B are disjoint, then $f(A) = f(B) = 0$ implies $f(A \cup B) = 0$.

This is equivalent to requiring that if $f(S) = 1$ then for any partition of S at least one member of the partition has an $f(.)$ value equal to 1. For this class of functions, the following lemma shows how to check whether an edge set is feasible and, if it is not, how to find the minimal violated sets.

LEMMA 4.1 Let f be a function with range $\{0,1\}$ satisfying the maximality property. Let A be any edge set. Then

1. A is feasible for f if and only if every connected component C of (V,A) satisfies $f(C) = 0$,
2. the minimal violated sets of A are the connected components C of (V,A) for which $f(C) = 1$.

Proof. Consider a violated set S, i.e. a set S for which $f(S) = 1$ but $\delta_A(S) = \emptyset$. Clearly, S must consist of the union of connected components of (V,A). But, by maximality, one of these components, say C, must satisfy $f(C) = 1$, and is thus a violated set. Thus, only connected components can correspond to minimal violated sets, and A is feasible only if no such component has $f(C) = 1$. ■

In the case of functions satisfying the maximality property, the collection $\mathcal{V}(A)$ of minimal violated sets can thus easily be updated by maintaining the collection $\mathcal{C}(A)$ of connected components of (V,A). This is exploited in Figure 4.5, where we present a more detailed implementation of the primal-dual algorithm of Figure 4.4 in the case of functions satisfying maximality. When implementing the algorithm, there is no need to keep track of the dual variables y_S. Instead, in order to be able to decide which edge to select next, we compute for every vertex $i \in V$ the quantity $d(i)$ defined by $\sum_{S:i\in S} y_S$. Initially, $d(i)$ is 0 (lines 5-6) and it increases by ϵ whenever the dual variable corresponding to the connected component containing i increases by ϵ (line 12). As long as i and j are in different connected components C_p and C_q (respectively), the quantity $(c_e - d(i) - d(j))/(f(C_p) + f(C_q))$ being minimized in line 10 represents the difference between the addition time of edge $e = (i,j)$ and the current time. This explains why the edge with the smallest such value is being added to A. When an edge is added to A, the collection $\mathcal{C}$ of connected components of (V,A) is updated in line 15. We are also maintaining and outputting the value LB of the dual solution, since this allows us to estimate the quality of the solution on any instance. The algorithm can be implemented quite easily. The connected components can be maintained as a union-find structure of vertices. Then all merging take at most $O(n\alpha(n,n))$ time overall, where α is the inverse Ackermann function and n is the number of vertices [Tar75]. To determine which edge to add to A, we can maintain a priority queue of edges, where the key of an edge is its addition time $a(e)$. If two components C_p and C_q merge, we only need to update the keys of the edges incident to $C_p \cup C_q$. Keeping only the smallest edge between two components, one derives a running time of $O(n^2 \log n)$ for all queue operations and this is the overall running time of the algorithm. This is the original implementation as proposed by the authors in [GW95a]. Faster implementations have been proposed by Klein [Kle94] and Gabow, Goemans and Williamson [GGW93].

Even for 0-1 functions obeying maximality, the parameter γ of Theorem 4.4 can be arbitrarily large. For example, consider the problem of finding a tree of minimum cost containing a given vertex s and having at least k vertices. This problem corresponds to the function $f(S) = 1$ if $s \in S$ and $|S| < k$, which satisfies maximality. However, selecting $A = \emptyset$ and B a star rooted at s with k vertices, we observe that $\gamma \geq k-1$. As a result, for this problem, the primal-dual algorithm can output a solution of cost at least $k-1$ times the value of the dual solution produced.

In the following two sections, we apply the primal-dual algorithm to some subclasses of 0-1 functions satisfying maximality. We show that, for these subclasses, the primal-dual algorithm of Figures 4.4 and 4.5 is a 2-approximation algorithm by proving that γ can be set to 2. Before defining these subclasses of functions, we reformulate γ in terms of the average degree of a forest. This explains why a performance guarantee of 2 naturally arises. To prove that $\gamma = 2$, we need to show that, for any infeasible set A and any minimal augmentation B of A, we have $\sum_{S\in\mathcal{V}(A)} |\delta_B(S)| \leq 2|\mathcal{V}(A)|$. For functions satisfying the maximality property, the collection $\mathcal{V}(A)$ of minimal violated sets consists of the connected components of (V,A) whose $f(.)$ value is 1 (Lemma 4.1). Now, construct a graph H formed by taking the graph (V,B) and shrinking the connected components

$A \leftarrow \emptyset$
Comment: Implicitly set $y_S \leftarrow 0$ *for all* $S \subset V$
$LB \leftarrow 0$
$\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
For each $i \in V$
  $d(i) \leftarrow 0$
$l \leftarrow 0$
While $\exists C \in \mathcal{C} : f(C) = 1$
  $l \leftarrow l + 1$
  Find edge $e_l = (i,j)$ with $i \in C_p \in \mathcal{C}$, $j \in C_q \in \mathcal{C}$, $C_p \neq C_q$ that minimizes $\epsilon = \frac{c_{e_l} - d(i) - d(j)}{f(C_p) + f(C_q)}$
  $A \leftarrow A \cup \{e_l\}$
  For all $k \in C_r \in \mathcal{C}$ do $d(k) \leftarrow d(k) + \epsilon \cdot f(C_r)$
  Comment: Implicitly set $y_C \leftarrow y_C + \epsilon \cdot f(C)$ *for all* $C \in \mathcal{C}$.
  $LB \leftarrow LB + \epsilon \sum_{C \in \mathcal{C}} f(C)$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$
For $j \leftarrow l$ downto 1
  If all components C of $A - \{e_j\}$ satisfy $f(C) = 0$ then $A \leftarrow A - \{e_j\}$
Output A and LB

FIGURE 4.5

Primal-dual algorithm for (IP) for functions satisfying the maximality property.

of (V,A) to vertices. For simplicity, we refer to both the graph and its vertex set as H. Because B is an edge-minimal augmentation, there will be a one-to-one correspondence between the edges of $B - A$ and the edges in H, and H is a forest. Each vertex v of H corresponds to a connected component $S_v \subset V$ of (V,A); let d_v denote the degree of v in H, so that $d_v = |\delta_B(S_v)|$. Let W be the set of vertices of H such that for $w \in W$, $f(S_w) = 1$. Then each of these vertices corresponds to a minimal violated set; that is, $\mathcal{V}(A) = \{S_w | w \in W\}$. Thus in order to prove the inequality $\sum_{S \in \mathcal{V}(A)} |\delta_B(S)| \leq 2|\mathcal{V}(A)|$, we simply need to show that

$$\sum_{v \in W} d_v \leq 2|W|. \tag{4.6}$$

In other words, the average degree of the vertices in H corresponding to the violated sets is at most 2. In the next two sections, we show that equation (4.6) holds for two subclasses of functions satisfying the maximality property.

EXERCISE 4.6 Show that the function f corresponding to the generalized Steiner tree problem satisfies the maximality property.

$A \leftarrow$ MINIMUM-SPANNING-TREE
Sort edges of $A = \{e_1, \ldots, e_{n-1}\}$ so that $c_{e_1} \leq \cdots \leq c_{e_{n-1}}$
For $j \leftarrow n-1$ downto 1
  If $A - \{e_j\}$ is feasible then $A \leftarrow A - \{e_j\}$.

FIGURE 4.6

Another 2-approximation algorithm for downwards monotone functions f.

4.5 DOWNWARDS MONOTONE FUNCTIONS

In this section, we consider the network design problems that can be modelled by the integer program (IP) with functions f that are *downwards monotone*. We say that a function is downwards monotone if $f(S) \leq f(T)$ for all $S \supseteq T \neq \emptyset$. Notice that any downwards monotone function satisfies maximality and, as a result, the discussion of the previous section applies. Later in the section, we will prove the following theorem.

THEOREM 4.5 The primal-dual algorithm described in Figure 4.5 gives a 2-approximation algorithm for the integer program (IP) with any downwards monotone function $f : 2^V \rightarrow \{0,1\}$.

In fact, we will also show that applying the reverse delete procedure to the edges of a minimum spanning tree is also a 2-approximation algorithm for the problem; see Figure 4.6 for the algorithm. The advantage of the algorithm in Figure 6 is that its running time is that of computing the minimum spanning tree and sorting its edges, rather than $O(n^2 \log n)$ time. Thus the algorithm takes $O(m + n \log n)$ time in general graphs, and $O(n \log n)$ time in Euclidean graphs.

THEOREM 4.6 The primal-dual algorithm described in Figure 4.6 gives a 2-approximation algorithm for the integer program (IP) with any downwards monotone function $f : 2^V \rightarrow \{0,1\}$.

Before we get to the proofs of these theorems, we consider the kinds of network design problems that can be modelled by (IP) with a downwards monotone function $f : 2^V \rightarrow \{0,1\}$.

4.5.1 THE EDGE-COVERING PROBLEM

The edge-covering problem is that of selecting a minimum-cost set of edges such that each vertex is adjacent to at least one edge. The problem can be solved in polynomial time via a reduction to the minimum-weight perfect matching problem (see Grötschel, Lovász, and Schrijver [GLS88, p. 259]). The problem can be modelled by the downwards monotone function $f(S) = 1$ iff $|S| = 1$. Thus the primal-dual algorithm yields a 2-approximation algorithm for this problem. It is interesting to observe that another primal-dual algorithm for the hitting set problem (or the set cover problem) due to Chvátal [Chv79] (see Chapter 3) gives a performance guarantee of $\frac{3}{2}$ for the edge-covering problem.

4.5.2 LOWER-CAPACITATED PARTITIONING PROBLEMS

In the *lower-capacitated partitioning problems*, we wish to find a minimum-cost set of edges that partitions the vertices into trees, paths, or cycles such that each tree, path, or cycle has at least k vertices, for some parameter k. When $k = 3$, the lower-capacitated cycle partitioning problem is also known as the binary two-matching problem; when $k = 4$, it is also known as the triangle-free binary two-matching problem. The lower-capacitated cycle partitioning problem is *NP*-complete for $k \geq 5$ (Papadimitriou in Cornuéjols and Pulleyblank [CP80] for $k \geq 6$ and Vornberger [Vor79] for $k = 5$), polynomially solvable for $k = 2$ or 3 (Edmonds and Johnson [EJ70]), while its complexity for $k = 4$ is open. Imielińska, Kalantari, and Khachiyan [IKK93] have shown that the lower-capacitated tree partitioning problem is *NP*-complete for $k \geq 4$, even if the edge costs obey the triangle inequality.

The lower-capacitated tree partitioning problem can be modelled by (IP) with the downwards monotone function $f(S) = 1$ if $0 < |S| < k$ and 0 otherwise. If the edge costs obey the triangle inequality, we can also obtain an approximation algorithm for the lower-capacitated path partitioning problem. Obviously the cost of the optimal tree partition is a lower bound on the cost of the optimal lower-capacitated path partition. Given the tree partition produced by our algorithm, we duplicate each edge and find a tour of each component by shortcutting the resulting Eulerian graph on each component; this gives a cycle partition of no more than twice the cost of the original solution. Removing an edge from each cycle gives a path partition; thus we have a 4-approximation algorithm for the lower-capacitated path partitioning problem.

If the edge costs obey the triangle inequality, then we can obtain a 2-approximation algorithm for the lower-capacitated cycle problem. The algorithm constructs a cycle partition as above. To show that the cost of solution is no more than twice optimal, notice that the following linear program is a relaxation of the lower-capacitated cycle problem:

$$
\begin{array}{ll}
\text{Min} & \displaystyle\sum_{e \in E} c_e x_e \\
\text{subject to:} & \\
& \displaystyle\sum_{e \in \delta(S)} x_e \geq 2f(S) \qquad\qquad \emptyset \neq S \subset V \\
& x_e \geq 0.
\end{array}
$$

The dual of this relaxation is

$$\begin{aligned} \text{Max} \quad & 2\sum_{S \subset V} f(S) y_S \\ \text{subject to:} \quad & \\ & \sum_{S: e \in \delta(S)} y_S \leq c_e \qquad \forall e \in E \\ & y_S \geq 0. \end{aligned}$$

The dual solution generated by the primal-dual algorithm for the lower-capacitated tree problem is feasible for this dual, but has twice the objective function value. Let y denote the dual solution given by the primal-dual algorithm, let T denote the set of tree edges produced by the algorithm for the lower-capacitated tree problem, let C denote the set of cycle edges produced by doubling and shortcutting the tree edges, and let Z_C^* denote the cost of the optimal cycle partition. We know $c(T) \leq 2\sum_S f(S) y_S$ and $c(C) \leq 2c(T)$, so that $c(C) \leq 2(2\sum_S f(S) y_S) \leq 2Z_C^*$, proving that the algorithm is a 2-approximation algorithm for the cycle partitioning problem. This illustrates one of the benefits of the primal-dual method: the dual lower bound can be used to prove stronger results.

A paper of the authors [GW95a] provided the first 2-approximation algorithms for these problems. Imielińska, Kalantari, and Khachiyan [IKK93] showed how to select a subset of the edges of a minimum spanning tree to get a 2-approximation algorithm for the tree partitioning problem and a 4-approximation algorithm for the cycle partitioning problem. A subsequent paper of the authors [GW94b] showed how spanning tree edges could be used for any downwards monotone function.

4.5.3 LOCATION-DESIGN AND LOCATION-ROUTING PROBLEMS

The primal-dual method can be used to solve a problem in network design and vehicle routing. Many problems of this type require two levels of decisions. In the first level, the location of special vertices, such as concentrators or switches in the design of communication networks, or depots in the routing of vehicles, needs to be decided. There is typically a set of possible locations and a fixed cost is associated with each of them. Once the locations of the depots are decided, the second level deals with the design or routing per se. These problems are called location-design or location-routing problems (Laporte [Lap88]).

The algorithm can be applied to one of the simplest location-routing problems. In this problem (Laporte et al. [LNP83, Lap88]), we need to select depots among a subset D of vertices of a graph $G = (V, E)$ and cover all vertices in V with a set of cycles, each containing a selected depot. The goal is to minimize the sum of the fixed costs of opening our depots and the sum of the costs of the edges of our cycles. In order to approximate this *NP*-complete problem, we consider an augmented graph $G' = (V \cup D', E')$, which we obtain from G by adding a new copy u' of every vertex $u \in D$ and adding edges of the form (u, u') for all $u \in D$. Edge (u, u') has a cost equal to half the value of the fixed cost of opening a depot at u. Consider the downwards monotone function $f(S) = 1$ if $\emptyset \neq S \subseteq V$ and 0 otherwise. We apply the 2-approximation algorithm for this function f. As in the case of the lower-capacitated cycle partitioning problem, doubling the edges and shortcutting the solution obtained can be shown to result in a 2-approximation algorithm for the original location-design problem.

4.5.4 PROOF OF THEOREMS 4.5 AND 4.6

We now turn to the proof of Theorems 4.5 and 4.6.

Proof of Theorem 4.5. Using the arguments developed in Section 4.4.1, we simply need to show that, for downwards monotone functions, equation (4.6) holds.

Recall that we construct a graph H by taking the graph (V, B) and shrinking the connected components of (V, A) to vertices. Each vertex v of H corresponds to a connected component S_v of (V, A), and has degree d_v. The set W is the set of vertices $\{v \in H : f(S_v) = 1\}$. We first claim that each connected component of H has at most one vertex v such that $f(S_v) = 0$. Suppose this is false, and some connected component of H has two vertices, v and w, such that $f(S_v) = f(S_w) = 0$. Let e be an edge of B corresponding to an edge on the path between v and w in H. By minimality of B, $B - \{e\}$ is not feasible. Thus, there is a set $S \subset V$ such that $e \in \delta(S)$ and $f(S) = 1$, but $(B - \{e\}) \cap \delta(S) = \emptyset$. The removal of e must split a connected component of H. In order that $e \in \delta(S)$ and $(B - \{e\}) \cap \delta(S) = \emptyset$, it must be the case that S contains the vertices of one of the two parts of this component. Thus either $S_v \subseteq S$ or $S_w \subseteq S$. By the downwards monotonicity of f, $f(S) = 0$, a contradiction.

Let c be the number of components of H. Then

$$\sum_{v \in W} d_v \leq \sum_{v \in H} d_v = 2(|H| - c) \leq 2|W|,$$

as desired, since H is a forest, and $|H - W| \leq c$ by the claim above. ∎

Proof of Theorem 4.6. If we increase the dual variables on all connected components $\mathcal{C}(A)$ of (V, A), rather than the minimal violated sets $\mathcal{V}(A)$, then, as was argued in Section 4.3, the first part of the algorithm reduces to Kruskal's algorithm for the minimum spanning tree. We can therefore use Theorem 4.3 to prove a performance guarantee of 2 for the algorithm of Figure 4.6 if we can show that

$$\sum_{S \in \mathcal{C}(A)} |\delta_B(S)| \leq 2|\{S \in \mathcal{V}(A)\}|,$$

where B is any minimal augmentation of A, $\mathcal{C}(A)$ is the set of connected components of (V, A) and $\mathcal{V}(A) = \{C \in \mathcal{C}(A) : f(C) = 1\}$ is the set of minimal violated sets. Using the notation developed in the previous section, this reduces to $\sum_{v \in H} d_v \leq 2|W|$, which was proved above. This proves that the algorithm of

Figure 4.6 is also a 2-approximation algorithm. ■

Further variations on the algorithm also yield 2-approximation algorithms. Imielińska et al. [IKK93] give a 2-approximation algorithm for the lower capacitated tree problem that selects appropriate edges of a minimum spanning tree in order of increasing cost, rather than deleting edges in order of decreasing cost. The authors have generalized this algorithm to a 2-approximation algorithm for downwards monotone functions $f : 2^V \to \mathbf{N}$ for the integer program (IP) with the constraint $x_e \in \mathbf{N}$ [GW94b].

EXERCISE 4.7 Show that the performance guarantee in the statement of Theorem 4.5 can be improved to $2 - 1/l$ where $l = |\{v : f(\{v\}) = 1\}|$.

EXERCISE 4.8 Give a very simple proof of the fact that the algorithm of Figure 4.6 is a 2-approximation algorithm for the edge covering problem.

4.6 0-1 PROPER FUNCTIONS

In this section, we consider the network design problems which can be modelled by the integer program (IP) with a proper function f with range $\{0,1\}$. A function $f : 2^V \to \mathbf{N}$ is proper if

- $f(V) = 0$,
- f satisfies the maximality property and
- f is symmetric, i.e. $f(S) = f(V - S)$ for all $S \subseteq V$.

Under symmetry, it can be shown that, for 0-1 functions, the maximality property is equivalent to requiring that if $f(S) = f(A) = 0$ for $A \subseteq S$ then $f(S - A) = 0$. We will refer to this property as *complementarity*. The class of 0-1 proper functions is incomparable to the class of downwards monotone functions; neither class is contained in the other. The class of network design problems which can be formulated by proper functions is particularly rich. It encompasses very diverse problems such as the shortest path problem, the minimum-weight T-join problem, the generalized Steiner tree problem, or the point-to-point connection problem. Later in this section, we elaborate on some of these applications. The work described in this section appeared in [GW95a], the first paper of the authors on the use of the primal-dual method for network design problems.

As with downwards monotone functions, the primal-dual algorithm described in Figure 4.5 is a 2-approximation algorithm.

THEOREM 4.7 The primal-dual algorithm described in Figure 4.5 gives a 2-approximation algorithm for the integer program (IP) with any 0-1 proper func-

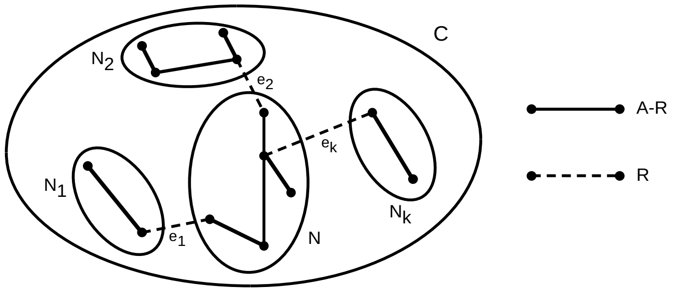

FIGURE 4.7

Illustration for the proof of Lemma 4.2.

tion $f : 2^V \rightarrow \{0, 1\}$.

The proof of this theorem is given below. With regard to the (reverse) delete step, proper functions behave very much as in case of the shortest path problem. No matter how the delete step is implemented, the same subgraph is output since, as is shown in the next lemma, there is a unique minimally feasible subset of any feasible solution.

LEMMA 4.2 Let f be any 0-1 proper function and let A be any feasible solution. Let $R = \{e : A - \{e\} \text{ is feasible}\}$. Then $A - R$ is feasible.

Notice that R represents the set of all edges that can possibly be removed without losing feasibility. The lemma shows that *all* these edges can be simultaneously removed without losing feasibility. For 0-1 proper functions, we can thus replace the reverse delete step (lines 16-17) in Figure 4.5 by the following command.

16 $\quad A \leftarrow \{e \in A : \text{For some connected component } N \text{ of } (V, A - \{e\}), f(N) = 1\}$.

Proof of Lemma 4.2. Let N be any connected component of $(V, A - R)$. We first claim that $f(N) = 0$. Clearly, $N \subseteq C$ for some connected component C of (V, A). Now let $e_1, \ldots, e_k$ be the edges of A such that $e_i \in \delta(N)$ (possibly $k = 0$). Let N_i and $C - N_i$ be the two components created by removing e_i from the edges of component C, with $N \subseteq C - N_i$ (see Figure 4.7). Since $e_i \in R$, it must be the case that $f(N_i) = 0$. Note also that the sets $N, N_1, \ldots, N_k$ form a partition of C. Therefore, by maximality, $f(C - N) = f(\cup_{i=1}^k N_i) = 0$. Since $f(C) = 0$, complementarity now implies that $f(N) = 0$. Since every connected component of $(V, A - R)$ has $f(.)$ value equal to 0, Lemma 4.1 implies that $A - R$ is feasible. ∎

Proof of Theorem 4.7. As discussed at the end of Section 4.4.1, the proof of the theorem can be reduced to the proof of inequality (4.6), as was the case for downwards monotone functions.

In order to prove (4.6) for 0-1 proper functions, we first claim that no leaf v of H satisfies $f(S_v)=0$. Suppose otherwise. Let e be the edge incident to v and let C be the connected component of (V,B) that contains S_v. By feasibility of B, $f(C)=0$. The assumption that $f(S_v)=0$ together with complementarity now implies that $f(C-S_v)=0$. But by minimality of B, $B-\{e\}$ is not feasible which implies that either S_v or $C-S_v$ has an $f(.)$ value equal to 1, which is a contradiction. Thus every leaf v of H belongs to W.

Showing that the average degree over the vertices in W is at most 2 is now easy. First discard all isolated vertices from H (they do not contribute in any way). Now,

$$\sum_{v\in W} d_v = \sum_{v\in H} d_v - \sum_{v\notin W} d_v \le (2|H|-2)-2(|H|-|W|)=2|W|-2,$$

since H is a forest of at most $|H|-1$ edges, and since all vertices not in W have degree at least two. This proves inequality (4.6), and completes the proof of the theorem. ■

Since the inequality proved is a bit stronger than what was claimed, the proof can be refined to show that the performance guarantee is in fact equal to $2-\frac{2}{l}$, where $l=|\{v: f(\{v\})=1\}|$. Also, observe the similarities and differences between the proofs of the performance guarantee for downwards monotone functions and proper functions. In both cases, the argument hinges on the fact that the average degree of a forest remains at most 2 if we discard certain vertices. In the former, we discard at most one vertex per component and in the latter we discard only inner vertices (i.e. non-leaves). The result would still hold if we discard any number of inner vertices but at most one leaf (or even two leaves) per component. By using the same arguments as in the proofs of Theorems 4.5 and 4.7, this for example shows that the algorithm of Figure 4.5 is still a 2-approximation algorithm for the class of functions satisfying maximality and the following condition: there do not exist a set S and two disjoint subsets A,B of S such that $f(S)=f(A)=f(B)=0$ and $f(S-A)=f(S-B)=1$. This class of functions contains both the downwards monotone functions and the proper functions, but we are not aware of any interesting application of this generalization not covered by the previous two classes.

We now discuss network design problems which can be modelled as integer programs (IP) with a proper function f.

4.6.1 THE GENERALIZED STEINER TREE PROBLEM

The generalized Steiner tree problem is the problem of finding a minimum-cost forest that connects all vertices in T_i for $i=1,\ldots,p$. The generalized Steiner tree problem corresponds to the proper function f with $f(S)=1$ if there exists $i\in\{1,\ldots,p\}$ with $\emptyset\neq S\cap T_i\neq T_i$ and 0 otherwise. In this case, the primal-dual algorithm we have presented simulates an algorithm of Agrawal, Klein, and Ravi [AKR95]. Their algorithm was the first approximation algorithm for this problem and has motivated much of the authors' research in this area.

When $p = 1$, the problem reduces to the classical Steiner tree problem. For a long time, the best approximation algorithm for this problem had a performance guarantee of $(2 - \frac{2}{k})$ (for a survey, see Winter [Win87]) but, recently, Zelikovsky [Zel93] obtained an $\frac{11}{6}$−approximation algorithm. Further improvements have been obtained; we refer the reader to Chapter 8.

4.6.2 THE T-JOIN PROBLEM

Given an even subset T of vertices, the T-join problem consists of finding a minimum-cost set of edges that has odd degree at vertices in T and even degree at vertices not in T. Edmonds and Johnson [EJ73] have shown that the T-join problem can be solved in polynomial time. The problem corresponds to the proper function f with $f(S) = 1$ if $|S \cap T|$ is odd and 0 otherwise. When $|T| = 2$, the T-join problem reduces to the shortest path problem. The primal-dual algorithm for 0-1 proper functions in this case reduces to a variant of Dijkstra's algorithm that uses bidirectional search (Nicholson [Nic66]).

4.6.3 THE MINIMUM-WEIGHT PERFECT MATCHING PROBLEM

The minimum-weight perfect matching problem is the problem of finding a minimum-cost set of non-adjacent edges that cover all vertices. This problem can be solved in polynomial time by a primal-dual algorithm discovered by Edmonds [Edm65]. The fastest strongly polynomial time implementation of Edmonds' algorithm is due to Gabow [Gab90]. Its running time is $O(n(m+n\log n))$. For integral costs bounded by C, the best weakly polynomial algorithm runs in $O(m\sqrt{n\alpha(m,n)\log n}\,\log nC)$ time and is due to Gabow and Tarjan [GT91].

These algorithms are fairly complicated and, in fact, time-consuming for large instances that arise in practice. This motivated the search for faster approximation algorithms. Reingold and Tarjan [RT81] have shown that the greedy procedure has a tight performance guarantee of $\frac{4}{3}n^{0.585}$ for general nonnegative cost functions. Supowit, Plaisted and Reingold [SPR80] and Plaisted [Pla84] have proposed an $O(\min(n^2\log n, m\log^2 n))$ time approximation algorithm for instances that obey the triangle inequality. Their algorithm has a tight performance guarantee of $2\log_3(1.5n)$.

As shown by Gabow and Tarjan [GT91], an exact scaling algorithm for the maximum-weight matching problem can be used to obtain an $(1 + 1/n^a)$-approximation algorithm ($a \geq 0$) for the minimum-weight perfect matching problem. Moreover, if the original exact algorithm runs in $O(f(m,n)\log C)$ time, the resulting approximation algorithm runs in $O(m\sqrt{n\log n} + (1+a)f(m,n)\log n)$. Vaidya [Vai91] obtains a $(3 + 2\epsilon)$-approximation algorithm for minimum-weight perfect matching instances satisfying the triangle inequality. His algorithm runs in $O(n^2\log^{2.5} n\log(1/\epsilon))$ time.

The primal-dual algorithm for problems modelled with a proper function can be used to approximate the minimum-weight perfect matching problem when the edge costs obey the triangle inequality. We use the algorithm with the proper function $f(S)$ being the parity of $|S|$, i.e. $f(S) = 1$ if $|S|$ is odd and 0 if $|S|$ is even. This function is the same as the one used for the V-join problem. The algorithm returns a forest whose components have even size. More precisely, the forest is a V-join, and each vertex has odd degree: if a vertex has even degree, then, by a parity argument, some edge adjacent to the vertex could have been deleted so that the resulting components have even size. Thus this edge would have been deleted in the delete step of the algorithm. The forest can be transformed into a perfect matching with no increase of cost by repeatedly taking two edges (u, v) and (v, w) from a vertex v of degree three or more and replacing these edges with the edge (u, w). This procedure maintains the property that the vertices have odd degree. This algorithm has a performance guarantee of $2 - \frac{2}{n}$.

Often the vertices of matching instances are given as points in the plane; the cost of an edge is then the Euclidean distance between its endpoints. Jünger and Pulleyblank [JP91] have observed that the dual variables of matching problems in this case correspond nicely to "moats" around sets of points. That is, a dual variable y_S corresponds to a region of the plane of width y_S surrounding the vertices of S. The dual program for these instances attempts to find a packing of non-overlapping moats that maximizes the sum of the width of the moats around odd-sized sets of vertices. The algorithm of Figure 4.5 applied to Euclidean matching instances can thus be interpreted as growing odd moats at the same speed until two moats collide, therefore adding the corresponding edge, and repeating the process until all components have even size. The reverse delete step then removes unnecessary edges. See Figure 4.8 for an example.

The notion of moats is not particular to matching problems: one can also consider moat packings for Euclidean instances of other problems modelled by (IP). The moats for a feasible dual solution y can be drawn in the plane whenever the non-zero dual variables y_S form a laminar family (any two sets in the family are either disjoint or one is contained in the other). One can show that whenever f is a 0-1 proper function, there exists an optimal dual solution y such that $\{S : y_S > 0\}$ is laminar (this even holds when f is an uncrossable function; see Section 4.7 for a definition). This is also clearly true for the dual solutions constructed by our primal-dual algorithms.

4.6.4 POINT-TO-POINT CONNECTION PROBLEMS

In the point-to-point connection problem, we are given a set $C = \{c_1, \ldots, c_p\}$ of sources and a set $D = \{d_1, \ldots, d_p\}$ of destinations in a graph $G = (V, E)$ and we need to find a minimum-cost set F of edges such that each source-destination pair is connected in F [LMSL92]. This problem arises in the context of circuit switching and VLSI design. The fixed destination case in which c_i is required to be connected to d_i is a special case of the generalized Steiner tree problem where $T_i = \{c_i, d_i\}$. In the non-fixed destination case, each component of the forest F is only required to contain the same number of sources and destinations. This

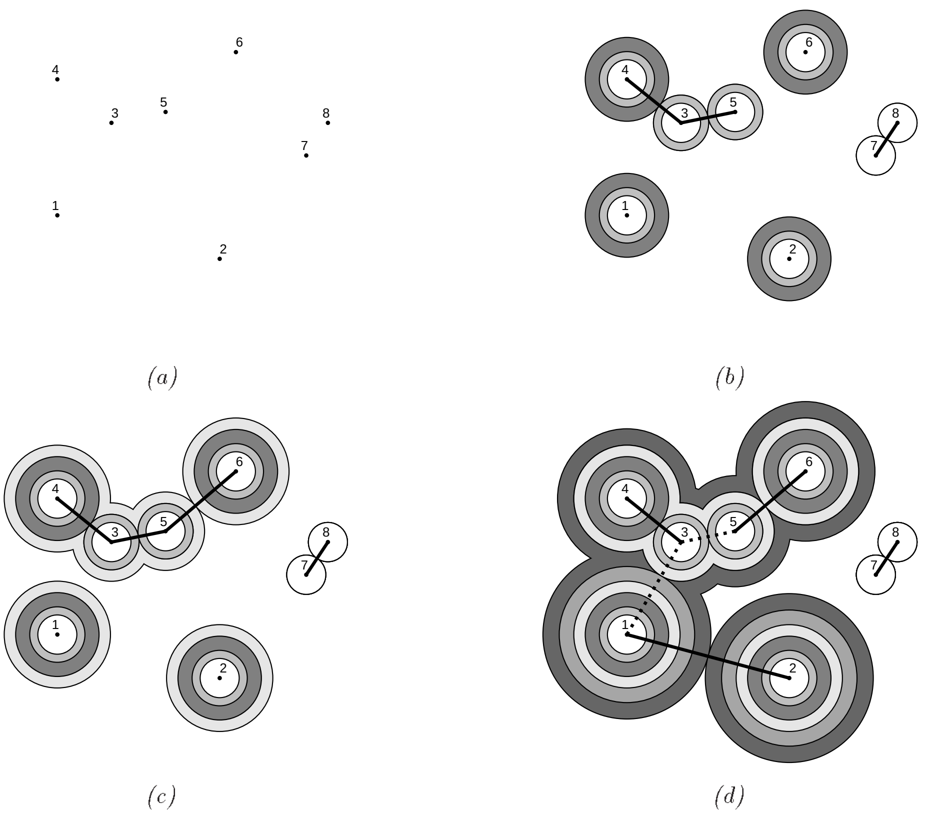

FIGURE 4.8

(a) A Euclidean matching instance. (b) An intermediate stage of the primal-dual algorithm of Figure 4.5 with a partial moat packing. The odd connected components are $\{1\}$, $\{2\}$, $\{3,4,5\}$ and $\{6\}$. (c) When growing odd moats uniformly, the edge $(5,6)$ becomes tight and is added to A. (d) The final solution. The edges removed in the reverse delete step are dashed, the others belong to the matching (or the V-join) output by the algorithm. Observe that every moat is intersected by exactly one edge of the matching, implying that the matching and the dual solution (or moat packing) are both optimal.

problem is NP-complete [LMSL92]. The non-fixed case can be modelled by the proper function f with $f(S) = 1$ if $|S \cap C| \neq |S \cap D|$ and 0 otherwise.

4.6.5 EXACT PARTITIONING PROBLEMS

In the exact tree (cycle, path) partitioning problem, for a given k we must find a minimum-cost collection of vertex-disjoint trees (cycles, paths) of size k that cover all vertices. These problems generalize the minimum-weight perfect matching problem (in which each component must have size exactly 2), the traveling salesman problem, the Hamiltonian path problem and the minimum-cost spanning tree problem.

We can approximate the exact tree, cycle and path partitioning problems for instances that satisfy the triangle inequality. For this purpose, we consider the proper function $f(S) = 1$ if $S \not\equiv 0 \,(\mathrm{mod}\ k)$ and 0 otherwise. Our algorithm finds a forest in which each component has a number of vertices which is a multiple of k, and such that the cost of the forest is within $2 - \frac{2}{n}$ of the optimal such forest. Obviously the cost of the optimal such forest is a lower bound on the optimal exact tree and path partitions. Given the forest, we duplicate each edge and find a tour of each component by shortcutting the resulting Eulerian graph on each component. If we remove every kth edge of the tour, starting at some edge, the tour is partitioned into paths of k nodes each. Some choice of edges to be removed (i.e., some choice of starting edge) accounts for at least $\frac{1}{k}$ of the cost of the tour, and so we remove these edges. Thus this algorithm is a $\left(4(1 - \frac{1}{k})(1 - \frac{1}{n})\right)$-approximation algorithm for the exact tree and path partitioning problems.

To produce a solution for the exact cycle partitioning problem, we add the edge joining the endpoints of each path; given the triangle inequality, this at most doubles the cost of the solution produced. However, the resulting algorithm is still a $\left(4(1 - \frac{1}{k})(1 - \frac{1}{n})\right)$-approximation algorithm for the cycle problem by the same argument as was used in Section 4.5.2.

The proper functions corresponding to the non-fixed point-to-point connection problem, the T-join problem and the exact partitioning problems are all of the form $f(S) = 1$ if $\sum_{i \in S} a_i \not\equiv 0 \,(\mathrm{mod}\ p)$ and 0 otherwise, for some integers a_i, $i \in V$, and some integer p.

EXERCISE 4.9 Prove that, for symmetric functions f, the maximality property is equivalent to complementarity.

EXERCISE 4.10 Consider a (non necessarily symmetric) function f satisfying maximality and complementarity. Consider the symmetrization of f defined by $f_{sym}(S) = \max(f(S), f(V - S))$. Observe that the integer programs corresponding to f and f_{sym} are equivalent. Show that f_{sym} is a proper function.

EXERCISE 4.11 Prove that the performance guarantee of Theorem 4.7 is in fact $2 - 2/l$, where $l = |\{v : f(\{v\}) = 1\}|$. What is the resulting performance guarantee for the shortest path problem (i.e. for $f(S) = 1$ iff $|S \cap \{s,t\}| = 1$)?

EXERCISE 4.12 Prove that the algorithm of Figure 4.5 is a 2-approximation algorithm for the class of functions f satisfying maximality and the property that there do not exist a set S and two disjoint subsets A, B of S such that $f(S) = f(A) = f(B) = 0$ and $f(S-A) = f(S-B) = 1$.

4.7 GENERAL PROPER FUNCTIONS

We now turn from 0-1 proper functions to the case of general proper functions in which the function f can range over the nonnegative integers. In the previous two sections we have discussed special cases of the hitting set problem in which we considered the integer program (IP) with a 0-1 function f. We now wish to consider the case in which we must hit a set $\delta(S)$ at least $f(S)$ times. We will give a $2\mathcal{H}(f_{max})$-approximation algorithm for any proper function f, where $f_{max} = \max_S f(S)$ and $\mathcal{H}(k) = 1 + \frac{1}{2} + \cdots + \frac{1}{k} \approx \ln k$. The results presented in this section were initially given in [WGMV95, GGW93, GGP$^+$94].

The main application of an algorithm for general proper functions is the survivable network design problem, as discussed in Section 4.4. As we previously observed, this problem can be modelled by (IP) with the function $f(S) = \max_{i\in S, j\notin S} r_{ij}$. It is not hard to show that this function is proper: first, it is obviously symmetric. To see that it obeys maximality, let A and B be disjoint sets, and pick $i \in A \cup B$, $j \notin A \cup B$ that attain the maximum $\max_{i\in A\cup B, j\notin A\cup B} r_{ij} = f(A\cup B)$. If $i \in A$, then $f(A) \geq f(A \cup B)$, else $f(B) \geq f(A \cup B)$, ensuring that $f(A\cup B) \leq \max(f(A), f(B))$.

In order to apply the primal-dual method to this class of problems, we reduce the overall problem to a sequence of hitting set problems, and apply the primal-dual approximation algorithm to each subproblem. Thus we build a solution to the original problem in a series of *phases*. We start with an empty set of edges $F_0 = \emptyset$. In each phase p, we consider a hitting set problem with the ground set of elements $E_p = E - F_{p-1}$. Let $\Delta_p(S) = f(S) - |\delta_{F_{p-1}}(S)|$ be the *deficiency* of the set S; that is, the number of edges we must still choose from $\delta(S)$ since a feasible solution to the overall problem must contain $f(S)$ edges, but our current solution F_{p-1} contains only $|\delta_{F_{p-1}}(S)|$ edges. Let $\Delta_{p,\max}$ denote the maximum deficiency, $\Delta_{p,\max} = \max_S \Delta_p(S)$. In the hitting set problem for phase p, the sets to be hit are defined as the sets $\delta(S)$ for which $\Delta_p(S) = \Delta_{p,\max}$. If A is a feasible solution to this problem, then the maximum deficiency of $A \cup F_{p-1}$ can be no greater than $\Delta_{p,\max} - 1$. Thus we apply the algorithm of Figure 4.4 to this hitting set problem; given the resulting set of edges A, we set F_p to $A \cup F_{p-1}$ and we proceed to phase $p+1$. Since the maximum deficiency in the first phase is f_{max}, where $f_{max} = \max_S f(S)$, at most f_{max} phases are necessary before we have a

```
F_0 ← ∅
for p ← 1 to f_max
  Comment: Phase p.
  Δ_p(S) ← f(S) − |δ_{F_{p−1}}(S)| for all S ⊂ V
  h_p(S) ← { 1  if Δ_p(S) = max_S Δ_p(S) = f_max − p + 1
           { 0  otherwise
  E_p ← E − F_{p−1}
  Let A be the edge set returned by the algorithm of Figure 4.4 applied to the
  hitting set problem associated with the the graph (V, E_p) and the function h_p
  F_p ← F_{p−1} ∪ A
Output F_{f_max}
```

FIGURE 4.9

Primal-dual algorithm for proper functions f.

feasible solution to the overall problem. It is possible to show that given this scheme all f_{max} phases are necessary, and the maximum deficiency in phase p is exactly $f_{max} - p + 1$. The algorithm for general proper functions is formalized in Figure 4.9. The idea of augmenting a graph in phases has been previously used in many graph algorithms; in terms of primal-dual approximation algorithms it was first used by Klein and Ravi [KR93] and Saran et al. [SVY92].

The central difficulties of obtaining an algorithm for general proper functions become applying the algorithm of Figure 4.4 to the hitting set problems generated by the algorithm above, and showing that a good performance guarantee for the solution of each hitting set problem leads to the performance guarantee of $2\mathcal{H}(f_{max})$ for the overall problem. We postpone the second difficulty for a moment in order to deal with the first. Given a hitting set problem from phase p, let $h_p(S) = 1$ if we must hit $\delta(S)$ and $h_p(S) = 0$ otherwise. Unfortunately, it is easy to come up with examples such that h_p does not obey maximality, and so we cannot straightforwardly apply the discussion of the previous two sections. Fortunately, the functions h_p arising from the hitting set problems of the phases have a particularly nice structure. We will prove below that the functions belong to the class of *uncrossable* functions. A function $h : 2^V \to \{0,1\}$ is uncrossable if

- $h(V) = 0$; and
- if $h(A) = h(B) = 1$ for any sets of vertices A, B, then either $h(A \cup B) = h(A \cap B) = 1$ or $h(A - B) = h(B - A) = 1$.

The class of uncrossable functions contains all functions satisfying the maximality property. We will show below that the minimal violated sets of uncrossable functions are disjoint.

LEMMA 4.3 Let f be a proper function, $F \subseteq E$, $\Delta(S) = f(S) - |\delta_F(S)|$, and $\Delta_{\max} = \max_S \Delta(S)$. Then the function $h(S) = 1$ if $\Delta(S) = \Delta_{\max}$ and $h(S) = 0$ otherwise is uncrossable.

Proof. Since $f(V) = |\delta_F(V)| = 0$, we have $h(V) = 0$. By the maximality of f, we have the following four inequalities for any two sets X and Y:

- $\max\{f(X-Y), f(X \cap Y)\} \geq f(X)$.
- $\max\{f(Y-X), f(X \cup Y)\} \geq f(X)$.
- $\max\{f(Y-X), f(X \cap Y)\} \geq f(Y)$.
- $\max\{f(X-Y), f(X \cup Y)\} \geq f(Y)$.

Summing the two inequalities involving the minimum of $f(X-Y)$, $f(Y-X)$, $f(X \cup Y)$, and $f(X \cap Y)$ shows that $f(X)+f(Y) \leq \max\{f(X-Y)+f(Y-X), f(X \cap Y)+f(X \cup Y)\}$. To prove the lemma, we use the well-known fact that $\delta_F(S)$ is submodular; that is, for any sets of vertices X and Y

$$|\delta_F(X)| + |\delta_F(Y)| \geq |\delta_F(X \cap Y)| + |\delta_F(X \cup Y)|,$$

and

$$|\delta_F(X)| + |\delta_F(Y)| \geq |\delta_F(X - Y)| + |\delta_F(Y - X)|.$$

Then we can see that $\Delta(X)+\Delta(Y) \leq \max\{\Delta(X-Y)+\Delta(Y-X), \Delta(X \cap Y)+\Delta(X \cup Y)\}$. From this inequality it is easy to see that h is uncrossable. ■

LEMMA 4.4 Let h be any uncrossable function. Then the minimal violated sets of any subset A are disjoint.

Proof. Note that a set S is violated if $h(S)=1$ and $\delta_A(S)=\emptyset$. Suppose there exist two minimal violated sets X and Y that are not disjoint. Then we know that $h(X)=h(Y)=1$ and $\delta_A(X)=\delta_A(Y)=\emptyset$. Since the sets are minimal, $Y-X \neq \emptyset$ and $X-Y \neq \emptyset$; since they are not disjoint, $X \cap Y \neq \emptyset$. By the definition of uncrossable functions, either $h(X-Y)=h(Y-X)=1$ or $h(X \cap Y)=h(X \cup Y)=1$. Suppose the latter is true. Then by submodularity, $\delta_A(X \cup Y)=\delta_A(X \cap Y)=\emptyset$, implying that $X \cup Y$ and $X \cap Y$ are also violated, and contradicting the minimality of X and Y. The other case is similar. ■

Despite Lemma 4.4, it is still difficult to find the minimal violated sets (or even just check feasibility, see Exercise 4.17) for an arbitrary uncrossable function if the function is given only as an oracle. Consider a function taking the value 1 for only one arbitrary set S; this function is uncrossable but the oracle would not allow us to find S without testing all sets in the worst case. Nevertheless, for the uncrossable functions generated by our algorithm, it is possible to find these minimal violated sets in polynomial time by using minimum cut computations. We refer the reader to Williamson et al. [WGMV95], Gabow et al. [GGW93], Williamson [Wil93] and to Exercise 4.19 for details.

Williamson et al. [WGMV95] have shown that the algorithm of Figure 4.4 is a 2-approximation algorithm for any uncrossable function; it runs in polynomial time given a polynomial-time algorithm to compute h and the minimal violated sets.

THEOREM 4.8 The primal-dual algorithm of Figure 4.4 is a 2-approximation algorithm for any uncrossable function f.

This theorem can again be proved using the proof technique developed in Section 4.3 (see Theorem 4.4). However, the proof that γ can be set to 2 is more complicated than in the previous cases, and is therefore omitted.

We must now tackle the second difficulty and show that the performance guarantee of 2 for the uncrossable functions arising in each phase leads to a performance guarantee of $2\mathcal{H}(f_{max})$ for the overall algorithm of Figure 4.9.

THEOREM 4.9 The primal-dual algorithm described in Figure 4.9 gives a $2\mathcal{H}(f_{max})$-approximation algorithm for the integer program (IP) with any proper function f, where $\mathcal{H}(k) = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k}$.

In order to prove Theorem 4.9 from Theorem 4.8, we first show that the dual solution y constructed in phase p by the algorithm can be mapped to a feasible solution to the dual of the LP relaxation of (IP). This dual is:

$$
\begin{array}{llll}
 & \text{Max} & \displaystyle\sum_{S \subset V} f(S) y_S - \sum_{e \in E} z_e & \\
 & \text{subject to:} & & \\
(D) & & \displaystyle\sum_{S : e \in \delta(S)} y_S \le c_e + z_e & e \in E, \\
 & & y_S \ge 0 & \emptyset \ne S \subset V, \\
 & & z_e \ge 0 & e \in E.
\end{array}
$$

Given the dual variables y constructed by the algorithm in phase p, define $z_e = \sum_{S : e \in \delta(S)} y_S$ for all $e \in F_{p-1}$, and $z_e = 0$ otherwise. It is easy to verify that (y, z) is a feasible solution for (D). We now provide a proof of Theorem 4.9.

Proof. Observe that

$$\sum_{e \in E} z_e = \sum_{e \in F_{p-1}} \sum_{S : e \in \delta(S)} y_S = \sum_S |\delta_{F_{p-1}}(S)| y_S.$$

Comparing the value of the dual solution produced by the algorithm in phase p to the optimum value Z_D^* of the dual (D), we deduce

$$
\begin{aligned}
Z_D^* &\ge \sum_S f(S) y_S - \sum_{e \in E} z_e \\
&= \sum_S (f(S) - |\delta_{F_{p-1}}(S)|) y_S \\
&= (f_{max} - p + 1) \sum_S y_S,
\end{aligned}
$$

where we have used the fact that in phase p the dual variable $y_S > 0$ only if the deficiency of S $(f(S) - |\delta_{F_{p-1}}(S)|)$ is $f_{max} - p + 1$. Using the proof of the performance guarantee for uncrossable functions and summing over all phases, we obtain that

$$\sum_{e \in F_{f_{max}}} c_e \le 2 \sum_{p=1}^{f_{max}} \frac{1}{f_{max} - p + 1} Z_D^* = 2\mathcal{H}(f_{max})\, Z_D^*,$$

proving the desired result. ■

Notice that the fact that the algorithm for uncrossable functions constructs a dual feasible solution is crucial for the proof of the above theorem. An improved approximation algorithm for uncrossable functions would be useless for proving any performance guarantee for proper functions if it only compared the solution produced to the optimum value rather than to a dual feasible solution.

It is an interesting open question whether the primal-dual method can be used to design approximation algorithms for general proper functions or the survivable network design problem with a performance guarantee *independent* of f_{max}.

EXERCISE 4.13 Prove that, for a proper function f, $f_{max} = \max\{f(\{v\}) : v \in V\}$.

EXERCISE 4.14 Show that any 0-1 function satisfying the maximality property is uncrossable.

EXERCISE 4.15 Dijkstra's algorithm corresponds to the algorithm of Figure 4.4 with $f(S) = 1$ if $s \in S$ and $t \notin S$, and $f(S) = 0$ otherwise. Show that this function does not satisfy the maximality property but is uncrossable.

EXERCISE 4.16 Show that Lemma 4.3 also holds for the more general class of *skew supermodular* functions. A function f is skew supermodular if $f(V) = 0$ and $f(A) + f(B) \le \max(f(A-B) + f(B-A), f(A \cup B) + f(A \cap B))$ for all sets A and B.

EXERCISE 4.17 Let h be an uncrossable function and assume we have an oracle for deciding the feasibility of a set A of edges. First prove that if $S \subset V$ is a *maximal* set such that $A \cup \{(i,j) : i,j \in S\}$ is not feasible then $V - S$ is a minimal violated set for A. Then deduce that the set of minimal violated sets can be obtained by less than $|V|^2$ calls to the feasibility oracle. Given a general proper function f, consider the problem of checking the feasibility of a set F of edges. Prove that F is feasible if and only if the $|V| - 1$ cuts induced by the Gomory-Hu cut equivalent tree [GH61] of F have the required number of edges [GGW93].

EXERCISE 4.18 Consider the uncrossable function h_p defined in phase p of the algorithm of Figure 4.9. Show that A is feasible for h_p if and only if $A \cup F_{p-1}$ is feasible for the function $g_p(S) = \max(f(S) - f_{max} + p, 0)$. Moreover, show that this function g_p is proper.

EXERCISE 4.19 Using Exercises 4.17–4.18, show how to find the minimal violated sets for the uncrossable function h_p of the algorithm of Figure 4.9

in polynomial time. More efficient solutions to this exercise can be found in [WGMV95, GGW93, Wil93].

EXERCISE 4.20 Prove Theorem 4.8. Can you improve the performance guarantee to $2-2/l$ where l denotes the maximum number of disjoint sets $C_1,\ldots,C_l$ such that $f(C_i)=1$ for all i?

EXTENSIONS 4.8

Up to this point, we have concentrated on showing how the primal-dual method can be applied to various network design problems that can be modelled by the integer program (IP) with different classes of functions f. In this section, we show that the method can be applied to other problems as well.

4.8.1 MINIMUM MULTICUT IN TREES

The primal-dual method can be applied to problems that are not network design problems. For example, Garg, Vazirani, and Yannakakis [GVY93a] have given a primal-dual 2-approximation algorithm for the problem of finding a *minimum multicut* in a tree. In the general minimum multicut problem, we are given an undirected graph $G=(V,E)$ with nonnegative capacities u_e on the edges, and pairs $s_i,t_i \in V$, for $i=1,\ldots,k$. We must remove a minimum-capacity subset of edges E' so that no s_i,t_i pair is in the same connected component of $(V,E-E')$. In the minimum multicut problem in trees, the set of edges E is a tree on V. In this case, we can formulate the problem as a hitting set problem: E is the ground set of elements, the cost of each element is u_e, and for each i we must hit a set T_i, where T_i contains the edges on the unique path from s_i to t_i in the tree.

The minimum multicut problem in trees generalizes the vertex cover problem. Indeed, consider a star graph G with center vertex r, with leaves $v_1,\cdots,v_n$, and with terminal pairs (s_i,t_i) for $i=1,\cdots,k$. Construct a graph H with vertex set $\{v_1,\cdots,v_n\}$, edge set $\{(s_i,t_i): i=1,\cdots,k\}$ and assign a weight of u_{rv} to any vertex v. Then $\{(r,v_i): i\in C\}$ is a multicut of G of capacity U *if and only if* $\{v_i : i \in C\}$ is a vertex cover of H of weight U.

We can get a 2-approximation algorithm for this problem by applying the algorithm in Figure 4.2. In order to do this, we must specify how to select a violated set. At the beginning of the algorithm, we root the tree at an arbitrary vertex r. Define the *depth* of a vertex v to be the number of edges in the path from v to r, and define the *least common ancestor* of vertices u and v to be the vertex x of smallest depth that lies on the path from u to v. For each i, we compute the depth d_i of the least common ancestor of s_i and t_i. Then, among the violated sets T_i, we choose a set that maximizes d_i. The resulting algorithm is the algorithm proposed by Garg et al. [GVY93a].

THEOREM 4.10 The algorithm given in Figure 4.2 is a 2-approximation algorithm for the minimum multicut problem in trees.

Proof. We will apply Theorem 4.1. For this purpose, let A be any infeasible solution, let T be the violated set selected by the algorithm (i.e. the one that maximizes the depth of the least common ancestor), and let B be any minimal augmentation of A. We only need to prove that $|T \cap B| \leq 2$. Recall that T corresponds to a path from s_i to t_i in the tree, and let a_i be the least common ancestor of s_i and t_i. Let T_1 denote the path from s_i to a_i and T_2 denote the path from a_i to t_i. Then the theorem will follow by showing that $|B \cap T_1| \leq 1$ (the proof that $|B \cap T_2| \leq 1$ is identical). Suppose that $|B \cap T_1| \geq 2$. We claim that removing all edges in $B \cap T_1$ from B except the edge closest to a_i is still a feasible solution, contradicting the minimality of B. To see this, notice that by the choice of T, for any other violated sets T_j such that $T_1 \cap T_j \neq \emptyset$, the set $T_1 \cap T_j$ is a path from some vertex in T_1 to a_i; if not, T_j would have a least common ancestor of depth $d_j > d_i$, a contradiction. Therefore, if T_j contains any edge in $B \cap T_1$, it contains the edge in $B \cap T_1$ closest to a_i. ■

The algorithm of Figure 4.2 not only constructs an approximate primal solution but also constructs an approximate dual solution. Moreover, if the capacities are integral, so is the dual solution constructed. In the case of the multicut problem, the (integral) dual is referred to as the *maximum (integral) multicommodity flow problem*: one needs to pack a maximum number of paths between terminal pairs without using any edge e more than u_e times. By Theorem 4.1, the algorithm of Figure 4.2 constructs a multicut and an integral multicommodity flow whose values are within a factor of 2 of each other.

4.8.2 THE PRIZE-COLLECTING PROBLEMS

We next show how we can derive 2-approximation algorithms for extensions of the traveling salesman problem and the Steiner tree problem. These extensions are known as the *prize-collecting traveling salesman problem* and the *prize-collecting Steiner tree problem*. In the prize-collecting traveling salesman problem, the input is an undirected graph $G = (V, E)$, nonnegative edge costs c_e, and nonnegative penalties on the vertices π_i. The goal is to find a tour on a subset of the vertices that minimizes the sum of the cost of the edges in the tour and the penalties on the vertices not in the tour. We will consider a variant in which a prespecified root vertex r must be in the tour; this is without loss of generality, since we can repeat the algorithm $n = |V|$ times, setting each vertex to be the root. The version of the prize-collecting TSP is a special case of a more general problem introduced by Balas [Bal89]. The prize-collecting Steiner tree problem is defined analogously; one needs to find a tree containing the root r which minimizes the sum of the cost of the edges of the tree plus the penalties of the vertices not spanned. The first approximation algorithms for these problems were given by Bienstock, Goemans, Simchi-Levi, and Williamson [BGSLW93]: they gave a 5/2-approximation algorithm for the TSP version (assuming the triangle inequality) and a 3-approximation algorithm for the Steiner tree version. The 2-approximation algorithms that we describe here are due to the authors [GW95a].

We first concentrate on deriving a 2-approximation algorithm for the prize-collecting Steiner tree problem; we will then show how a 2-approximation algorithm for the prize-collecting TSP can be derived from it. The 2-approximation algorithm is simply going to be the algorithm of Figure 4.3 for an appropriate formulation of the hitting set problem. Given the input graph (V, E) and root vertex r, the set of ground elements for the hitting set problem is the set of all edges E together with the set of all subsets of V not containing r; that is, the set of ground elements is $E \cup \{S : S \subseteq V - \{r\}\}$. The cost of a ground element $e \in E$ is c_e, while the cost of a ground element $S \subset V$ is $\sum_{v \in S} \pi_v$. The sets that must be hit are the sets $T_i = \delta(S_i) \cup \{S : S \supseteq S_i\}$ ranging over all $\emptyset \neq S_i \subseteq V - \{r\}$. Throughout the section, we assume that A denotes a subset of the ground set, so A contains vertex sets as well as edges; we will denote the collection of edges of A by A_e and the collection of vertex sets by A_s.

We now argue that this hitting set problem exactly models the prize-collecting Steiner tree problem. First, any feasible solution A to this hitting set problem is a feasible solution of no greater cost to the prize-collecting Steiner tree problem. Let S be the set of vertices not connected to the root r by the edges in A_e. The set $T = \delta(S) \cup \{S' \supseteq S\}$ must be hit, so some $S' \supseteq S$ must be in A. Thus the cost of A includes the penalty $\sum_{v \in S} \pi_v$. Furthermore, given any feasible solution to the prize-collecting Steiner tree problem, we get a feasible solution to the hitting set problem of no greater cost by taking the set of all edges in the Steiner tree plus the set S of the vertices not connected to the root.

Since the ground set contains two types of elements, the dual of the LP relaxation will contain two types of constraints; the one corresponding to an edge e is as usual

$$\sum_{S : e \in \delta(S)} y_S \leq c_e,$$

while the one corresponding to a set C is

$$\sum_{S : S \subseteq C} y_S \leq \sum_{v \in C} \pi_v.$$

If a dual feasible solution y satisfies the dual constraints for C_1 and C_2 at equality then it can easily be seen that it also satisfies the dual constraint for $C_1 \cup C_2$ at equality. This means that, given a solution A and a dual feasible solution y satisfying the primal complementary slackness conditions, the solution obtained by replacing the sets in A_s by their union still satisfies the primal complementary slackness conditions with y. Although it will be important to keep track of the different sets in A_s, we will always assume that the union of the sets in A_s

$y \leftarrow 0$
$A \leftarrow \emptyset$
$\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
$l \leftarrow 0$
While A is not feasible
  $l \leftarrow l+1$
  $\mathcal{V} \leftarrow \{S \in \mathcal{C} : r \notin S$ and $S \notin A_s\}$
  Increase y_S uniformly for all $S \in \mathcal{V}$ until
    either (i) $\exists e_l \in \delta(T), T \in \mathcal{V}$ such that $\sum_{S: e_l \in \delta(S)} y_S = c_{e_l}$
    or (ii) $\exists S_l \in \mathcal{V}$ such that $\sum_{S: S \subseteq S_l} y_S = \sum_{v \in S_l} \pi_v$
  If (i) then
    $a_l \leftarrow e_l$
    merge the two components of $\mathcal{C}$ spanned by e_l
  else $a_l \leftarrow S_l$
  $A \leftarrow A \cup \{a_l\}$
For $j \leftarrow l$ downto 1
  if $A - \{a_j\}$ is feasible then $A \leftarrow A - \{a_j\}$
Output A_e (and the union of the sets in A_s and y)

FIGURE 4.10

Primal-dual algorithm for the prize-collecting Steiner tree problem.

is implicitly taken before checking feasibility of A. Thus, we will regard A as feasible if the set of vertices not connected to the root r by edges in A_e can be covered by subsets of A_s.

Since the sets to be hit naturally correspond to vertex sets, we will again refer to the vertex sets S_i instead of referring to the associated subsets T_i of the ground set. In particular, we can use the algorithm of Figure 4.4 rather than the one of Figure 4.3. But, for this, we first need to understand what sets are violated and which ones are minimal. Given the definition of T_i, a violated set for the current solution A will be a union of connected components of (V, A_e) provided the union (i) does not contain the root and (ii) cannot be covered by sets of A_s. Thus the *minimal* violated sets $\mathcal{V}$ are the connected components $\mathcal{C}$ of (V, A_e) which do not contain the root and which cannot be covered by sets in A_s. We give the specialization of the algorithm of Figure 4.4 to the prize-collecting Steiner tree problem in Figure 4.10. In the figure, $\mathcal{C}$ denotes the connected components of (V, A_e) and $\mathcal{V}$ denotes the collection of minimal violated sets. Also, for simplicity, we have allowed a set $S \notin A_s$ to be violated even though it can be covered by sets of A_s. The algorithm would then simply add S to A_s without increasing any dual variable.

THEOREM 4.11 The primal-dual algorithm described in Figure 4.10 gives a 2-approximation algorithm for the prize-collecting Steiner tree problem.

Before we present the proof of this theorem, it is useful to understand what the reverse delete step really achieves in this case. First, observe that at any point during the execution of the algorithm the sets in A_s form a laminar family; i.e., any two sets in A_s are either disjoint or one is contained in the other. Moreover, if $S_1 \subset S_2$ are two sets of A_s then S_2 was added after S_1 and will be considered for removal before S_1 is. Because of the reverse delete step, in any solution B output by the algorithm, the sets in B_s will be disjoint. Furthermore, a set S in A_s will be considered for removal after all the edges in $\delta(S)$, but before all the edges with both endpoints within S. This implies that S must be kept in the reverse delete step only if all the edges in $\delta(S)$ have been removed.

Proof. Since the algorithm is equivalent to the algorithm of Figure 4.3, we can use Theorem 4.2. We must therefore show that for any infeasible solution A and any minimal augmentation B of A,

$$\sum_{i:S_i \in \mathcal{V}(A)} |B \cap T_i| \leq 2|\mathcal{V}(A)|,$$

where $\mathcal{V}(A)$ is the the collection of violated sets. In fact, looking back at the proof of Theorem 4.2, we don't need to show the inequality for *any* minimal augmentation B, but only for those which could be produced by the algorithm. Given the above discussion, we can thus assume that the sets in B_s are disjoint, that they all consist of unions of connected components of (V, A_e), and that no edge $e \in B_e$ belongs to $\delta(S)$ for some set $S \in B_s$.

Consider now the graph H formed by shrinking the connected components of (V, A_e). Let W denote the vertices of H corresponding to the sets in $\mathcal{V}(A)$, and let $W' \subseteq W$ denote the subset of these vertices corresponding to the union of the sets in B_s. Then $\sum_{i:S_i \in \mathcal{V}(A)} |B \cap T_i| = \sum_{S_i \in \mathcal{V}(A)} (|B_e \cap \delta(S_i)| + |B_s \cap \{S : S \supseteq S_i\}|) = \sum_{v \in W} d_v + |W'|$, and we must show that this quantity is no more than $2|W|$. By the observations about B_s above, if $v \in W'$, then $d_v = 0$, so that we must prove that $\sum_{v \in W - W'} d_v + |W'| \leq 2|W|$. The fact that the reverse delete step produces a minimal solution implies that any leaf of H must be a vertex of W; if a leaf of H is not in W, we could delete the corresponding edge of B_e without affecting the feasibility of B. Then, as before, we derive that $\sum_{v \in W - W'} d_v \leq 2|W - W'|$ since we are only discarding vertices of degree at least 2. Thus

$$\sum_{v \in W - W'} d_v + |W'| \leq 2|W - W'| + |W'| = 2|W| - |W'| \leq 2|W|,$$

which is the desired inequality. ■

Given that edge costs obey the triangle inequality, a 2-approximation algorithm for the prize-collecting TSP can be obtained as follows: given the input graph G, edge costs c_e, penalties π_i, and root vertex r, we apply the above algorithm for the prize-collecting Steiner tree to the graph G, edge costs c_e, penalties $\pi_i' = \pi_i/2$, and root vertex r. The resulting tree is converted to a tour by the usual technique of doubling the edges and shortcutting the resultant Eulerian tour. The proof that this algorithm is a 2-approximation algorithm for the prize-collecting TSP is similar to the proof used in Section 4.5.2 for the lower capacitated cycle problem, and we leave it as an exercise for the reader. This

2-approximation algorithm has been used for deriving approximation algorithms for more complex problems; see [BCC+94, GK96, AABV95, BRV95].

4.8.3 VERTEX CONNECTIVITY PROBLEMS

So far all network design problems discussed have involved finding minimum-cost subgraphs with certain edge-connectivity properties. However, the primal-dual method can also be applied to some vertex-connectivity problems. Ravi and Williamson [RW95] have shown that the primal-dual method gives a $2\mathcal{H}(k)$-approximation algorithm for the minimum-cost k-vertex-connected subgraph problem, in which one must find a minimum-cost set of edges such that there are at least k vertex-disjoint paths between any pair of vertices. They also present a 3-approximation algorithm for the survivable network design problem when there must be r_{ij} vertex-disjoint paths between i and j, and $r_{ij} \in \{0,1,2\}$ for all i, j. No approximation algorithms were previously known for either of these problems.

We briefly sketch how the primal-dual algorithm is used in the case of the minimum-cost k-vertex-connected subgraph problem. As in the case of general proper functions (Section 4.6), the solution is constructed in a sequence of k phases. In phase p, the current solution is augmented to a p-vertex-connected graph. By Menger's Theorem, a graph is p-vertex-connected if there does not exist any set of $p-1$ or fewer vertices such that removing the set divides the graph into two non-empty pieces. Let (V,E) be the input graph, and let F_{p-1} denote the set of edges selected at the end of phase $p-1$. To augment a $(p-1)$-vertex-connected graph (V, F_{p-1}) to a p-vertex-connected graph, we apply the algorithm of Figure 4.4 to the hitting set problem in which the ground elements are the edges of $E - F_{p-1}$. For any set of $p-1$ vertices whose removal separates (V, F_{p-1}) into two pieces S_i and S'_i, we must hit the set $T_i = \delta(S_i : S'_i) \cap (E - F_{p-1})$, where $\delta(S : S')$ denotes the set of edges with one endpoint in S and one in S'. If A is any feasible solution to this hitting set problem, then $A \cup F_{p-1}$ is a p-vertex-connected graph by Menger's Theorem. We correspond the smaller of S_i and S'_i to each violated set T_i; one can then show for this problem that the minimal violated sets S are disjoint. The algorithm of Figure 4.4 can then be applied in a straightforward way to find a low-cost augmentation A of F_{p-1}; we set F_p to $A \cup F_{p-1}$. As with Theorem 4.8, it is possible to show that the algorithm yields a 2-approximation algorithm for this hitting set problem, and using a proof similar to that of Theorem 4.9, it can be proven that the overall algorithm gives a $2\mathcal{H}(k)$-approximation algorithm for the k-vertex-connected subgraph problem.

Other results known for vertex-connectivity problems can be found in Chapter 6.

EXERCISE 4.21 Show that for star graphs and unit capacities, finding the maximum integral multicommodity flow is equivalent to a maximum matching problem.

EXERCISE 4.22 Prove that the primal-dual algorithm for the prize-collecting TSP is a 2-approximation algorithm.

EXERCISE 4.23 Show that the algorithm of Figure 4.10 returns a tree such that the sum of the cost of the edges plus *twice* the sum of the penalties of the vertices not visited is at most twice the cost of the optimum solution. For an application, see [GK96, BRV95].

EXERCISE 4.24 Does the fact that the elements are deleted *in reverse* matter for the algorithm of Figure 4.10?

4.9 CONCLUSIONS

To this point, we have concentrated mainly on showing how the primal-dual method allows the proof of good performance guarantees, and have mostly set aside the issues of running time and performance in practice. A common criticism of approximation algorithms is that they might not generate "nearly-optimal" solutions in practice. A practitioner will seldom be satisfied with a solution guaranteed to be of cost less than twice the optimum cost, as guaranteed by most of the algorithms of this chapter, and would prefer an algorithm that finds solutions within a few percent of optimal. The good news is that the studies of the primal-dual method performed thus far show that it seems to perform very well in practice, at least on some problems. The authors [WG94] report computational results with the 2-approximation algorithm for the minimum-weight perfect matching problem under the triangle inequality. They consider both random and real-world instances having between $1,000$ and $131,072$ vertices. The results indicate that the algorithm generates a matching within 2% of optimal in most cases. In over 1,400 experiments, the algorithm was never more than 4% from optimal. Hu and Wein [Wei94] implemented the algorithm for the generalized Steiner tree problem, and found that the algorithm was usually within 5% of optimal. Because of the difficulty of finding the optimal solution in this case, their instances had at most 64 vertices. Finally, Mihail and Shallcross implemented a modification of the algorithm given for the survivable network design problem for inclusion in a network design software package. Although they did no rigorous testing, they report that the algorithm does well in practice, coming within a few percent of the expected optimal solution [MSDM96].

In this chapter, we have shown the power of the primal-dual method for designing approximation algorithms for a wide variety of problems. Most of the problems considered in this chapter were network design problems, but the method is so general that it is likely to have interesting applications for many kinds of problems. Indeed, primal-dual techniques have also been applied to derive approximation algorithms for other problems, such as the feedback vertex set problem (see Chapter 9) or some of its variants in planar graphs [GW95b]. For network design problems, the moral of the chapter is that two design rules are very important. First, one should grow uniformly the dual variables corresponding to the minimal violated sets. Secondly, one should delete unnecessary edges in a reverse order before the solution is output. These rules should lead to approximation algorithms for many more problems.

ACKNOWLEDGEMENTS

The first author was supported by NSF grant 9302476-CCR. The second author was supported by an NSF Postdoctoral Fellowship, and by the IBM Corporation.

References

[AABV95] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 277–283, 1995.

[AG94] M. Aggarwal and N. Garg. A scaling technique for better network design. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 233–240, 1994.

[AKR95] A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24:440–456, 1995.

[Bal89] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[BCC+94] A. Blum, P. Chalasani, D. Coppersmith, W. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 163–171, 1994.

[BGSLW93] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993.

[Bir46] G. Birkhoff. Tres observaciones sobre el algebra lineal. *Revista Facultad de Ciencias Exactas, Puras y Aplicadas Universidad Nacional de Tucuman, Serie A*, 5:147–151, 1946.

[BMMN94] M. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser. *Network Models*. Handbooks in Operations Research and Management Science. North-Holland, 1994.

[BMW89] A. Balakrishnan, T. L. Magnanti, and R. Wong. A dual-ascent procedure for large-scale uncapacitated network design. *Operations Research*, 37:716–740, 1989.

[BRV95] A. Blum, R. Ravi, and S. Vempala. A constant-factor approximation algorithm for the k-MST problem. Manuscript, 1995.

[BT95] D. Bertsimas and C.-P. Teo. From valid inequalities to heuristics: A unified view of primal-dual approximation algorithms in covering problems. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 102–111, 1995.

[BYE81] R. Bar-Yehuda and S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.

[CFN77] G. Cornuéjols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytical study of exact and approximate algorithms. *Management Science*, 23:789–810, 1977.

[Chr76] N. Christofides. Worst case analysis of a new heuristic for the traveling salesman problem. Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[Chv79] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[Chv83] V. Chvátal. *Linear Programming.* W.H. Freeman and Company, New York, NY, 1983.

[CP80] G. Cornuéjols and W. Pulleyblank. A matching problem with side constraints. *Discrete Mathematics*, 29:135–159, 1980.

[DFF56] G. B. Dantzig, L. R. Ford, and D. R. Fulkerson. A primal-dual algorithm for linear programs. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 171–181. Princeton University Press, Princeton, NJ, 1956.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[Edm65] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards B*, 69B:125–130, 1965.

[Edm67] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71B:233–240, 1967.

[Ege31] E. Egerváry. Matrixok kombinatorius tujajdonságairól. *Matematikai és Fizikai Lapok*, 38:16–28, 1931.

[EJ70] J. Edmonds and E. Johnson. Matching: A well-solved class of integer linear programs. In R. Guy, H. Hanani, N. Sauer, and J. Schonheim, editors, *Proceedings of the Calgary International Conference on Combinatorial Structures and Their Applications*, pages 82–92. Gordon and Breach, 1970.

[EJ73] J. Edmonds and E. L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[Erl78] D. Erlenkotter. A dual-based procedure fo uncapacitated facility location. *Operations Research*, 26:992–1009, 1978.

[FF56] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[Gab90] H. N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 434–443, 1990.

[GGP+94] M. Goemans, A. Goldberg, S. Plotkin, D. Shmoys, E. Tardos, and D. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 1994.

[GGW93] H. N. Gabow, M. X. Goemans, and D. P. Williamson. An efficient approximation algorithm for the survivable network design problem. In *Proceedings of the Third MPS Conference on Integer Programming and Combinatorial Optimization*, pages 57–74, 1993.

[GH61] R. Gomory and T. Hu. Multi-terminal network flows. *SIAM Journal of Applied Mathematics*, 9:551–570, 1961.

[GK96] M. X. Goemans and J. M. Kleinberg. Improved approximation algorithms for the minimum latency problem. To appear in the *Proceedings of the Seventh Annual Symposium on Discrete Algorithms*, 1996.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin, 1988.

[GMS94] M. Grötschel, C. L. Monma, and M. Stoer. Design of survivable networks. In *Handbook in Operations Research and Management Science*. North-Holland, 1994.

[GT91] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph-matching problems. *Journal of the ACM*, 38:815–853, 1991.

[GVY93a] N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming*, 1993. To appear in *Algorithmica* under the title "Primal-dual approximation algorithms for integral flow and multicut in trees".

[GVY93b] N. Garg, V. V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 698–707, 1993.

[GW94a] M. X. Goemans and D. P. Williamson. .878-approximation algorithms for MAX CUT and MAX 2SAT. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 422–431, 1994.

[GW94b] M. X. Goemans and D. P. Williamson. Approximating minimum-cost graph problems with spanning tree edges. *Operations Research Letters*, 16:183–189, 1994.

[GW95a] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.

[GW95b] M. X. Goemans and D. P. Williamson. Primal-dual approximation algorithms for feedback problems in planar graphs. Manuscript, 1995.

[Hoc82] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11:555–556, 1982.

[IKK93] C. Imielińska, B. Kalantari, and L. Khachiyan. A greedy heuristic for a minimum-weight forest problem. *Operations Research Letters*, 14:65–71, 1993.

[JDU+74] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham. Worst-case performance bounds for simple one-dimensional packing problems. *SIAM Journal on Computing*, 3:299–325, 1974.

[JP91] M. Jünger and W. Pulleyblank. New primal and dual matching heuristics. Research Report 91.105, Universität zu Köln, 1991.

[Kle94] P. N. Klein. A data structure for bicategories, with application to speeding up an approximation algorithm. *Information Processing Letters*, 52:303–307, 1994.

[KR93] P. Klein and R. Ravi. When cycles collapse: A general approximation technique for constrained two-connectivity problems. In *Proceedings of the Third MPS Conference on Integer Programming and Combinatorial Optimization*, pages 39–55, 1993. Also appears as Brown University Technical Report CS-92-30.

[Kru56] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[Kuh55] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[Lap88] G. Laporte. Location-routing problems. In B. L. Golden and A. A. Assad, editors, *Vehicle routing: Methods and studies*, pages 163–197. North-Holland, Amsterdam, 1988.

[LMSL92] C.-L. Li, S. T. McCormick, and D. Simchi-Levi. The point-to-point delivery and connection problems: Complexity and algorithms. *Discrete Applied Mathematics*, 36:267–292, 1992.

[LNP83] G. Laporte, Y. Nobert, and P. Pelletier. Hamiltonian location problems. *European Journal of Operations Research*, 12:82–89, 1983.

[Lov75] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[MSDM96] M. Mihail, D. Shallcross, N. Dean, and M. Mostrel. A commercial application of survivable network design: ITP/INPLANS CCS network topology analyzer. To appear in the *Proceedings of the Seventh Annual Symposium on Discrete Algorithms*, 1996.

[Nic66] T. Nicholson. Finding the shortest route between two points in a network. *Computer Journal*, 9:275–280, 1966.

[Pla84] D. A. Plaisted. Heuristic matching for graphs satisfying the triangle inequality. *Journal of Algorithms*, 5:163–179, 1984.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Rag94] S. Raghavan. *Formulations and algorithms for network design problems with connectivity requirements*. PhD thesis, MIT, 1994.

[RT81] E. M. Reingold and R. E. Tarjan. On a greedy heuristic for complete matching. *SIAM Journal on Computing*, 10:676–681, 1981.

[RT87] P. Raghavan and C. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[RW95] R. Ravi and D. Williamson. An approximation algorithm for minimum-cost vertex-connectivity problems. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 332–341, 1995. To appear in *Algorithmica*.

[SPR80] K. J. Supowit, D. A. Plaisted, and E. M. Reingold. Heuristics for weighted perfect matching. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 398–419, 1980.

[Sto92] M. Stoer. *Design of Survivable Networks*, volume 1531 of *Lecture Notes in Mathematics*. Springer-Verlag, 1992.

[Str88] G. Strang. *Linear Algebra and its Applications*. Harcourt Brace Jovanovich, San Diego, CA, Third edition, 1988.

[SVY92] H. Saran, V. Vazirani, and N. Young. A primal-dual approach to approximation algorithms for network Steiner problems. In *Proceedings of Indo-US workshop on Cooperative Research in Computer Science*, pages 166–168, 1992.

[Tar75] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22:215–225, 1975.

[Vai91] P. Vaidya. Personal communication, 1991.

[vN53] J. von Neumann. A certain zero-sum two-person game equivalent to the optimal assignment problem. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games, II*, pages 5–12. Princeton University Press, Princeton, NJ, 1953.

[Vor79] O. Vornberger. *Complexity of path problems in graphs*. PhD thesis, Universität-GH-Paderborn, 1979.

[Wei94] J. Wein. Personal communcation, 1994.

[WG94] D. P. Williamson and M. X. Goemans. Computational experience with an approximation algorithm on large-scale Euclidean matching instances. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 355–364, 1994. To appear in *ORSA Journal on Computing*.

[WGMV95] D. P. Williamson, M. X. Goemans, M. Mihail, and V. V. Vazirani. An approximation algorithm for general graph connectivity problems. *Combinatorica*, 15:435–454, 1995.

[Wil93] D. P. Williamson. *On the design of approximation algorithms for a class of graph problems*. PhD thesis, MIT, Cambridge, MA, September 1993. Also appears as Tech Report MIT/LCS/TR-584.

[Win87] P. Winter. Steiner problem in networks: a survey. *Networks*, 17:129–167, 1987.

[Wol80] L. A. Wolsey. Heuristic analysis, linear programming and branch and bound. *Mathematical Programming Study*, 13:121–134, 1980.

[Won84] R. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984.

[Zel93] A. Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9:463–470, 1993.

Index

arborescence
Edmonds' algorithm, 14
minimum-cost, 9
assignment, 2

complementarity property, 28, 34
complementary slackness conditions, 5
connectivity
vertex, 45

Dijkstra's algorithm, 12, 39
downwards monotone functions, 24–28
dual-ascent algorithm, 11

edge covering, 24

feedback vertex set, 47
function
downwards monotone, 24–28
proper, 28–40
skew supermodular, 39
uncrossable, 36–38

generalized Steiner tree, 19, 23, 30
Gomory-Hu cut tree, 39

hitting set, 8–18

linear programming, 1–47
location problems, 26–27

matching
minimum-weight perfect, 2, 6–8, 31–32
maximality property, 21–23, 34, 39
minimum multicut, 40–41
minimum spanning tree, 15, 17, 19, 24
Kruskal's algorithm, 15, 17
multicommodity flow, 41, 45
multicut, 40–41

network design problems, 3, 18–19
survivable, 19, 35

partitioning problem
exact, 34
lower-capacitated, 25–26
path
Dijkstra's algorithm, 12, 39
shortest $s-t$, 9, 12, 19, 35
point-to-point connection, 32
primal
restricted, 5–6
primal-dual method, 1–47
prize-collecting Steiner tree, 41–44
prize-collecting TSP, 41, 44, 46
proper functions, 28–40

restricted primal, 5–6
reverse delete step, 13, 29, 46

set cover, 8
shortest path, 9, 19, 35
skew supermodular functions, 39
spanning tree
Kruskal's algorithm, 15, 17
minimum, 9, 15, 17, 19, 24
Steiner tree, 19
generalized, 19, 23, 30
prize-collecting, 41–44
survivable network design, 19, 35

T-join, 31
traveling salesman problem
    prize-collecting, 41, 44, 46

uncrossable functions, 36–38
uniform increase rule, 15

vertex connectivity, 45
vertex cover, 9, 11
violated sets, 10
    minimal, 12, 21, 39