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 this 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 of network design problems in the 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:

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

(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:

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

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 prob-

lems (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:

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

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

$$\begin{aligned} \text{Max} \quad & \sum_{a \in A} u_a + \sum_{b \in B} v_b & \\ \text{subject to:} & & \\ & u_a + v_b \leq c_{ab} & (a, b) \in E. \end{aligned}$$

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

$$\begin{aligned} \text{Min} \quad & \sum_{a \in A} s_a + \sum_{b \in B} s_b & \\ \text{subject to:} & & \\ & \sum_{b:(a,b)\in E} x_{ab} + s_a = 1 & a \in A \\ & \sum_{a:(a,b)\in E} x_{ab} + s_b = 1 & b \in B \\ & x_e = 0 & e \in (E - J) \\ & x_e \geq 0 & e \in J \\ & s \geq 0. & \end{aligned}$$

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:

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

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, along the way we will "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:

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

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:

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

and

$$\begin{aligned} \text{Max} \quad & \sum_{i=1}^{p} y_i \\ \text{subject to:} & \\ & \sum_{i : e \in T_i} y_i \leq c_e && e \in E \\ & y_i \geq 0 && i = 1, \ldots, p. \end{aligned}$$

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

```
1   y ← 0
2   A ← ∅
3   While ∃k : A ∩ T_k = ∅
4     Increase y_k until ∃e ∈ T_k : Σ_{i:e∈T_i} y_i = c_e
5     A ← A ∪ {e}
6   Output A (and y)
```

FIGURE 4.1

The basic primal-dual algorithm.

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 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 $p < j$ was removed already. 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

```
1   y ← 0
2   A ← ∅
3   l ← 0
4   While ∃k : A ∩ T_k = ∅
5     l ← l + 1
6     Increase y_k until ∃e_l ∈ T_k : Σ_{i:e_l ∈ T_i} y_i = c_{e_l}
7     A ← A ∪ {e_l}
8   For j ← l downto 1
9     if A − {e_j} is feasible then A ← A − {e_j}
10  Output A (and y)
```

FIGURE 4.2

Primal-dual algorithm with reverse delete step.

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

$$\beta = \begin{array}{c} \max \\ \left\{ \begin{array}{c} \text{infeasible} \\ A \subset E \end{array} \right. \end{array} \left\{ \begin{array}{c} \max \\ \text{minimal} \\ \text{augmentations } B \text{ of } A \end{array} \right\} |B \cap T(A)|, \tag{4.5}$$

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

```
1   y ← 0
2   A ← ∅
3   l ← 0
4   While A is not feasible
5     l ← l + 1
6     V ← VIOLATION(A)
7     Increase y_k uniformly for all T_k ∈ V until ∃e_l ∉ A : Σ_{i:e_l∈T_i} y_i = c_{e_l}
8     A ← A ∪ {e_l}
9   For j ← l downto 1
10    if A − {e_j} is feasible then A ← A − {e_j}
11  Output A (and y)
```

FIGURE 4.3

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

algorithm of Agrawal et al. can be considered the first highly sophisticated use of the primal-dual method in the design of approximation algorithms.

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 $j = 1, \ldots, l$,

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

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 $\gamma \sum_{i=1}^{p} y_i \leq \gamma z_{OPT}$, 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 $2k - 2$, independent 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 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:

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

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 $\delta(S)$. Sometimes we 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

First we 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

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

and the dual of its LP relaxation is:

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

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.

```
1   y ← 0
2   A ← ∅
3   l ← 0
4   While A is not feasible
5     l ← l + 1
6     V ← {minimal violated sets S}
7     Increase y_S uniformly for all S ∈ V until ∃ e_l ∈ δ(T), T ∈ V : Σ_{S:e_l∈δ(S)} y_S = c_{e_l}
8     A ← A ∪ {e_l}
9   For j ← l downto 1
10    if A − {e_j} is feasible then A ← A − {e_j}
11  Output A (and y)
```

FIGURE 4.4

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

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 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

```
1   A ← ∅
2   Comment: Implicitly set y_S ← 0 for all S ⊂ V
3   LB ← 0
4   C ← {{v} : v ∈ V}
5   For each i ∈ V
6     d(i) ← 0
7   l ← 0
8   While ∃C ∈ C : f(C) = 1
9     l ← l + 1
10    Find edge e_l = (i, j) with i ∈ C_p ∈ C, j ∈ C_q ∈ C, C_p ≠ C_q that minimizes ε = (c_{e_l} − d(i) − d(j)) / (f(C_p) + f(C_q))
11    A ← A ∪ {e_l}
12    For all k ∈ C_r ∈ C do d(k) ← d(k) + ε · f(C_r)
13    Comment: Implicitly set y_C ← y_C + ε · f(C) for all C ∈ C.
14    LB ← LB + ε Σ_{C∈C} f(C)
15    C ← C ∪ {C_p ∪ C_q} − {C_p} − {C_q}
16  For j ← l downto 1
17    If all components C of A − {e_j} satisfy f(C) = 0 then A ← A − {e_j}
18  Output A and LB
```

FIGURE 4.5

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

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 mergings 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 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.

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

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 4.6 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.