CHAPTER

5

CUT PROBLEMS AND THEIR APPLICATION TO DIVIDE-AND-CONQUER

David B. Shmoys

Divide-and-conquer is one of the most basic techniques in the design and analysis of algorithms, and yet, until recently, there have been essentially no performance guarantees known for approximation algorithms based on this approach. This chapter will present approximation algorithms for several *NP*-hard graph-theoretic cut problems, and their subsequent application as a tool for the "divide" part of divide-and-conquer approximation algorithms for a wide variety of problems.

5.1 INTRODUCTION

One of the most important paradigms in the design and analysis of algorithms is the notion of a divide-and-conquer algorithm. Every undergraduate course on algorithms teaches this method as one of its staples: to solve a problem quickly, one carefully splits the problem into two subproblems, each substantially smaller than the original, recursively solves each of these, and then pieces together the solution to each part into the overall solution desired.

This approach has also been used in the design of heuristics for *NP*-hard optimization problems, but until recently, the heuristics designed in this way were either too complicated to analyze, or had extremely poor performance guarantees. In one of the most important breakthroughs in the design and analysis of approximation algorithms in the past decade, Leighton and Rao [LR88, LR94] devised an elegant approach for using divide-and-conquer in a way that produced superior performance guarantees for a wide range of problems. The key ingredient to their approach is the design of approximation algorithms for certain cut problems. These algorithms, which are of independent interest, provide approximate max-flow min-cut theorems for multicommodity flow problems that are analogous to the famous max-flow min-cut theorem for (single-commodity) network flow of Ford and Fulkerson [FF56]. In this chapter, we will survey the techniques that were introduced by Leighton and Rao, as well as a number of advances that have been obtained subsequently in the area.

Although the methods that we will describe are much more general, we shall first illustrate this approach on a concrete example, the *minimum cut linear arrangement problem*. Suppose that we are given an undirected graph $G = (V, E)$ with n vertices, and we wish to lay out the graph on integer points of the real line; that is, for $i = 1, \ldots, n$, we assign one vertex to i, in such a way that each vertex is assigned to a value in this range; let $\sigma(i)$ denote the vertex assigned to i, for each $i = 1, \ldots, n$. We can evaluate this layout in the following way: for each $i = 1, \ldots, n$, let S_i denote the set of edges in E of the form $(\sigma(j), \sigma(k))$, where $j \leq i$ and $k > i$; we wish to choose a layout so that $\max_{i=1,\ldots,n} |S_i|$ is minimized. More generally, in the weighted version of this problem, each edge $e \in E$ has a non-negative cost $c(e)$ associated with it, and we wish to lay out the graph on the line so as to minimize

$$\max_{i=1,\ldots,n} \sum_{e \in S_i} c(e). \tag{5.1}$$

Let $OPT_{LA}(G)$ denote the optimal value. Figure 5.1 gives an instance of this problem, and feasible solution σ.

The results of Leighton & Rao and are based on an algorithm for the *graph bisection problem*, which serves as the engine for the divide-and-conquer approach. In words, this is the problem of finding a minimum-cost subset of edges whose deletion separates the graph into two components of essentially equal size. More formally, we are given an undirected graph $G = (V, E)$ where each edge $e \in E$ has a given non-negative cost $c(e)$; once again, and throughout this chapter, we shall let n denote the number of vertices in the given graph. We wish to partition the vertex set into V_1 and V_2 such that $|V_1| = \lfloor n/2 \rfloor$ (and hence $|V_2| = \lceil n/2 \rceil$) so as to minimize $\sum_{e \in \delta(V_1)} c(e)$, where for any $S \subseteq V$, $\delta(S)$ denotes the set of edges $\{(u, v) = e \in E : u \in S, v \notin S\}$.

Suppose that we have a ρ-approximation algorithm for the graph bisection problem: that is, the cost of the bisection found, $\mathcal{B}(G)$, is no more than ρ times the cost of the optimal bisection. This can be used to derive the following divide-and-conquer algorithm for the minimum cut linear arrangement problem. Apply the bisection algorithm to your input G to obtain V_1 and V_2; we shall lay out V_1 on the set $\{1, 2, \ldots, \lfloor n/2 \rfloor\}$ and lay out V_2 on the set $\{\lfloor n/2 \rfloor + 1, \ldots, n\}$. Thus, we have divided the problem into two problems of half the size, and we recursively compute good layouts for the graphs induced by V_1 and V_2, which we call G_1 and G_2, respectively. Of course, when there is only one vertex to be laid out, there is no need for further recursion.

How good is this divide-and-conquer algorithm for the minimum cut linear arrangement problem? Let $\mathcal{A}(G)$ denote the objective function value of the solution produced by this algorithm on input G. We will show that $\mathcal{A}(G) = O(\rho \log n) \cdot OPT_{LA}(G)$ for any graph G; that is, it is a $O(\rho \log n)$-approximation for the minimum cut linear arrangement problem. Consider the top level of the recursion: the edges of the graph G can be partitioned into three sets: the edges of G_1, the edges of G_2, and the edges in $\delta(V_1)$; the graphs G_1 and G_2 are independent problems in the sense that in evaluating the objective function value (5.1) of a layout, for $i = 1, \ldots, \lfloor n/2 \rfloor$, none of the edges in G_2 are relevant, and for $i = \lfloor n/2+1 \rfloor, \ldots, n$, none of the edges in G_1 are relevant (see Figure 5.1). More precisely, each subset of edges S_i, $i = 1, \ldots, \lfloor n/2 \rfloor$ can be partitioned into two subsets: those in G_1 and those in $\delta(V_1)$ and hence the total cost of those edges can be bounded by $\mathcal{A}(G_1) + \mathcal{B}(G)$. For each subset S_i, $i = \lfloor n/2+1 \rfloor, \ldots, n$, the total cost of these edges can be similarly bounded by $\mathcal{A}(G_2) + \mathcal{B}(G)$. Hence,

$$\mathcal{A}(G) \leq \max\{\mathcal{A}(G_1), \mathcal{A}(G_2)\} + \mathcal{B}(G).$$

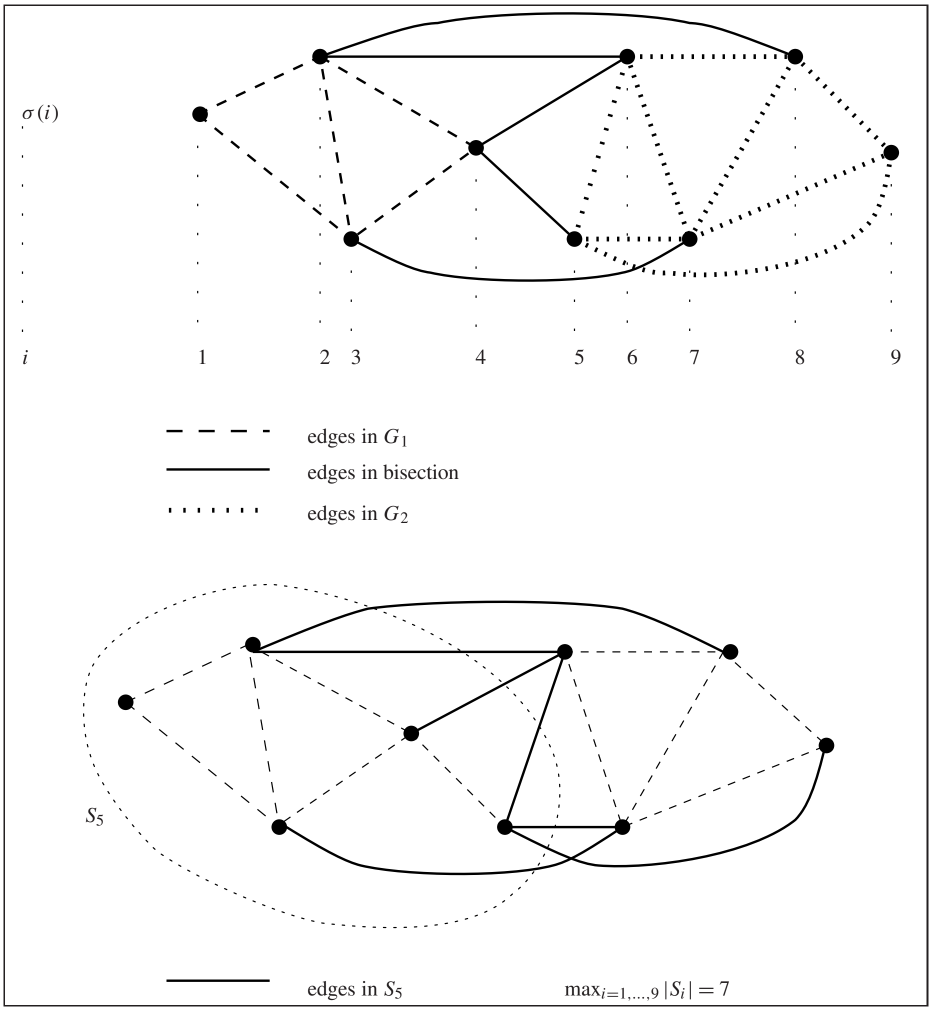

FIGURE 5.1

An instance of the minimum cut linear arrangement problem

Observe that we can construct a bisection from any linear arrangement σ: let V_1 and V_2 be, respectively, those vertices mapped by σ to values no more than $\lfloor n/2 \rfloor$ and those mapped to values greater than $\lfloor n/2 \rfloor$. The cost of the linear arrangement σ is, by definition, at least the cost of this bisection. Consequently, the cost of the optimal bisection is at most $OPT_{LA}(G)$, and hence the bisection used in the algorithm has cost $\mathcal{B}(G) \leq \rho OPT_{LA}(G)$. But this implies that

$$\mathcal{A}(G) \leq \max\{\mathcal{A}(G_1), \mathcal{A}(G_2)\} + \rho OPT_{LA}(G). \tag{5.2}$$

It is easy to see that this implies the claimed performance guarantee. With each level of recursion, the size of the relevant graphs is reduced by a factor of 2. Hence, there are $\lceil \log n \rceil$ levels of recursion (unless explicitly noted, any logarithm in this chapter has base equal to 2). With each level of recursion, we incur a cost of $\rho OPT_{LA}(\bar{G})$ for some subgraph $\bar{G}$ of G; clearly, $OPT_{LA}(\bar{G}) \leq OPT_{LA}(G)$. Hence, $\mathcal{A}(G) = O(\rho \log n) \cdot OPT_{LA}(G)$. A straightforward inductive argument can be used to write out a more formal proof.

In fact, this analysis did not require that we find a near-optimal bisection. It would have been sufficient to produce a partition in which, say, both V_1 and V_2 have at most $2n/3$ vertices. So if we can produce such a *balanced cut* for which the cost $\sum_{e \in \delta(V_1)} c(e)$ is within a factor of ρ of the optimal bisection cost, then we could still apply the same analysis. This is precisely what Leighton & Rao did: they gave an algorithm that produced a balanced cut of cost within a factor of $O(\log n)$ of the optimal bisection cost. Consequently, they obtained an $O(\log^2 n)$-approximation algorithm for the minimum cut linear arrangement problem. In Section 5.5 we will discuss a variety of balanced cut theorems and their application to several different optimization problems. It is important to note that this framework for obtaining approximation algorithms by divide-and-conquer was proposed by Bhatt & Leighton [BL84], who focused attention on designing an approximation algorithms for the graph bisection problem.

We shall next try to motivate the mathematical ideas underlying Leighton & Rao's algorithm to find a good balanced cut. The balanced cut problem seems superficially related to the *minimum (s,t)-cut problem*, where one is given a graph and two specified nodes $s, t \in V$, and wishes to find a minimum-cost subset of edges whose deletion disconnects the nodes s and t. One of the most fundamental results in combinatorial optimization is the max-flow min-cut theorem of Ford and Fulkerson [FF56], which states that the total cost of the minimum cut is exactly equal to the *maximum flow* value in the graph between s and t. In the latter problem, flow may be shipped along any path between s and t, and the aim is to maximize the total flow shipped, subject to the constraint that for each edge $e \in E$, the total flow shipped on paths that contain e is at most $c(e)$. It is easy to see that for any flow and any cut, the value of the flow can be no more than the total cost of the cut. Ford and Fulkerson gave an algorithm that simultaneously constructed a flow and a cut of equal value, and hence each is an optimal solution to its respective optimization problem.

The balanced cut problem is more closely related to a so-called multicommodity generalization of the maximum flow problem. In multicommodity network flow problems, there are k pairs of terminals (s_i, t_i), $i = 1, \ldots, k$, and the edge capacity $c(e)$ limits the total flow, over all commodities, that can be shipped through edge e, for each $e \in E$. There are two variants of this problem that we will consider. In the *maximum multicommodity flow*, we simply want to maximize the total amount that is shipped between pairs of terminals. In the *maximum concurrent flow problem*, there is a specified demand $d(i)$, for each commodity $i = 1, \ldots, k$. One might ask: can we find a feasible solution in which the total flow between s_i and t_i is at least $d(i)$, for each $i = 1, \ldots, k$? In the maximum concurrent flow problem, we wish to maximize λ such that the demands $\lambda d(i)$, $i = 1, \ldots, k$, can be met.

It is common to view the single-commodity max-flow min-cut theorem as having two parts: the weak duality theorem, which states that, for any instance, the maximum flow value is at most the minimum cut capacity, and the strong duality theorem, which states that, for any instance, there exists a feasible flow and a feasible cut for which their objective function values are equal. It is easy to obtain generalizations of the weak duality theorem in these multicommodity settings: for any instance, the maximum multicommodity flow value is at most the optimal value of the *minimum multicut problem*, and the maximum concurrent flow value is at most the optimal value of the *sparsest cut problem*. Unfortunately, the strong duality theorem does not generalize to either of these pairs of problems. Furthermore, whereas both multicommodity generalizations of the flow problem can be solved (via linear programming algorithms) in polynomial time, the minimum multicut and sparsest cut problems are *NP*-hard. (In contrast, the variants of these multicommodity flow problems in which we restrict attention to integer-valued flows in *NP*-hard.) Leighton & Rao [LR88] gave an approximate multicommodity max-flow min-cut theorem, by giving an algorithm that finds a feasible concurrent flow and a corresponding sparse cut for which the objective function values are within a logarithmic factor of each other. Consequently, this algorithm is an approximation algorithm for the sparsest cut problem with a logarithmic performance guarantee. This algorithm can be adapted to yield the approximation algorithm for the balanced cut problem that was claimed above, in the discussion of the minimum cut linear arrangement problem.

The work of Leighton & Rao provided an exciting starting point for research on approximation algorithms for cut problems and their application to divide-and-conquer algorithms for other types of problems. Furthermore, their techniques for cut problems have subsequently been applied *directly* to other problems, such as the feedback arc set problem in directed graphs; a divide-and-conquer approach is still employed, but the "divide" part of the algorithm exploits the structure of the problem at hand to obtain a superior performance guarantee. In this chapter, we shall try to highlight some of the algorithmic techniques that have been developed in this area, and give an overview of the results that have been obtained.

5.2 MINIMUM MULTICUTS AND MAXIMUM MULTICOMMODITY FLOW

In this section, we will consider the *minimum multicut problem* and the *maximum multicommodity flow problem* for undirected graphs. We shall see that these problems are closely related, and we shall exploit this relationship to derive an approximation algorithm for the cut problem.

5.2.1 MULTICUTS, MAXIMUM MULTICOMMODITY FLOW, AND A WEAK DUALITY THEOREM

Given an undirected graph $G = (V, E)$ and k pairs of vertices $(s_1, t_1), \ldots, (s_k, t_k)$, a *multicut* is a subset of edges $F \subseteq E$, such that if all edges in F are deleted, none of the pairs (s_i, t_i), $i = 1, \ldots, k$, are in the same connected component in the remaining graph $\bar{G} = (V, E - F)$. In the minimum multicut problem, we are also given a positive cost $c(e)$ for each edge $e \in E$, and we wish to find the multicut of minimum total cost. We shall let m denote the number of edges, $|E|$.

Let $\mathcal{P}_i$ denote the set of all paths between s_i and t_i, $i = 1, \ldots, k$. The minimum multicut problem can be stated as the following integer linear programming problem:

$$\text{minimize} \quad \sum_{e \in E} c(e)x(e) \tag{5.3}$$

subject to

$$\sum_{e \in P} x(e) \geq 1, \qquad \text{for each } P \in \mathcal{P}_i, i = 1, \ldots, k, \tag{5.4}$$

$$x(e) \in \{0, 1\}, \qquad \text{for each } e \in E. \tag{5.5}$$

It is easy to see that if $k = 1$, then this problem reduces to the traditional minimum (s,t)-cut problem in undirected graphs. Unfortunately, the more general problem is *NP*-hard [DJP+92], and hence we shall consider approximation algorithms for it. Just as the optimization algorithm for the minimum (s,t)-cut is based on solving a maximum (single-commodity) flow problem, we shall derive an approximation algorithm for the multicut problem that is based on solving a maximum multicommodity flow problem, which is dual to it.

The *maximum multicommodity flow problem* is as follows: given an undirected graph $G = (V, E)$, where each edge $e \in E$ has an associated positive capacity $c(e)$, and k pairs of terminal vertices, (s_i, t_i), $i = 1, \ldots, k$, assign a flow value $f(P)$ to each path $P \in \mathcal{P}_i$, $i = 1, \ldots, k$, so that, for each edge $e \in E$, the total flow value assigned to paths that contain this edge is at most $c(e)$. The aim is to maximize the total flow between all of the given pairs of vertices. Let $\mathcal{P}_i(e)$ denote the set of all paths between s_i and t_i that contain edge e; that is, $\mathcal{P}_i(e) = \{P : e \in P, P \in \mathcal{P}_i\}$, $i = 1, \ldots, k$. This problem can be formulated as the following linear programming problem.

$$\text{maximize} \quad \sum_{i=1}^{k} \sum_{P \in \mathcal{P}_i} f(P) \tag{5.6}$$

subject to

$$\sum_{i=1}^{k} \sum_{e \in \mathcal{P}_i(e)} f(P) \leq c(e), \qquad \text{for each } e \in E, \tag{5.7}$$

$$f(P) \geq 0, \qquad \text{for each } P \in \mathcal{P}_i,\ i = 1, \ldots, k. \tag{5.8}$$

Note that in this case, we do not require that the variables be restricted to integer values, i.e., it is a linear programming problem, not an integer linear programming problem.

The reader should note that the same input is required for both the maximum multicommodity flow problem and the minimum multicut problem. We shall argue next that, for any such input, if we consider any multicut and any feasible multicommodity flow, then the total flow value is at most the cost of the multicut; that is, the weak duality theorem for single-commodity case generalizes to this setting. Consider any unit of flow that uses the path P between s_i and t_i. The multicut must contain at least one of edges in this path P. As in the single-commodity case, we can view the total cost of a cut as the capacity of this cut. Thus, each unit of flow of the total multicommodity flow "uses up" at least one unit of the total capacity of the multicut, which implies the claim. Hence, we have obtained the following weak duality theorem.

LEMMA 5.1 For any input to the maximum multicommodity flow and the minimum multicut problems, the total flow value of any feasible flow is at most the cost of any multicut.

5.2.2 FRACTIONAL MULTICUTS, PIPE SYSTEMS, AND A STRONG DUALITY THEOREM

Garg, Vazirani, & Yannakakis [GVY93] give an approximation algorithm for the minimum multicut problem that first solves the linear relaxation of (5.3)-(5.5), in which the constraints (5.5) are replaced by

$$x(e) \geq 0, \qquad \text{for each } e \in E, \tag{5.9}$$

and then uses this optimal *fractional multicut* to guide the search for a good (integer) multicut. Consequently, it is useful to have a good understanding of this linear relaxation.

Feasible solutions to the linear relaxation (5.4) and (5.9) have a natural physical interpretation. For each edge $e \in E$, one can view its decision variable $x(e)$ as specifying the length of edge e. From this perspective, the constraints (5.4) can be described as requiring that the edge lengths have the following property: for each path P between s_i and t_i, the total length of P is at least 1. Equivalently, we could require that the length of the shortest path between s_i and t_i is at least 1. What physical interpretation does this imply for the objective function? The contribution of each edge $e \in E$ to the objective function is $c(e)$ times its length. Suppose that we view the graph as a system of pipes, where the edges correspond to the pipes themselves, and the vertices correspond to junction points at which the pipes meet. In this case, if we interpret $c(e)$ as the cross-section area of the pipe corresponding to the edge e, then $c(e)$ times its length is exactly equal to the volume of this pipe, and the overall objective function is the total volume of the pipe system. So the linear relaxation is to find the minimum-volume pipe system for the input in which s_i and t_i are at least 1 unit apart from each other, for each $i = 1, \ldots, k$.

The linear program for the maximum multicommodity flow problem, (5.6)-(5.8), is the dual linear program of minimum fractional multicut problem, (5.3), (5.4), & (5.9). To reinforce our intuition about these linear programs, we shall argue directly that any feasible length function for the pipe system gives an upper bound on the total value of any feasible multicommodity flow. Consider any unit of flow that uses the path P between s_i and t_i. This unit of flow "uses up" $\sum_{e \in P} x(e)$ units of volume. However, we have constrained this sum to be at least 1; that is, each unit of flow sent between an (s_i, t_i) pair consumes at least one unit of volume. Hence, the total flow value is at most the total volume of the pipe system. Of course, the strong duality theorem of linear programming states that the optimal value of the primal is equal to the optimal value of the dual, provided that an optimal solution exists (see, for example, Chvátal [Chv83]); hence, the minimum feasible pipe-system volume is equal to the total value of the maximum multicommodity flow.

5.2.3 SOLVING THE LINEAR PROGRAMS

We have already indicated that the approximation algorithm of Garg, Vazirani, & Yannakakis will first find an optimal fractional multicut. As presently stated, this might appear to be a daunting task, since the size of this linear program is exponential. However, we shall argue that one can still find an optimal solution in polynomial time, and that there are a number of approaches to doing so.

Perhaps the simplest approach is to observe that both the minimum fractional multicut problem and the maximum multicommodity flow problems have equivalent formulations that are of polynomial size. We first consider the maximum multicommodity flow problem. We define a flow variable $f_i(e)$ for each commodity $i = 1, \ldots, k$, and each edge $e \in E$. For each vertex $v \in V - \{s_i, t_i\}$, we require that the total flow of commodity i into v is equal to the total flow of commodity i out of vertex v. For each edge $e \in E$, we require that the total flow (in both directions) on e over all commodities is at most $c(e)$. The total flow of commodity i is computed by considering the net flow of commodity i out of vertex s_i, and the objective function is, as before, the sum of these totals over all commodities.

It is not hard to show that this more compact linear program is equivalent to (5.6)-(5.8). The more compact version has $O(km)$ variables and $O(m + kn)$ constraints, and hence can be solved in polynomial time, using any polynomial-time linear programming algorithm (e.g., that of Khachiyan [Kha79] or Karmarkar [Kar84]). Furthermore, this optimal solution can be converted, in polynomial time, to an optimal solution to the linear program (5.6)-(5.8) by standard "flow decomposition" methods (see, e.g., Ahuja, Magnanti, & Orlin [AMO93]). Hence, we can find an optimal solution to the maximum multicommodity flow problem in polynomial time.

If we consider the dual linear program of this more compact formulation of the maximum multicommodity flow, then we obtain a linear program equivalent to (5.3), (5.4), & (5.9). In fact, we can easily compute an optimal pipe system from the optimal primal and dual solutions to the compact formulation: set $x(e)$, the length of edge e, to be the optimal value to the dual variable corresponding to the primal constraint that the total flow on edge e is at most $c(e)$.

Another option is to rely on the nature of the first polynomial-time algorithm for linear programming, the ellipsoid algorithm. One of the most important features of this algorithm is that in order to obtain a polynomial-time algorithm for a linear program with an exponential number of constraints, we need only provide a polynomial-time subroutine that, given a solution x, either proves that x is feasible, or else finds a violated constraint (see, for example, Grötschel, Lovász, & Schrijver [GLS88]). This subroutine is said to solve the *separation problem* for this linear program. For the minimum fractional multicut problem, given by (5.3), (5.4), & (5.9), the subroutine required is just a shortest-path procedure: we need to verify that the length (with respect to x) of the shortest path between s_i and t_i is at least 1, for each $i = 1, \ldots, k$. Either we have identified an (s_i, t_i) pair for which the length of the shortest path is less than 1, in which case this shortest path corresponds to a violated constraint, or else we have shown that the current length function is feasible. Hence, the ellipsoid algorithm can be used to solve our (exponential-sized) linear program in polynomial time.

Finally, we will see that a nearly-optimal length function would work essentially as well in finding a good multicut. There has been a sequence of results obtained in designing fully polynomial approximation schemes for the maximum concurrent flow problem and other related problems. The goal of this line of research is to design algorithms that take advantage of the combinatorial structure of these problems (as opposed to a general purpose LP algorithm) in order to obtain more efficient algorithms, albeit with the disadvantage of finding solutions of objective function value within a factor of $(1+\epsilon)$ of optimal. It is important to note that unlike for other problems that we have been discussing here, these approximation schemes are for problems solvable in polynomial time. Since we are losing a logarithmic factor in finding a multicut from the LP solution, it makes sense to relax our aim to finding a near-optimal LP solution, especially since this appears to be an easier computational problem.

Shahrokhi & Matula [SM90] have proposed a combinatorial algorithm for a rather special case of the maximum concurrent flow problem that delivers a flow of objective function value within a factor of $(1+\epsilon)$ of optimal in time bounded by a polynomial in the size of the graph and $1/\epsilon$. The key to analyzing the running time of this algorithm is an exponential potential function, which has been the basis for several subsequent papers as well. Klein, Plotkin, Stein, & Tardos [KPST94] and Leighton, Makedon, Plotkin, Stein, Tardos, & Tragoudas [LMP$^+$95] subsequently improved and extended this result to derive an algorithm for the maximum concurrent flow problem for which the running time for a k-commodity problem is competitive with the running time for k single-commodity maximum flow computations. This was generalized by Plotkin, Shmoys, & Tardos [PST95] to yield a fully polynomial approximation scheme for a wide range of combinatorially defined linear programming problems, called *fractional packing and covering problems*. Results similar to those in [PST95] were independently obtained by Grigoriadis & Khachiyan [GK94].

We will explain the main ideas behind the framework proposed in [PST95] and show how it can be applied to the problem of computing a maximum multicommodity flow. A *fractional packing problem* is a linear programming problem of the following form: for a given matrix $A \in \Re^{m \times n}$ and vector $b \in \Re^m$, we wish to minimize λ subject to $Az \leq \lambda b$, $z \in Q$, where Q is a polytope such that $Az \geq 0$ for each $z \in Q$. The idea of the algorithm is that Q is supposed represent "easy" constraints, and that $Az \leq \lambda b$ are supposed to represent "complicating" constraints. The sense in which Q is "easy" is that the algorithm assumes that there is a fast subroutine that, given a positive row vector $y \in \Re^m$, computes $z^* = \text{argmin}\{cz : z \in Q\}$, where $c = yA$.

Before explaining the ideas underlying this algorithm, we will first show how to cast the maximum multicommodity flow problem in this setting. We will perform a bisection search for the maximum value f such that there is feasible flow of total value f. We will work from the exponential linear programming formulation (5.6)-(5.8). The objective function (5.6) has been transformed into a constraint that the total flow value is exactly equal to f. The polytope Q is the set of of solutions satisfying this total flow constraint as well as the non-negativity constraints (5.8); note that this is just a rescaled simplex. The constraints $Az \leq \lambda b$ correspond to the capacity constraints (5.7), where λ reflects the amount by which the capacities need to be multiplied so as to ensure that z is a feasible flow. Thus, for a candidate flow value f, we must decide if the optimum λ^* is at most 1. If we can approximate λ within a factor of $(1+\epsilon)$, then we can also approximate the optimal flow value within a factor of $(1+\epsilon)$.

Finally, we need to provide the subroutine to optimize over Q. There always exists an optimal solution at one of its extreme points; each extreme point of Q has exactly one coordinate set equal to f, and the rest set equal to 0. Hence, an optimal solution can be found by determining the smallest objective coefficient and setting the corresponding coordinate to f. In this setting, each coordinate of y corresponds the capacity constraint of one edge $e \in E$. The vector yA specifies the objective function coefficients, and the coordinate of yA corresponding to the flow variable $f(P)$ has value equal $\sum_{e \in P} y(e)$. Hence, we wish to find the coordinate, or equivalently the path between some (s_i, t_i) pair, for which the total edge length (with respect to the lengths y) is smallest. This can be done, for example, by Dijkstra's algorithm, starting at each source node s_i, $i = 1, \ldots, k$, and then choosing the minimum-length path found over all k computations. Hence, the maximum multicommodity flow problem can be cast into the framework of [PST95].

The main idea of the algorithm of [PST95] is as follows: iteratively maintain a solution z (or in our case, a flow of value f). This solution is feasible for some choice of λ, and we wish to change z so that the corresponding minimal choice of λ decreases over the execution of the algorithm. In our application, λ can be viewed as the minimum value so that the current flow is feasible with respect to the *adjusted capacities* $\lambda c(e)$. For the current solution (z, λ), we wish to focus on the constraints $Az \leq \lambda b$ that are most important in determining λ; in the flow setting, these correspond to the edges e that are saturated, or nearly saturated, to their adjusted capacity. For each of these constraints, we wish to perturb the current z so as reduce the value of the left-hand side; in our example, the total flow on each nearly saturated edge should be reduced. In our particular case, one way to do this would be to find a $s_i - t_i$ path for which none of the edges is nearly saturated, increase the flow on that path, and decrease the flow on all other paths (uniformly) so that once again the total flow value is f. Instead, the algorithm selects a path as follows: for each edge e, we set $y(e) = (1/c(e)) \cdot \exp(\alpha f(e)/c(e))$, where $f(e)$ denotes the total flow on e in the current solution and α is a parameter that is proportional to $1/\lambda$; then compute a shortest $s_i - t_i$ path with respect to these lengths $y(e)$. Observe that this approach can be viewed as a continuous version of the path selection rule described above: short edge lengths correspond to undersaturated edges. In general, we compute $y_i = (1/b_i) \cdot \exp(\alpha a_i z/b_i)$, where a_i is the ith row of A, compute $z^* = \text{argmin}\{yAz : z \in Q\}$, and set $z = (1-\sigma)z + \sigma z^*$, where $0 < \sigma < 1$ is a parameter set by the algorithm.

The analysis of the running time of the algorithm is based on showing that each iteration decreases the exponential potential function $\Phi = \sum_i \exp(\alpha a_i z/b_i)$. The overall running time depends on the *width* ρ of the formulation, which is the minimum λ such that *every* $z \in Q$ is feasible: $\rho = \max_{z \in Q} \max_i a_i z/b_i$. Plotkin, Shmoys, & Tardos show that $O(\rho\epsilon^{-2} \log(m\epsilon^{-1}))$ iterations suffice to find a solution within a factor of $(1+\epsilon)$ of optimal, where m denotes the number of rows in A.

The width of the formulation proposed above for the maximum multicommodity

```
procedure Pipe-Cut(G, x)
  { we shall assume that x(e) is given as a fraction with denominator at most Δ }
  V̄ ← V(G);  l ← 0;  ϵ ← 1/(2Δ²);
  repeat
    fix i such that {s_i, t_i} ⊆ V̄;
    r* = argmin{C_x(s_i, r)/V_x(s_i, r) : r = dist_x(s_i, v) − ϵ, v ∈ V̄};
    l ← l + 1; V_l ← B_x(s_i, r*); V̄ ← V̄ − V_l;
  until |{s_i, t_i} ∩ V̄| ≤ 1, for each i = 1, ..., k;
  output (V_1, V_2, ..., V_l, V̄).
```

FIGURE 5.2

The algorithm Pipe-Cut

flow is at most $\max_e f/c(e)$; if there are small capacity edges, then this quantity can be quite large. However, if we delete all edges of capacity at most $\epsilon f/m$, then the resulting width is at most m/ϵ. Furthermore, by modifying the instance in this way we can change the optimum by at most a further $(1+\epsilon)$ factor. The subroutine required for each iteration is nothing more than k shortest path computations, and so if we ignore log factors and set ϵ to a constant, we get an overall running time of $\tilde{O}(km^2)$. Significantly, by rescaling y, we can also show that the algorithm produces a near-optimal solution x for the linear program (5.3), (5.4), and (5.9), or in other words, a near-optimal fractional multicut. Finally, we should note that there exist more sophisticated approaches to formulate the maximum multicommodity flow problem as fractional packing problem, which lead to improved running times, but we shall instead discuss these ideas in the context of the maximum concurrent flow in Section 5.3.

5.2.4 FINDING A GOOD MULTICUT

Garg, Vazirani, & Yannakakis [GVY93] gave an algorithm that takes as input a pipe system of volume ϕ, and computes a multicut of cost $O(\log k)\phi$. If we start with the optimal pipe system of volume ϕ^*, since ϕ^* is a lower bound on the minimum-cost multicut, the resulting algorithm is an $O(\log k)$-approximation algorithm.

Next we describe the algorithm Pipe-Cut of Garg, Vazirani, & Yannakakis, which is summarized in Figure 5.2.

The algorithm iteratively produces the desired multicut. At the start of iteration l, the Pipe-Cut has already produced a sequence of disjoint sets of vertices $V_1, \ldots, V_{l-1}$, with the property that, for each $j = 1, \ldots, l-1$, the set V_j contains at most one of s_i and t_i, for each $i = 1, \ldots, k$; furthermore, for each set V_j we have specified one commodity $i(j)$, such that the set V_j contains exactly one of the terminals $s_{i(j)}$ and $t_{i(j)}$. Let $\bar{V} = V - (V_1 \cup \cdots \cup V_{l-1})$, and consider the graph induced by $\bar{V}$ and its corresponding pipe system. If there does not exist a commodity i such that both s_i and t_i are in $\bar{V}$, then we have found a multicut: $\{e \in E: \ e = (u, v), \ u \in V_j, \ v \in V_{j'}, j \neq j'\}$, where $V_l = \bar{V}$, and this is the desired output. Otherwise, select any commodity i such that both s_i and t_i are in $\bar{V}$. In this iteration will find a particular subset $V_l \subset \bar{V}$ such that exactly one of s_i and

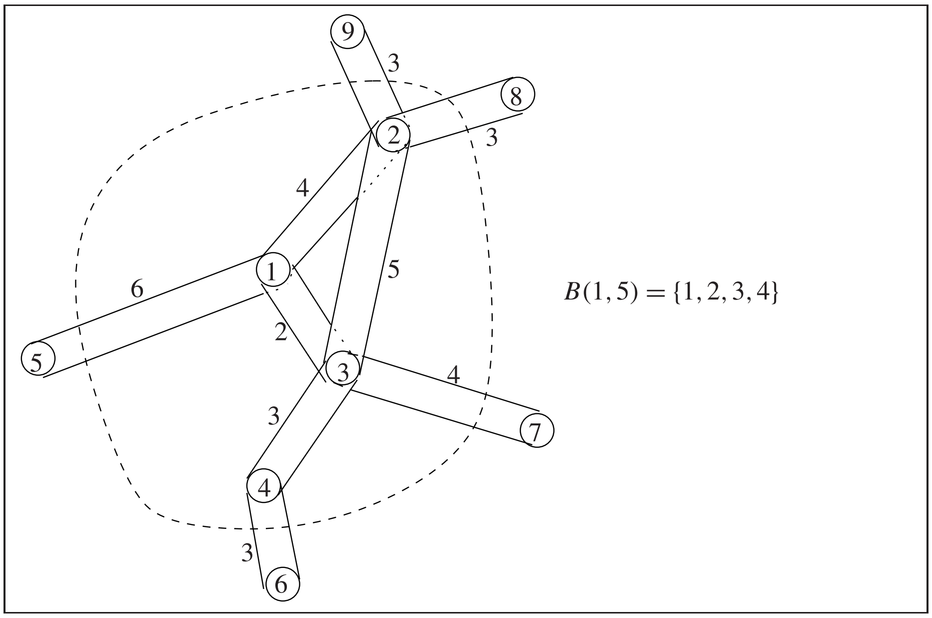

FIGURE 5.3

A pipe system & a ball of radius 5 centered at vertex 1

t_i are in V_l, and hence we can set $i(l) = i$.

Before describing the details of the remainder of this iteration, we first introduce some notation. Let $B_x(s_i, r)$ denote the set of vertices that are contained within a ball of radius r centered at vertex s_i in the pipe system, where the distances are given with respect to the length function x. Figure 5.3 gives an example of a portion of a pipe system and ball centered at one of its vertices. More formally, let $dist_x(u, v)$ denote length of the shortest path between vertices u and v with respect to the length function x, and, for each $v \in V$, set

$$B_x(v, r) = \{u \in V : dist_x(u, v) \leq r\}. \tag{5.10}$$

We shall define the part of the pipe system that is within radius r of s_i in the following way: first, we include each pipe that corresponds to an edge with both of its endpoints in $B_x(s_i, r)$; second, for each edge with exactly one endpoint in $B_x(s_i, r)$, we include the fraction of the corresponding pipe that is still within a distance r of s_i; finally, we will add a "point volume" at s_i of volume ϕ/k. More formally, the volume of the pipe system within radius r of vertex s_i is defined to be

$$V_x(s_i, r) = \phi/k + \sum_{u,v \in B_x(v,r)} c(u, v)x(u, v) + \sum_{(u,v) \in \bar{\delta}(B_x(s_i,r))} c(u, v)(r - dist_x(v, u)), \tag{5.11}$$

where $\bar{\delta} = \{(u, v) \in E : u \in B_x(s_i, r)$ and $v \in \bar{V} - B_x(s_i, r)\}$.

We shall set $V_l = B_x(s_i, r)$ for a judicious choice of radius r. Let the cost

$$C_x(s_i, r) = \sum_{(u,v)\in\bar{\delta}(B_x(s_i,r))} c(u, v). \tag{5.12}$$

We shall choose the radius r to be a value less than 1/2 such that $C_x(s_i, r)/V_x(s_i, r)$ is sufficiently small. More precisely, to guarantee the claimed performance, we need only choose a radius $r = \rho_l$ for which this ratio is at most $2 \ln 2k$; in fact, it is possible to choose the radius essentially minimizing this ratio, and the minimum will be guaranteed to be this small.

We shall see that it is easy to find a good choice for ρ_l, but first we will prove a few useful facts about the function $V_x(s_i, r)$ (as a function of r). If there does not exist a vertex v such that $dist_x(s_i, v) = r$, then this function is differentiable at r, and its derivative is equal to the total cost of the cut defined by the current radius r, $C_x(s_i, r)$. The function might not even be continuous at these breakpoints corresponding to vertices in $\bar{V}$, but it is an increasing function of r nonetheless. Furthermore, the cost $C_x(s_i, r)$ changes only at these breakpoints. Hence, if we want to minimize the cost per volume ratio, this minimum is achieved as the radius approaches one of these breakpoints. Since the algorithm depends on the radius only in selecting $V_l = B_x(s_i, \rho_l)$, we need only set ρ_l to a value sufficiently close to the limiting point.

There are two main points in analyzing the performance of the algorithm Pipe-Cut. The first step is to show that the algorithm does produce a feasible multicut. This is quite simple, since we must only prove that the set V_l contains at most one terminal for each commodity. To prove this, we observe that the feasibility of the length function implies that, for each commodity i, its two terminals s_i and t_i are at least distance 1 apart from each other. However, we set V_l to be a ball of radius less than 1/2, and any two points within it must be of distance less than 1 apart. The second main point is to show that $C_x(s_i, \rho_l)/V_x(s_i, \rho_l) \le 2 \ln 2k$, and then show that this implies the claimed performance for the algorithm.

For notational simplicity, let $f(r) = V_x(s_i, r)$. Recall that

$$C_x(s_i, r) = \frac{\partial V_x(s_i, r)}{\partial r} = \frac{df}{dr},$$

whenever the derivative is well-defined. Suppose that for each radius r within the interval $(0, 1/2)$, the ratio $C_x(s_i, r)/V_x(s_i, r) > 2 \ln 2k$. If we consider any interval (r_1, r_2) throughout which $f(r)$ is differentiable, we have that $f'(r)/f(r) > 2 \ln 2k$, or equivalently, that

$$\frac{d}{dr}(\ln f(r)) > 2 \ln 2k.$$

If we integrate both sides, we see that this implies that

$$\ln f(r_2) - \ln f(r_1) > (r_2 - r_1)(2 \ln 2k). \tag{5.13}$$

If f were differentiable throughout the interval $(0, 1/2)$, (5.13) would imply that

$$\ln(f(1/2)/f(0)) > \ln 2k;$$

that is, $f(1/2) > 2kf(0) = 2\phi$. But this is impossible, since the total volume in the pipe system, including the additional point volume, is at most $(1 + 1/k)\phi$, and hence $f(1/2)$ cannot exceed this value. Of course, f might not be differentiable throughout the interval $(0, 1/2)$. We can apply the same trick to each interval throughout which f is differentiable. By combining the results of these integrals, along with the fact that f does not decrease at any of the breakpoints, we can still conclude that $\ln f(1/2) - \ln f(0) > \ln 2k$, which is not possible. Hence, we have shown the following lemma.

LEMMA 5.2 There exists a radius $r < 1/2$, such that $C_x(s_i, r)/V_x(s_i, r) \leq 2 \ln 2k$.

Finally, we must bound the total cost of the multicut found by the algorithm Pipe-Cut. Consider some edge (u, v) for which $u \in V_j$, $v \in V_{j'}$, $j \leq j'$. The cost of this edge, $c(u, v)$, is included in the sum $C_x(s_{i(j)}, \rho_j)$. Thus, we can upper bound the cost of the multicut found by the algorithm by $\sum_j C_x(s_{i(j)}, \rho_j)$. By Lemma 5.2, for each iteration j we have that $C_x(s_{i(j)}, \rho_j) \leq 2\ln(2k) V_x(s_{i(j)}, \rho_j)$. If we consider the balls $B_x(s_{i(j)}, \rho_j)$ found in each iteration, they each have an associated volume that corresponds to disjoint portions of the original pipe system. The total volume in the original pipe system is at most 2ϕ: there were ϕ units of volume originally, and at most k point volumes, each of volume ϕ/k, that are added throughout the course of the algorithm. Hence, the cost of the multicut is no more than

$$\sum_j 2\ln(2k) V_x(s_{i(j)}, \rho_j) \leq 4\ln(2k)\phi,$$

and we have obtained the result of Garg, Vazirani, & Yannakakis [GVY93].

THEOREM 5.1 The algorithm Pipe-Cut, when applied to an optimal fractional multicut x^*, is a $4\ln(2k)$-approximation algorithm for the minimum multicut problem.

Since we have shown that there always exists a multicut of cost at most $4\ln(2k)$ times the cost of the optimal fraction multicut, we have also proved the following approximate max-flow min-cut theorem for multicommodity flow.

COROLLARY 5.1 For any instance with k terminal pairs, the cost of the minimum multicut is always at least the maximum multicommodity flow value, and is always at most $O(\log k)$ times as much.

5.3 SPARSEST CUTS AND MAXIMUM CONCURRENT FLOW

In this section we will present approximation algorithms for the sparsest cut problem for undirected graphs; these algorithms will also play a central role in the design of approximation algorithms for the balanced cut problem.

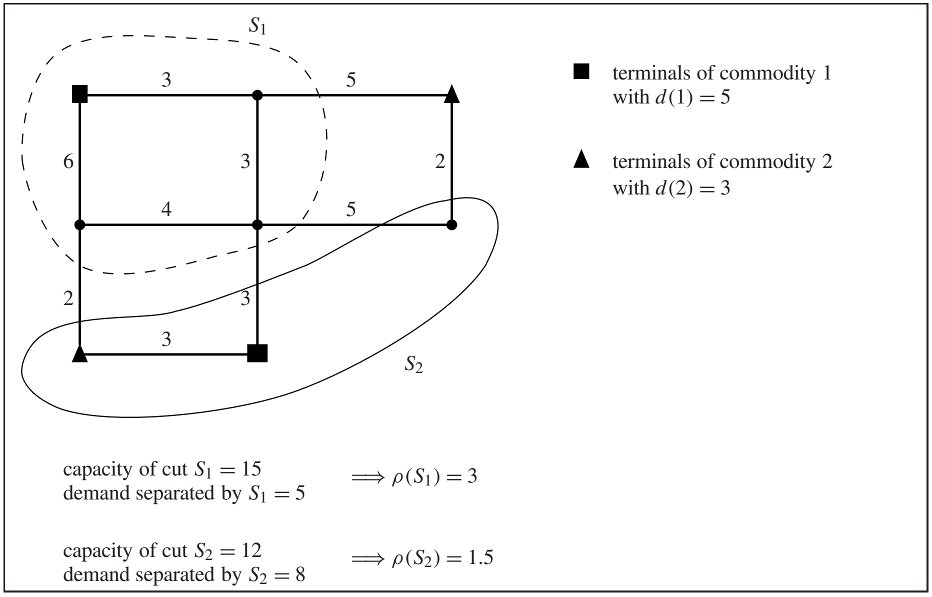

FIGURE 5.4

Illustrating the sparsity ratio of a cut

5.3.1 THE SPARSEST CUT PROBLEM

In the *sparsest cut problem*, we are given an undirected graph $G = (V, E)$, where there is a positive cost $c(e)$ associated with each edge $e \in E$, and k pairs of vertices (s_i, t_i), $i = 1, \dots, k$, where there is a positive demand $d(i)$ for the commodity corresponding to (s_i, t_i). For each subset of nodes $S \subseteq V$, let $I(S)$ indicate those terminal pairs that are disconnected by deleting the edges in $\delta(S) = \{(u, v) \in E : u \in S,\ v \notin S\}$; more formally, $I(S) = \{i : |S \cap \{s_i, t_i\}| = 1\}$. The *sparsity ratio* of the set S, $\rho(S)$, is the ratio of the total cost of the edges connecting S with $V - S$ to the total demand separated by this cut:

$$\rho(S) = \frac{\sum_{e \in \delta(S)} c(e)}{\sum_{i \in I(S)} d(i)}.$$

We wish to find the set S for which $\rho(S)$ is minimum. Observe that the definition of the sparsity ratio of a cut captures the fact that in some settings we might be most interested in “getting our money’s worth” in that we are willing to pay more for a cut if it succeeds in disconnecting a greater amount of the demand. Figure 5.4 illustrates these notions for a particular instance.

Although we have defined the sparsity ratio of a cut in terms of a subset of vertices, we can instead view a cut as a subset of edges whose deletion disconnects the graph. We can extend the notion of sparsity ratio to apply to a subset of edges F whose deletion leaves a graph with more than two connected components. Let $\mathcal{S} = \{S_1, S_2, \dots, S_c\}$ be a partition of V such that each S_j denotes the vertex set of a connected component in $\bar{G} = (V, E - F)$. Let $I(\mathcal{S})$ denote the set of terminal pairs that are disconnected by $\mathcal{S}$:

$\{i : s_i \in S_j,\ t_i \in S_k,\ j \neq k\}$. The sparsity ratio of $\mathcal{S}$ is then

$$\rho(\mathcal{S}) = \frac{\sum_{e \in F} c(e)}{\sum_{i \in I(\mathcal{S})} d(i)}.$$

The following proposition merely asserts that an average cannot be less than the minimum of its components.

PROPOSITION 5.1 For any non-negative integers, $a_1, \ldots, a_n$, and positive integers, $b_1, \ldots, b_n$, $\min_{i=1,\ldots,n} a_i/b_i \leq (\sum_{i=1}^n a_i)/(\sum_{i=1}^n b_i)$.

Applying this to our setting, we obtain the following lemma.

LEMMA 5.3 Let $\mathcal{S} = \{S_1, S_2, \ldots, S_c\}$ be a partition of V such that each S_j, $j = 1, \ldots, c$, denotes the vertex set of a connected component in $\bar{G} = (V, E - F)$. Then

$$\min_{j=1,\ldots,c} \rho(S_j) \leq \rho(\mathcal{S}).$$

This lemma implies that we can find a good set of edges to delete without concern for the number of connected components in the remaining graph (provided that it is at least two). Furthermore, it implies that we can obtain a simple (non-linear) integer programming formulation of the sparsest cut problem. We let $x(e)$, $e \in E$ be a 0-1 variable that indicates whether edge e is cut, and let $y(i)$, $i = 1, \ldots, k$, be a 0-1 variable that indicates whether commodity i is disconnected by this cut. As in the previous section, let $\mathcal{P}_i$ denote the set of paths between s_i and t_i, $i = 1, \ldots, k$.

$$\text{minimize} \quad \frac{\sum_{e \in E} c(e)x(e)}{\sum_{i=1}^k d(i)y(i)} \tag{5.14}$$

subject to

$$\sum_{e \in P} x(e) \geq y(i), \qquad \text{for each } P \in \mathcal{P}_i,\ i = 1, \ldots, k, \tag{5.15}$$

$$y(i) \in \{0, 1\}, \qquad \text{for each } i = 1, \ldots, k, \tag{5.16}$$

$$x(e) \in \{0, 1\}, \qquad \text{for each } e \in E. \tag{5.17}$$

Consider the relaxation of this formulation in which constraints (5.16) and (5.17) are replaced by $y(i) \geq 0$, $i = 1, \ldots, k$, and $x(e) \geq 0$, $e \in E$, respectively. First, observe that if (x, y) is a feasible fractional solution, then for any constant $\alpha > 0$, the solution $(\alpha x, \alpha y)$ is also feasible, and the two solutions have equal objective function value. Consequently, it is without loss of generality that we have not constrained each variable to be at most 1. Furthermore, we can choose a normalization of the variables by requiring that $\sum_{i=1}^k d(i)y(i) = 1$, and by doing this, we have obtained the following linear programming relaxation of the sparsest cut problem:

$$\text{minimize} \quad \sum_{e \in E} c(e)x(e) \tag{5.18}$$

subject to

$$\sum_{i=1}^{k} d(i)y(i) = 1, \tag{5.19}$$

$$\sum_{e \in P} x(e) \geq y(i), \qquad \text{for each } P \in \mathcal{P}_i,\ i = 1, \ldots, k, \tag{5.20}$$

$$y(i) \geq 0, \qquad \text{for each } i = 1, \ldots, k, \tag{5.21}$$

$$x(e) \geq 0, \qquad \text{for each } e \in E. \tag{5.22}$$

This linear program can be solved in polynomial time; we defer further discussion of this until Section 5.3.5

Rao [Rao87] focused attention on the sparsest cut problem, and showed its importance in computing a balanced cut (albeit only in the context of planar graphs). Leighton & Rao [LR88] considered arbitrary graphs, but restricted attention to the case in which there is a commodity for each pair of vertices, and each demand $d(i) = 1$. For this case, they showed how to construct a cut of sparsity ratio within a $O(\log n)$ factor of the optimal value of the linear relaxation. Klein, Rao, Agrawal & Ravi [KRAR95] considered the general case, and gave an $O(\log C \log D)$-approximation algorithm, where C denotes the sum of the edge costs in the graph and D denotes the total demand; Tragoudas [Tra91] subsequently improved this performance guarantee to $O(\log n \log D)$. Kahale [Kah93] showed how to obtain a bound of $O(\log k \log D)$ by showing, in effect, how to reduce the sparsest cut problem to the minimum multicut problem. Plotkin & Tardos [PT95] gave a sophisticated rounding technique that yields an improved performance guarantee of $O(\log^2 k)$. Finally, work of London, Linial, & Rabinovich [LLR95] and Aumann & Rabani[AR95] led to a much simpler $O(\log k)$-approximation algorithm for the general case, thereby matching the performance of Leighton & Rao for the all-pairs unit-demand problem.

5.3.2 REDUCING THE SPARSEST CUT PROBLEM TO THE MINIMUM MULTICUT PROBLEM

We will show how to use the information given by an optimal fractional solution to (5.18)-(5.22) to construct a sparse cut. We shall actually give two algorithms to do this. In this subsection, we will present the result of Kahale [Kah93], which relies on Theorem 5.1 to produce a cut of sparsity ratio within a factor of $O(\log D \log k)$ of the fractional optimum, where D denotes the total demand $\sum_{i=1}^{k} d(i)$.

The main idea of this algorithm is quite simple. We use the optimal fractional solution (x, y) to identify a particular subset S of the commodities and apply the approximation algorithm of Section 5.2 to find a multicut that separates each commodity in S. The choice of S will ensure that the multicut found is sparse, and hence, by Lemma 5.3, we find a sparse cut. Let $\bar{D} = \sum_{i \in S} d(i)$ and $y_{\min} = \min_{i \in S} y(i)$. We will choose S such that

$$y_{\min} \geq \frac{1}{\bar{D} \cdot \mathcal{H}(\bar{D})}, \tag{5.23}$$

where $\mathcal{H}(k) = 1 + 1/2 + 1/3 + \cdots + 1/k$ is the kth Harmonic number; it is straightforward to show that $\mathcal{H}(k) \le \log_e k + 1$. Suppose that we apply the multicut approximation algorithm given in Section 5.2 to the modified instance in which S specifies the terminal pairs that must be separated (and there are no specified demands). Given any feasible fractional solution (x, y) for the sparsest cut relaxation, (5.18)-(5.22), we can construct a solution $\bar{x}(e) = x(e)/y_{\min}$, for each $e \in E$. We shall argue that $\bar{x}$ is a feasible fractional multicut for the modified instance. To see this, observe that the constraints (5.20) imply that

$$\sum_{e \in P} \bar{x}(e) \ge 1, \qquad \text{for each } P \in \mathcal{P}_i,\ i \in S.$$

By (5.23), $\sum_e c(e)\bar{x}(e) \le \bar{D} \cdot \mathcal{H}(\bar{D}) \cdot (\sum_e c(e)x(e))$; since $\bar{x}$ is a feasible fractional multicut, the cost of the optimal fractional multicut is clearly at most this much. The multicut approximation algorithm of Garg, Vazirani, & Yannakakis produces a multicut $\mathcal{S}$ of cost within an $O(\log|S|)$ factor of the optimal fractional solution, and hence the cost of $\mathcal{S}$ is $O(\log k \cdot \bar{D} \cdot \mathcal{H}(\bar{D}) \cdot (\sum_e c(e)x(e)))$. Since $\mathcal{S}$ is a multicut that disconnects each terminal pair in S, $\sum_{i \in I(\mathcal{S})} d(i) \ge \sum_{i \in S} d(i) = \bar{D}$; hence $\rho(\mathcal{S})$ is $O(\log k \cdot \mathcal{H}(\bar{D})) \cdot \sum_e c(e)x(e)$. Since (x, y) is an optimal solution to the linear relaxation (5.18)-(5.22), we see that $\rho(\mathcal{S})$ is within $O(\log k \cdot \mathcal{H}(\bar{D}))$ factor of optimal.

We must still show how to construct the required set S that satisfies (5.23). Assume, without loss of generality, that the commodities are indexed such that $y(1) \ge y(2) \ge \cdots \ge y(k)$. Let $D(j) = \sum_{i=1}^{j} d(i)$. We shall show that there exists j^* such that $y(j^*) \ge 1/(D(j^*) \cdot \mathcal{H}(D))$ and hence $S = \{1, 2, \ldots, j^*\}$ satisfies (5.23). Suppose, for a contradiction, that

$$y(i) < \frac{1}{D(i)\mathcal{H}(D)}, \quad \text{for each } i = 1, \ldots, k.$$

This would imply that

$$\begin{aligned}
\sum_{i=1}^{k} d(i)y(i) &< \sum_{i=1}^{k} \frac{d(i)}{\mathcal{H}(D) \cdot D(i)} \\
&= \frac{1}{\mathcal{H}(D)} \sum_{i=1}^{k} \frac{D(i) - D(i-1)}{D(i)} \\
&\le \frac{1}{\mathcal{H}(D)} \sum_{i=1}^{k} \sum_{s=D(i-1)+1}^{D(i)} \frac{1}{s} \\
&= 1,
\end{aligned}$$

which contradicts (5.19).

5.3.3 EMBEDDINGS AND THE SPARSEST CUT PROBLEM

In this section, we give an improved rounding method due to Linial, London, & Rabinovich [LLR95] and Aumann & Rabani [AR95]; this yields an $O(\log k)$-approximation algorithm for the sparsest cut problem. In Section 5.2, we viewed the fractional multicut x as specifying a length function on the edges. We can similarly interpret the variables x in the linear programming relaxation of the sparsest cut problem; the variable $y(i)$ correspondingly represents the distance between s_i and t_i, $i = 1, \ldots, k$. Suppose that this length function were derived from an embedding of the input graph $G = (V, E)$ in some low-dimensional space with L_1 norm: that is, there is a function $f : V \to \Re^d$ such that $x(e) = \|f(u) - f(v)\|_1$, for each edge $e = (u, v) \in E$, and $y(i) = \|f(s_i) - f(t_i)\|_1$ for each $i = 1, \ldots, k$. We first shall show if such an embedding were achievable, then we could find a cut of sparsity ratio at most $\sum_e c(e)x(e)$; that is, if the optimal fractional solution (x, y) has the property that it can be derived from such an embedding, then, given the embedding, we can find an optimal sparsest cut.

It is, of course, too much to hope for that the optimal length function admits such an embedding. The crux of the approximation algorithm for the sparsest cut problem is an algorithm to find an embedding that roughly corresponds to the distances. The embedding algorithm that we shall give is due to Linial, London, & Rabinovich [LLR95], which is based on a result of Bourgain [Bou85], which gives an exponential algorithm to find an embedding. The application of this embedding algorithm to the sparsest cut problem was discovered by Aumann & Rabani [AR95], based on a preliminary version of [LLR95], and an essentially identical result was independently discovered by Linial, London, & Rabinovich [LLR95], and published together with their embedding algorithm.

LEMMA 5.4 Let (x, y) be a feasible solution to the linear program (5.18)-(5.22) of objective function value α such that there exists a function $f : V \to \Re^d$ with the property that

$$x(e) = \|f(u) - f(v)\|_1, \quad \text{for each } e = (u, v) \in E, \tag{5.24}$$

and

$$y(i) = \|f(s_i) - f(t_i)\|_1, \quad \text{for each } i = 1, \ldots, k. \tag{5.25}$$

Then, given f, one can find a cut S of sparsity ratio $\rho(S)$ at most α in polynomial time.

Proof. We first try to understand some simple consequences of the existence of the embedding f. This embedding defines a metric μ on the set of vertices V: that is, for each pair of vertices $u, v \in V$, we have a value $\mu(u, v) = \|f(u) - f(v)\|_1$, and these values satisfy the triangle inequality: for each $u, v, w \in V$, $\mu(u, v) + \mu(v, w) \geq \mu(u, w)$. (If $e = (u, v) \in E$, then we shall also use $\mu(e)$ to denote $\mu(u, v)$.) We can also derive a metric corresponding to each subset $S \subseteq V$: let $\mu_S(u, v)$ be 1 if $|\{u, v\} \cup S| = 1$, and 0 otherwise. To see that this defines a metric, observe that for any $u, v, w \in V$, if (u, w) crosses the cut defined by S, then exactly one of (u, v) and (v, w) must cross this cut too.

The crucial property of μ that we shall use is that it is in the cone defined by the cut metrics μ_S: that is, there exists $\lambda_S \geq 0$ such that

$$\mu(u, v) = \sum_{S \subseteq V} \lambda_S \cdot \mu_S(u, v), \quad \text{for each } u, v \in V. \tag{5.26}$$

To show this, we start by considering just the contribution of the first coordinate of our d-dimensional space to μ, $|f_1(u) - f_1(v)|$. Suppose that the vertices of V are indexed

$v_1, \ldots, v_n$ such that

$$f_1(v_1) \le f_1(v_2) \le \cdots \le f_1(v_n).$$

Then, for each pair of vertices $v_j, v_{j'}$, $j > j'$, we can rewrite the contribution of the first coordinate to $\mu(v_j, v_{j'})$ as

$$f_1(v_j) - f_1(v_{j'}) = \sum_{\ell=j'}^{j-1} f_1(v_{\ell+1}) - f_1(v_\ell).$$

Let $S(\ell) = \{v_1, \ldots, v_\ell\}$, $\ell = 1, \ldots, n$. Note that the term $f_1(v_{\ell+1}) - f_1(v_\ell)$ is included in this sum precisely when the distance, with respect to the metric $\mu_{S(\ell)}$, between v_j and $v_{j'}$ is 1. In other words,

$$f_1(v_j) - f_1(v_{j'}) = \sum_{\ell=1}^{n} (f_1(v_{\ell+1}) - f_1(v_\ell)) \cdot \mu_{S(\ell)}(v_j, v_{j'}).$$

Observe that if $j < j'$, then the right-hand side of this equation still gives $|f_1(v_j) - f_1(v_{j'})|$. Hence, we have shown that the contribution of the first coordinate to μ can be expressed in the claimed form. By repeating this construction for each coordinate and adding them together, we have produced a decomposition of μ satisfying (5.26).

Note that although there are 2^n sets $S \subseteq V$, we have produced a decomposition that uses at most nd sets. Second, we have given an efficient algorithm to produce this decomposition. Finally, it is not hard to show that (5.26) precisely characterizes those metrics that correspond to an embedding in L_1; they are precisely those metrics that lie in the cone defined by the cut metrics, or more simply the *cut cone*. For much more information the cut cone, and its relationship to embeddings and multicommodity flows, the reader is referred to the expository paper of Avis and Deza [AvisD91].

However, we shall now see that (5.26) easily implies the lemma. Let $\mu = \sum_{S \in \mathcal{S}} \lambda_S \mu_S$, as in (5.26), where $\lambda_S > 0$ for each $S \in \mathcal{S}$. But then, by substituting $\mu(e)$ for $x(e)$, $\mu(s_i, t_i)$ for $y(i)$, and recalling that (x, y) is feasible, we see that

$$\alpha = \sum_{e \in E} c(e)x(e) = \frac{\sum_{e \in E} c(e) \sum_{S \in \mathcal{S}} \lambda_S \cdot \mu_S(e)}{\sum_{i=1}^{k} d(i) \sum_{S \in \mathcal{S}} \lambda_S \cdot \mu_S(s_i, t_i)} = \frac{\sum_{S \in \mathcal{S}} \lambda_S \sum_{e \in E} c(e)\mu_S(e)}{\sum_{S \in \mathcal{S}} \lambda_S \sum_{i=1}^{k} d(i)\mu_S(s_i, t_i)}.$$

Using the fact that $\mu(S)(u, v) = 1$ exactly when (u, v) crosses the cut defined by S, we see that

$$\alpha = \frac{\sum_{S \in \mathcal{S}} \lambda_S \sum_{e \in \delta(S)} c(e)}{\sum_{S \in \mathcal{S}} \lambda_S \sum_{i \in I(S)} d(i)} \ge \min_{S \in \mathcal{S}} \frac{\sum_{e \in \delta(S)} c(e)}{\sum_{i \in I(S)} d(i)} = \min_{S \in \mathcal{S}} \rho(S),$$

where the inequality follows from Proposition 5.1. Hence, we have shown that one of the cuts $S \in \mathcal{S}$ has sparsity ratio at most α.

The algorithm Embed-Cut, given in Figure 5.5, summarizes this quite trivial procedure to find a good cut, given an embedding. ■

Finally, it is not hard show that if there are only two commodities, then there exists an optimal length function (x^*, y^*) for which a suitable embedding f is easy to construct; by applying Lemma 5.4, one obtains the min-max theorem of Hu [Hu63].

5.3.4 FINDING A GOOD EMBEDDING

We can view equations (5.24) and (5.25) as taking an embedding f, and from the embedding constructing a fractional solution (x, y). This solution is not necessarily feasible, since although (x, y) satisfies constraints (5.20), it need not satisfy (5.19). However, this is not significant, since this is just a question of normalization: if $\sum_i d(i)y(i) = \beta$, then $(x/\beta, y/\beta)$ is a feasible fractional solution; we shall refer to this solution as the one *induced by the embedding f*.

The $O(\log k)$-approximation algorithm for the sparsest cut problem is an immediate corollary of the following result: given any feasible solution (x, y) to (5.19)-(5.22), we can construct an embedding f which induces a feasible fractional solution $(\bar{x}, \bar{y})$ such that $\sum_{e\in E} c(e)\bar{x}(e)$ is $O(\log k)\sum_{e\in E} c(e)x(e)$. The embedding is constructed by a randomized algorithm: that is, for any input, the solution induced by the embedding is sufficiently good with high probability, where the probability depends only on the random choices made by the algorithm, and not on any (probabilistic) assumption about the input. Randomized approximation algorithms are discussed in much greater detail in Chapter 11 of this volume, and we shall rely on basic tools of analysis that are developed in that chapter. Alternatively, the reader can consult the recent book of Motwani & Raghavan [MR95] for an even more thorough investigation of this area.

Let T denote the set of terminal vertices $\{s_i, t_i : i = 1, \dots, k\}$; for simplicity of notation, we shall assume that $|T|$ is a power of 2, i.e., $|T| = 2^\tau$. We shall use the notation $dist_x(u, v)$ to denote the length of the shortest path between vertices u and v with respect to the length function x. Furthermore, for any set of vertices $A \subseteq V$, let $dist_x(u, A) = \min_{v\in A} dist_x(u, v)$.

Given a feasible fractional solution (x, y), our embedding will be quite easy to compute. We shall let $L = q\log k$ where q is a constant that will be determined later. The dimension of the embedding is $d = \tau L = O(\log^2 k)$. For $\ell = 1, \dots, L$, $t = 1, \dots, \tau$, construct the set $A_{t\ell}$ by choosing $2^{\tau-t} = k/2^t$ points from T uniformly at random with replacement (that is, a given element of T may be selected more than once). The embedding function f is then defined to be

$$f_{t\ell}(v) = dist_x(v, A_{t\ell}), \quad \text{for each } v \in V. \tag{5.27}$$

We first state the two key lemmas, show how they imply that the embedding is good, and then give their proofs.

LEMMA 5.5 For each edge $e = (u, v)$, $\|f(u) - f(v)\|_1 \le d \cdot x(e)$.

```
procedure Embed-Cut(G, f)
    { Each vertex v ∈ V(G) is embedded at (f_1(v), ..., f_d(v)) }
    (j*, ξ*) ← argmin{ρ(S(j, ξ)) : ξ = f_j(v), v ∈ V(G), j = 1, ..., d};
    output S(j*, ξ*) = {v : f_{j*}(v) ≤ ξ*}.
```

FIGURE 5.5

The algorithm Embed-Cut

LEMMA 5.6 With probability at least $1/2$,

$$\|f(s_i) - f(t_i)\|_1 \geq L \cdot y(i)/88, \quad \text{for each } i = 1, \ldots, k. \tag{5.28}$$

Since it is easy to check if a particular embedding f satisfies (5.28), we can arbitrarily increase the probability of success by repeatedly constructing independent embeddings. Furthermore, by iterating until an embedding satisfying (5.28) is found, we can obtain a Las Vegas-style algorithm, where the performance is guaranteed, and the expected running time is polynomial.

We can combine Lemmas 5.5 and 5.6 in the obvious way. We focus on the case in which f satisfies the property (5.28). In this case,

$$\sum_{i=1}^{k} d(i)\|f(s_i) - f(t_i)\|_1 = \Omega(\log k \sum_{i=1}^{k} d(i)y(i)) = \Omega(\log k). \tag{5.29}$$

On the other hand, Lemma 5.5 implies that

$$\sum_{(u,v)=e\in E} c(e)\|f(u) - f(v)\|_1 = O(\log^2 k)\sum_{e\in E} c(e)x(e). \tag{5.30}$$

Combining these two equations, we get the following theorem.

THEOREM 5.2 With probability at least $1/2$, the embedding f induces a feasible fractional solution $(\bar{x}, \bar{y})$ of objective function value

$$\sum_{e\in E} c(e)\bar{x}(e) = O(\log k)\sum_{e\in E} c(e)x(e).$$

By applying this theorem starting with a optimal solution (x, y) to the linear program (5.18)-(5.22), we obtain the following corollary.

COROLLARY 5.2 There is a randomized polynomial-time algorithm that, with probability at least 1/2, computes a cut of sparsity ratio within an $O(\log k)$ factor of optimal.

As we observed above, it is possible to verify efficiently if the embedding found is sufficiently good, and hence we can obtain an improved algorithm for which the performance guarantee holds with high probability; furthermore, by running the embedding algorithm until a good one is found, we obtain a randomized algorithm that runs in polynomial expected time, and is guaranteed to produce a cut of sparsity ratio within an $O(\log k)$ factor of optimal.

Linial, London, & Rabinovich and, independently, Garg [Gar95] have subsequently shown that, like many other randomized algorithms whose analysis is based on Chernoff-type calculations, this algorithm can be derandomized. However, this deterministic version does not simply follow from an application of standard derandomization techniques to this algorithm; we shall omit the details.

THEOREM 5.3 There is a deterministic polynomial-time $O(\log k)$-approximation algorithm for the sparsest cut problem.

We now return to the proofs of Lemmas 5.5 and 5.6.

Proof of Lemma 5.5. This is quite straightforward to show. For any set $A \subseteq V$ and edge $(u, v) = e \in E$,

$$dist_x(u, A) \leq x(e) + dist_x(v, A).$$

Equivalently, we have that

$$dist_x(u, A) - dist_x(v, A) \leq x(e).$$

By the same argument, we see that

$$dist_x(v, A) - dist_x(u, A) \leq x(e).$$

By definition,

$$\|f(u) - f(v)\|_1 = \sum_{t,\ell} |f_{t\ell}(u) - f_{t\ell}(v)| = \sum_{t,\ell} |dist_x(u, A_{t\ell}) - dist_x(v, A_{t\ell})|.$$

Combining this with the inequalities above, we see that

$$\|f(u) - f(v)\|_1 \leq \tau L x(e) = d \cdot x(e).$$

■

Proof of Lemma 5.6. This is the crucial observation that drives the embedding-based proof of Bourgain [Bou85] and Linial, London, & Rabinovich [LLR95], and its proof is rather involved.

We first focus on one particular commodity i, and prove that with probability at least $1 - (1/2k)$, $\|f(s_i) - f(t_i)\|_1 = \Omega(L \cdot y(i))$. Before proving this, observe that this claim implies the lemma: the probability that property (5.28) fails to hold is at most k times the probability that it fails for one particular commodity.

For $v \in \{s_i, t_i\}$, let

$$B_x(v, r) = \{w \in T : dist_x(v, w) \leq r\},$$

and let

$$B_x^\circ(v, r) = \{w \in T : dist_x(v, w) < r\}.$$

Let $r_0 = 0$, and let r_t be the smallest value of r such that $|B_x(v, r)| \geq 2^t$, for both $v \in \{s_i, t_i\}$. Furthermore, we let $\hat{t}$ denote the smallest value of t for which $r_{\hat{t}} \geq y(i)/4$, and reset $r_{\hat{t}} = y(i)/4$. Observe that since $y(i) \leq dist_x(s_i, t_i)$ (by constraints (5.20)), we have ensured that $B(s_i, r_{\hat{t}})$ and $B(t_i, r_{\hat{t}})$ are disjoint.

To give some intuition of the proof, we wish to show that f embeds s_i and t_i so that they are reasonably far apart as compared to $y(i)$. Since we are computing distances in the L_1 norm, we can prove this by showing that, with sufficiently high probability, each coordinate of f contributes a certain distance to the total distance between s_i and t_i. More precisely, we will show that each of the coordinates $f_{t\ell}$ is likely to contribute a distance $r_t - r_{t-1}$. By summing this over the L choices for ℓ, we get that those coordinates are likely to contribute $\Omega(L(r_t - r_{t-1}))$. By summing this bound for $t = 1, \ldots, \hat{t}$, we get that this sum is $\Omega(Lr_{\hat{t}}) = \Omega(Ly(i))$, which is what we wish to prove.

Observe that $A \cap B_x^\circ(s_i, r_t) = \emptyset$ if and only if $dist_x(s_i, A) \geq r_t$; alternatively, we have that $A \cap B_x(t_i, r_{t-1}) \neq \emptyset$ if and only if $dist_x(t_i, A) \leq r_{t-1}$. Thus, if we let $E_{t\ell}$, $t = 1, \ldots, \hat{t}$,

$\ell = 1, \ldots, L$, denote the event that

$$A_{t\ell} \cap B_x^\circ(s_i, r_t) = \emptyset \text{ and } A_{t\ell} \cap B_x(t_i, r_{t-1}) \neq \emptyset,$$

then $E_{t\ell}$ implies that

$$|f_{t\ell}(s_i) - f_{t\ell}(t_i)| = |dist_x(s_i, A_{t\ell}) - dist_x(t_i, A_{t\ell})| \geq r_t - r_{t-1}.$$

Hence, we are interested in showing that $E_{t\ell}$ is likely to occur. In order to bound this probability, we shall first do some preliminary calculations. Suppose that we are given a ground set X in which there is a specified good subset G and disjoint bad subset B; finally, A is formed by selecting p elements of X independently and uniformly at random from X (with replacement). Then

$$\begin{aligned} Pr[A \cap G \neq \emptyset \text{ and } A \cap B = \emptyset] &= Pr[A \cap G \neq \emptyset \mid A \cap B = \emptyset] \cdot Pr[A \cap B = \emptyset] \\ &\geq Pr[A \cap G \neq \emptyset] \cdot Pr[A \cap B = \emptyset], \end{aligned}$$

where the last inequality follows from the observation that knowing that the elements of A are not selected from B can only increase the likelihood that they are selected from G. For any subset $Y \subseteq X$, the probability that $A \cap Y = \emptyset$ is $(1 - \frac{|Y|}{|X|})^p$. If $p = |X|/|Y|$, then this bound approaches $1/e$ as p tends to infinity, and is always within the interval $[1/4, 1/e]$ (provided $|X|/|Y| > 1$). More generally, if $p = \beta(|X|/|Y|)$, then the probability that $A \cap Y = \emptyset$ is within the interval $[(1/4)^\beta, (1/e)^\beta]$.

Now consider the event $E_{t\ell}$, $t = 1, \ldots, \hat{t}$, $\ell = 1, \ldots, L$. We can assume without loss of generality that it was the ball around s_i that determined the radius r_t (since otherwise we could interchange s_i and t_i). We can apply the previous probability calculations by setting $A = A_{t\ell}$, $X = T$, $B = B_x^\circ(s_i, r_t)$ and $G = B_x(t_i, r_{t-1})$. We have that $p = 2^{\tau - t}$, $|X| = 2^\tau$, $|B| < 2^t$ and $|G| \geq 2^{t-1}$; hence, $p < |X|/|B|$ and $p \geq (1/2)|X|/|G|$. This implies that $Pr[A \cap B = \emptyset] \geq 1/4$ and $Pr[A \cap G \neq \emptyset] \geq (1 - (1/e)^{1/2})$. Hence, $Pr[E_{t\ell}] \geq (1 - (1/e)^{1/2})/4 \geq 1/11$, $t = 1, \ldots, \hat{t}$, $\ell = 1, \ldots, L$. (It is important to note that the above calculation applies verbatim to the seemingly exceptional case when $t = \hat{t}$.)

If we fix a particular $t = 1, \ldots, \hat{t}$, and define X_ℓ, $\ell = 1, \ldots, L$, to be the 0-1 variable that indicates whether (or not) event $E_{t\ell}$ occurs, then we can apply the Chernoff bound to show that $\sum_{\ell=1}^{L} X_\ell$ does not deviate too much from its expectation, which is at least $L/11$. We shall apply this bound in a very crude way, and will pay no attention to optimizing the constants involved. In particular, we shall use the following statement of the Chernoff bound: if $E[\sum_\ell X_\ell] = \mu$, then

$$Pr[\textstyle\sum_\ell X_\ell < \mu/2] \leq \exp(-\mu/8).$$

(This follows, for example, from Theorem 4.2 of [MR95].) Since $\mu \geq L/11 = q \log k/11$, we see that if we set q to be, say, 200, then this probability is less than $\frac{1}{(2k)\log(2k)}$. Most importantly, if $\sum_\ell X_\ell \geq L/22$, then we know that for $L/22$ of the components $f_{t\ell}$, $\ell = 1, \ldots, L$, the event $E_{t\ell}$ occurs, and so

$$\sum_{\ell=1}^{L} |f_{t\ell}(s_i) - f_{t\ell}(t_i)| \geq (r_t - r_{t-1})L/22. \tag{5.31}$$

We have shown that for any fixed value of $t = 1, \ldots, \hat{t}$, (5.31) fails to hold with probability at most $1/(2k\log(2k))$. Since $\hat{t} \leq \log(2k)$, it follows that (5.31) holds for *every* $t = 1, \ldots, \hat{t}$, with probability at least $1 - (1/2k)$. But this implies that, with probability at least $1-(1/2k)$,

$$\sum_{t=1}^{\hat{t}} \sum_{\ell=1}^{L} |f_{t\ell}(s_i) - f_{t\ell}(t_i)| \geq \sum_{t=1}^{\hat{t}} (r_t - r_{t-1})L/22 = r_{\hat{t}}L/22 = y(i)L/88. \tag{5.32}$$

This completes the proof of Lemma 5.6. ■

5.3.5 THE MAXIMUM CONCURRENT FLOW PROBLEM

We have just seen that one can compute a cut S of sparsity ratio $\rho(S)$ within a factor of $O(\log k)$ of the optimal fractional solution to (5.18)-(5.22). By considering the linear programming dual to this, we can obtain an elegant approximate max-flow-min-cut theorem for multicommodity flow.

The dual of (5.18)-(5.22) is as follows:

$$\text{maximize} \qquad \alpha \tag{5.33}$$

subject to

$$\sum_{P \in \mathcal{P}_i} f(P) \geq \alpha d(i), \qquad \text{for each } i = 1, \ldots, k, \tag{5.34}$$

$$\sum_{i=1}^{k} \sum_{P \in \mathcal{P}_i(e)} f(P) \leq c(e), \qquad \text{for each } e \in E, \tag{5.35}$$

$$f(P) \geq 0, \qquad \text{for each } P \in \mathcal{P}_i, i = 1, \ldots, k. \tag{5.36}$$

This problem is called the *maximum concurrent flow*; it is the version of the multicommodity flow problem in which there is a given demand $d(i)$ for each commodity $i = 1, \ldots, k$, and one wishes to maximize the fraction α such that a α fraction of each commodity's demand is met, subject to the joint capacity constraints of the network.

It is straightforward to see that the maximum concurrent flow value, α^* is at most $\rho(S)$, for any cut S, and hence at most the minimum sparsity ratio: if the total cost (or equivalently, capacity) of S is $\bar{C}$, and it disconnects terminal pairs of total demand $\bar{D}$, then if one just focuses on these commodities, the maximum fraction of the demand that can be met is at most $\bar{C}/\bar{D} = \rho(S)$. Since LP duality implies that the maximum concurrent flow value is exactly equal to the optimal value of the linear relaxation (5.18)-(5.22), we can reinterpret Theorem 5.2 as the following approximate max-flow-min-cut theorem.

THEOREM 5.4 For any instance with k terminal pairs, the minimum sparsity ratio is always at least the maximum concurrent flow value, and is always at most $O(\log k)$ times as much.

Throughout this section, we have ignored the question of computing an optimal solution to the linear relaxation (5.18)-(5.22) and its dual, the maximum concurrent flow problem, (5.33)- (5.36). As for the maximum multicommodity flow problem, there are three distinct options. First, one can give polynomial-sized LP reformulations. Second, one can apply the ellipsoid algorithm. And finally, we will see that the maximum concurrent flow problem can be easily formulated as fractional packing problem.

To give a fractional packing formulation of this problem, we first make a simple change of variables, where maximizing α is replaced by minimizing a variable λ, which corresponds to $1/\alpha$. This results in the following equivalent linear program.

$$\text{minimize} \qquad \lambda \tag{5.37}$$

subject to

$$\sum_{P \in \mathcal{P}_i} f(P) = d(i), \qquad \text{for each } i = 1, \ldots, k, \tag{5.38}$$

$$\sum_{i=1}^{k} \sum_{P \in \mathcal{P}_i(e)} f(P) \leq \lambda c(e), \qquad \text{for each } e \in E, \tag{5.39}$$

$$f(P) \geq 0, \qquad \text{for each } P \in \mathcal{P}_i,\ i = 1, \ldots, k. \tag{5.40}$$

The most natural way to formulate this as a fractional packing problem is to let Q denote the set of flows f satisfying the demand constraints (5.38) and the non-negativity constraints (5.40), and to let the packing constraints $Az \leq \lambda b$ correspond to the capacity constraints (5.39). However, one obtains a more efficient algorithm by also adding to Q the constraints that each commodity alone must not violate the given capacity constraint.

$$\sum_{P \in \mathcal{P}_i(e)} f(P) \leq c(e), \quad \text{for each } e \in E,\ i = 1, \ldots, k. \tag{5.41}$$

It is easy to see that, even with these additional constraints, the two linear programs (5.33)-(5.36) and (5.37)-(5.41) are equivalent. However, by strengthening Q in this way, the width of the formulation can now be bounded by k, since for each $f \in Q$, the total flow on each edge e can be at most $kc(e)$. Furthermore, the required subroutine to optimize over Q is still relatively easy: it requires k (single-commodity) maximum flow computations. By incorporating randomization as first proposed by Klein, Plotkin, Stein, & Tardos [KPST94], the effort for this step can be reduced to, in effect, a single max flow computation. This algorithm for the maximum concurrent flow problem is due to Leighton, Makedon, Plotkin, Stein, Tardos, & Tragoudas [LMP+95]. Finally, the dual variables maintained by this algorithm can be easily adapted to yield the required near-optimal solution for (5.18)-(5.22).

5.4 MINIMUM FEEDBACK ARC SETS AND RELATED PROBLEMS

The techniques described in the previous two sections were limited to undirected graphs. Leighton & Rao [LR88] also considered a rather special case in which the graphs are directed: they were able to extend Theorem 5.4, which bounds the ratio between the optimal values to the maximum concurrent flow and the sparsest cut problems, to the case in which there is a unit-demand commodity between each (ordered) pair of vertices. One of the primary applications of this generalization is an $O(\log^2 n)$-approximation algorithm for the minimum-cost feedback arc set in directed graphs. This was improved by Seymour [Sey95], who gave an $O(\log n \log\log n)$-approximation algorithm. His algorithm is quite similar in spirit to the techniques that we have described in the previous two sections, and hence we shall give his result, even though it does not explicitly rely on any connection to multicommodity flow problems. We shall present a more intuitive description of Seymour's result that is due to [ENRS95]. Even, Naor, Rao, & Schieber [ENRS95] have also extended this approach to yield similarly improved performance guarantees for a number of other problems.

5.4.1 AN LP-BASED APPROXIMATION ALGORITHM

A *feedback arc set* in a directed graph $G = (V, A)$ is a subset of arcs $F \subseteq A$, such that if all arcs in F are deleted, the remaining graph $(V, A - F)$ is acyclic. Equivalently, we can require that for each cycle C in G, at least one arc of C is contained in F. In the minimum-cost feedback arc set problem, we are given a directed graph $G = (V, E)$, a cost $c(a)$ for each arc $a \in A$, and we wish to find a feedback arc set F of minimum total cost $\sum_{a\in F} c(a)$.

It is straightforward to show that the minimum-cost feedback arc set problem in directed graphs is equivalent to the minimum-cost feedback vertex set problem in directed graphs (where the aim is to delete a cheap set of vertices so that no cycle remains), in that any performance guarantee obtained for one problem would translate into the same guarantee for the other. Hence, we see that there is a dramatic difference between our current knowledge for the minimum-cost feedback vertex set problem in undirected and directed graphs (see Chapter 9).

We are given a directed graph $G = (V, A)$, and a cost $c(a)$ for each arc $a \in A$. We shall assume, without loss of generality, that each cost $c(a)$ is a positive integer. Let $\mathcal{C}$ denote the set of all cycles in G. We can formulate the minimum-cost feedback arc set problem as the following integer programming problem:

$$\text{minimize} \qquad \sum_{a\in A} c(a)x(a) \tag{5.42}$$

subject to

$$\sum_{a\in C} x(a) \geq 1, \qquad \text{for each } C \in \mathcal{C}, \tag{5.43}$$

$$x(a) \in \{0, 1\}, \qquad \text{for each } a \in A. \tag{5.44}$$

In its linear relaxation, we replace (5.44) with

$$x(a) \geq 0, \qquad \text{for each } a \in A. \tag{5.45}$$

Seymour's algorithm, which we call Feedback, has the following outline: solve the linear relaxation (5.42), (5.43), & (5.45) and then interpret the optimal fractional solution x^* as a length function for the arcs; use the length function to compute a partition of the nodes (V_1, V_2) (in a way to be specified later), delete all arcs from V_1 to V_2 or vice versa (whichever is cheaper), and recurse on the graphs induced by V_1 and V_2. Of course, in order to initiate this approach, one must observe that the linear program can be solved in polynomial time by the ellipsoid algorithm, since the separation problem required is essentially an all-pairs shortest path computation [GLS88]. Alternatively, one can obtain a more efficient algorithm by formulating the problem as a fractional covering problem, and then applying the algorithm of Plotkin, Shmoys, & Tardos [PST95].

From this outline, it is clear that the key to the performance bound is the way in which the graph is partitioned. Let ϕ denote the value of the optimal fractional solution x^*. Furthermore, for a subset of vertices S, let $G[S]$ denote the subgraph of G induced by S; that is, the graph with vertex set S and arc set $A[S] = \{(u,v) : (u,v) \in A,\ u, v \in S.\}$. Finally, let $\delta^+(S) = \{(u,v) : (u,v) \in A,\ u \in S,\ v \in \bar{S}\}$, and analogously, let $\delta^-(S) = \{(v,u) : (v,u) \in A,\ u \in S,\ v \in \bar{S}\}$.

LEMMA 5.7 For a given strongly connected directed graph $G = (V, A)$, suppose that there exists a feasible solution x to (5.43) and (5.45) of objective function value ϕ. There exists a partition $(S, \bar{S})$ of V such that, for some ϵ, $0 < \epsilon < 1$, the following three conditions hold:

$$\sum_{a \in A[S]} c(a)x(a) \leq \epsilon\phi; \tag{5.46}$$

$$\sum_{a \in A[\bar{S}]} c(a)x(a) \leq (1-\epsilon)\phi; \tag{5.47}$$

and either

$$\sum_{a \in \delta^+(S)} c(a) \leq 20\epsilon\phi\log(1/\epsilon)\log\log\phi \tag{5.48}$$

or

$$\sum_{a \in \delta^-(S)} c(a) \leq 20\epsilon\phi\log(1/\epsilon)\log\log\phi. \tag{5.49}$$

Furthermore, this partition can be found in polynomial time.

We will first show that Lemma 5.7 suffices to show that Feedback has the claimed performance guarantee, and then give its proof. We should note there is some slack in the constant given in Lemma 5.7 (that is, 20), and that we are not attempting to give the best constant that is obtainable via this line of proof.

5.4.2 ANALYZING THE ALGORITHM FEEDBACK

We next show that Feedback is an $O(\log n \log\log n)$-approximation algorithm for the minimum-cost feedback arc set in directed graphs (see Figure 5.6). In computing a feedback arc set, a natural first step is to partition the graph into its strongly connected components. Since each cycle is contained in some strongly connected component, we can solve each component separately, and obtain a solution for the original graph by taking the union of the solutions found for the components. Of course, one need only consider the non-trivial strongly connected components; for example, if a graph is acyclic, each strongly connected component consists of a single vertex, and the empty set is a feedback

```
function Feedback(G, x)
  if G is acyclic then return ∅;
  else
    find strongly connected components 𝒞 of G;
    for each non-trivial strongly connected component C ∈ 𝒞
      S ←Feedback-Cut(C, x)
      if ∑_{e∈δ+(S)} c(e) ≤ ∑_{e∈δ−(S)} c(e)
        then F ← δ+(S)
        else F ← δ−(S);
      let G1 and G2 be the graphs induced by S and V(G) − S;
      return F∪Feedback(G1, x)∪Feedback(G2, x)
```

FIGURE 5.6

The algorithm Feedback

arc set.

We can view the input to the algorithm Feedback as the graph G, the cost function c, and a feasible solution x to the linear relaxation (5.43) and (5.45). First, we compute the strongly connected components of G. For each non-trivial component, we apply Lemma 5.7 to partition its vertex set into S and $\bar{S}$. Either the arcs from S to $\bar{S}$ are added to the feedback arc set, or those from $\bar{S}$ to S, whichever is cheaper. We apply the algorithm recursively to each of $G[S]$ and $G[\bar{S}]$; it is useful to note that, for any subgraph of G, the restriction of x to its arcs is a feasible solution to its linear relaxation. The recursion stops when the input is acyclic. It is clear that the algorithm finds a feasible feedback arc set.

As observed above, we can assume without loss of generality that the input is strongly connected. Let $F(\phi)$ denote the maximum value of the cost of a solution produced by the algorithm when given a graph and a feasible fractional solution of cost at most ϕ. By Lemma 5.7, we see that the function F obeys the following recurrence relation:

$$\begin{aligned}
&F(\phi) \leq \max_{0<\epsilon<1} \{F(\epsilon\phi) + F((1-\epsilon)\phi) + 20\epsilon\phi \log(1/\epsilon) \log\log\phi\};\\
&F(\phi) = 0 \text{ if } \phi < 1.
\end{aligned}$$

We shall prove by induction on ϕ that this implies that

$$F(\phi) \leq 20\phi \log\phi \log\log\phi. \tag{5.50}$$

The claim is clearly true for $\phi = 0$, since the existence of a feasible fractional solution of cost 0 implies that the graph is acyclic. Let $\bar{\epsilon} = 1 - \epsilon$. Since $\epsilon\phi$ and $\bar{\epsilon}\phi$ are both less than ϕ, we apply the inductive hypothesis to the recurrence relation to get that

$$\begin{aligned}
F(\phi) &\leq \max_{0<\epsilon<1} \{20\epsilon\phi \log(\epsilon\phi) \log\log(\epsilon\phi) + \\
&\qquad\qquad 20\bar{\epsilon}\phi \log(\bar{\epsilon}\phi) \log\log(\bar{\epsilon}\phi) + 20\epsilon\phi \log(1/\epsilon) \log\log\phi\} \\
&\leq \max_{0<\epsilon<1} \{20\epsilon\phi (\log\phi - \log(1/\epsilon)) \log\log\phi + \\
&\qquad\qquad 20\bar{\epsilon}\phi \log\phi \log\log\phi + 20\epsilon\phi \log(1/\epsilon) \log\log\phi\} \\
&= 20\phi \log\phi \log\log\phi
\end{aligned}$$

If we apply the algorithm with an optimal fractional solution x^*, its objective function value ϕ^* is a lower bound on the integer optimum; hence we have shown that the algorithm is a $20 \log \phi^* \log\log \phi^*$-approximation algorithm.

As was observed in [ENSS95], the following standard ideas allow this to be converted to an $O(\log n \log\log n)$-approximation algorithm. The main idea is that we use an optimal fractional solution x^* to round and rescale the weights for an equivalent input so that the new fractional optimal value is upper bounded by a polynomial in n. First observe that the ratio between the optimal integer value and the optimal fractional value is at most n. To see this, take the optimal fractional solution x^*, and for each $a \in A$, set $\bar{x}(a) = 1$ if $x^*(a) \geq 1/n$, and set $\bar{x}(a) = 0$ otherwise. This is a feasible integer solution, since for any cycle $C \in \mathcal{C}$, there must exist some arc $a \in C$ such that $x^*(a) \geq 1/n$. Hence, each arc such that $c(a) > n\phi^*$ is not in any optimal feedback arc set. This implies that we can contract each such edge and obtain an equivalent instance.

Furthermore, if we include each arc a such that $c(a) < \phi^*/m$ in our feedback arc set, then the total cost of these arcs is at most ϕ^*. If we delete these arcs and apply a ρ-approximation algorithm to the resulting graph, the sum of the total costs of the deleted arcs and of the solution found is at most $\rho + 1$ times the optimum. Furthermore, we can now round down each cost $c(a)$ to its nearest integral multiple of ϕ^*/m, and then rescale by dividing ϕ^*/m to obtain a new input, in which each cost is an integer between 0 and nm. The rescaling changes only the units in which the costs are expressed, and the rounding down can introduce an error of at most ϕ^*. Hence, if we apply a ρ-approximation algorithm to our modified data, we obtain a solution of cost within a factor of $\rho + 2$ of optimal. Hence, we have achieved our goal of reducing the problem to instances in which ϕ^* can be bounded by a polynomial in n.

THEOREM 5.5 Given an optimal fractional solution x^* of value ϕ^*, Feedback is an $O(\log \phi^* \log\log \phi^*)$-approximation algorithm, and by preprocessing the input, yields an $O(\log n \log\log n)$-approximation algorithm for the minimum-cost feedback arc set problem in directed graphs.

One corollary of Theorem 5.5 is that the ratio between the optimal value of the integer program (5.42)–(5.44), and its linear relaxation is $O(\log \phi^* \log\log \phi^*)$. Alon and Seymour [Sey95] have shown that there are instances for which this ratio is $\Omega(\log \phi^*)$. Consequently, in order to improve this algorithm by more than a $\log\log \phi^*$ factor, it will be necessary to rely on a stronger lower bound.

5.4.3 FINDING A GOOD PARTITION

We return now to the proof of Lemma 5.7. The algorithm to find the claimed cut is quite similar to the one used in the proof of Theorem 5.1. As in the proof of that theorem, we can think of the graph G, the length function x, and the cost function c as defining a system of pipes: for each $(u, v) \in A$, there is a (one-way) pipe from u to v of length $x(u, v)$ and cross-section area $c(u, v)$. Hence, the objective function value ϕ corresponds to the total volume in the system.

Let $dist_x(u, v)$ denote the length of the shortest path from u to v in G with respect to the arc-length function x. For a given vertex $\hat{v} \in V$, we can compute the set of vertices reachable from $\hat{v}$ within a given radius r:

$$B_x^+(\hat{v}, r) = \{u \in V : dist_x(\hat{v}, u) \le r\}. \tag{5.51}$$

Analogously, let $B_x^-(\hat{v}, r) = \{u \in V : dist_x(u, \hat{v}) \le r\}$. Furthermore, we can compute the total volume of the pipe system that is contained within a radius r of vertex $\hat{v}$:

$$V^+(\hat{v}, r) = \sum_{u,v \in B_x^+(\hat{v},r)} c(u,v)x(u,v) + \sum_{(u,v)\in\delta^+(B_x^+(\hat{v},r))} c(u,v)(r - dist_x(\hat{v}, u)) \tag{5.52}$$

and

$$V^-(\hat{v}, r) = \sum_{u,v \in B_x^-(\hat{v},r)} c(u,v)x(u,v) + \sum_{(u,v)\in\delta^-(B_x^-(\hat{v},r))} c(u,v)(r - dist_x(v, \hat{v})). \tag{5.53}$$

Throughout the remainder of this section, we will omit the explicit reference to the particular length function x in our notation, with the understanding that it should be understood unambiguously.

Given the graph G, its cost function c, and a length function x of objective function value ϕ, we select a vertex $\hat{v}$ and then consider a sequence of radii from $\hat{v}$. For each radius r, we can consider the cuts defined by setting S to $B^+(\hat{v}, r)$ or to $B^-(\hat{v}, r)$; in the former case, we aim to bound the cost of $\delta^+(S)$, whereas in the latter we aim to bound the cost of $\delta^-(S)$.

For the purpose of the proof, we shall consider radii defined in the following way. We let the initial radius $r = 1/8$. While $V^+(\hat{v}, r) \le \phi/2$, we define an increment

$$d(r) = \frac{1}{10\log(\phi/V^+(\hat{v}, r))\log\log\phi};$$

If $V^+(\hat{v}, r + d(r)) < 2V^+(\hat{v}, r)$, then we claim that we can identify a good cut of the form $S = B^+(\hat{v}, \rho)$, where $r \le \rho \le r + d(r)$. If the volume repeatedly doubles until the volume captured exceeds $\phi/2$, then we enter a second phase in which the analogous steps are taken with respect to $V^-(\hat{v}, r)$ and $B^-(\hat{v}, r)$.

To complete the proof of the lemma, we need to establish three claims:
(1) if the volume does not double, then we can identify a good cut;
(2) the volume cannot expand beyond $\phi/2$ in both phases;
(3) the good cut is of a form that is easy to compute.

To prove the first claim, it is useful to recall some facts about the behavior of the the function $V^+(\hat{v}, r)$ for a fixed vertex $\hat{v}$. If there does not exist a vertex u such that $dist_x(\hat{v}, u) = r$, then this function is differentiable at r, and its derivative is equal to the total cost of the cut defined by the current radius r:

$$\sum_{(u,v)\in\delta^+(B^+(\hat{v},r))} c(u,v).$$

So the function consists of fewer than n linear pieces. There might be discontinuities at the breakpoints, each of which corresponds to a vertex at the prescribed distance from $\hat{v}$. However, it is a non-decreasing function of r. Hence, if we have identified an interval $[r, r + d]$ over which the volume increases by at most ν, there must exist a point within that interval at which the derivative is at most ν/d, and hence there is a cut of a total cost at most ν/d.

Suppose that we have found a radius $\bar{r}$ for which $V^+(\hat{v}, \bar{r} + d(\bar{r})) < 2V^+(\hat{v}, \bar{r})$. Hence, the increase in volume over this range is less than $V^+(\hat{v},\bar{r})$, and so the overall rate of increase is less than

$$\frac{V^+(\hat{v},\bar{r})}{d(\bar{r})} = 10V^+(\hat{v},\bar{r})\log\frac{\phi}{V^+(\hat{v},\bar{r})}\log\log\phi.$$

Let $\rho, \bar{r} \le \rho \le \bar{r}+d(\bar{r})$, denote a value of r for which

$$\frac{\partial V^+(\hat{v},r)}{\partial r} < 10V^+(\hat{v},\bar{r})\log\frac{\phi}{V^+(\hat{v},\bar{r})}\log\log\phi,$$

and let $S = B^+(\hat{v},\rho)$.

We shall show S satisfies the properties required by the lemma. Let $\epsilon = V^+(\hat{v},\rho)/\phi$. Since

$$V^+(\hat{v},\bar{r}) \le V^+(\hat{v},\rho) = \epsilon\phi \le 2V^+(\hat{v},\bar{r}),$$

we have that

$$\sum_{a\in\delta^+(S)} c(a) \le 10\epsilon\phi\log(2/\epsilon)\log\log\phi;$$

applying the crudest of estimates, we see that the cost of the cut defined by S attains the claimed bound. Furthermore, the objective function value of x restricted to the arcs $A[S]$ is at most the volume $V^+(\hat{v},\rho)$; hence, this is at most $\epsilon\phi$. On the other hand, the objective function value of x restricted to the arcs $A[\bar{S}]$ is at most the volume $\phi - V^+(\hat{v},\rho)$; hence, this is at most $\bar{\epsilon}\phi$. Repeating the analogous argument with respect to V^-, we see that a cut S found in either phase satisfies the conditions of the lemma.

We next turn to the proof of the second claim, that at some point in the two phases, we find some interval $[r, r+d(r)]$ for which the volume does not double. We shall assume, for a contradiction, that this is not the case, and then show that this assumption implies that there exists a directed cycle C of length $\sum_{a\in C} x(a)$ is less than 1; this contradicts the fact that x is a feasible solution to the linear relaxation. In fact, we will show that there exists some vertex $\hat{u}$ such that $dist_x(\hat{u},\hat{v}) < 1/2$ and $dist_x(\hat{v},\hat{u}) < 1/2$.

First, we compute an upper bound on the final radius computed in the first phase; that is, the radius $\hat{r}$ such that $V^+(\hat{v},\hat{r}) > \phi/2$ that causes the phase to end. To give an upper bound on $\hat{r}$, we upper bound the increments $d(r)$ computed along the way. For each increment $d(r)$ computed, there is a corresponding volume $V^+(\hat{v},r)$, where $1/8 \le V^+(\hat{v},r) \le \phi/2$. Since the volume doubles with each iteration, at most one of these values lies in the interval $(\phi/2^{\ell+1}, \phi/2^{\ell}]$, for each $\ell = 1,\ldots,\lfloor\log\phi\rfloor+4$. If there is an increment that corresponds to the interval $(\phi/4,\phi/2]$, then it is at most

$$\frac{1}{(10\log(\phi/(\phi/2))\log\log\phi)} = \frac{1}{(10\log\log\phi)};$$

the increment corresponding to the interval $(\phi/2^{\ell+1}, \phi/2^{\ell}]$ is at most $1/(10\ell\log\log\phi)$. Hence, we see that

$$\hat{r} \le \frac{1}{8} + \sum_{\ell=1}^{\lfloor\log\phi\rfloor+4}\frac{1}{10\ell\log\log\phi} = \frac{1}{8} + \frac{1}{10\log\log\phi}\mathcal{H}(\lfloor\log\phi\rfloor+4). \tag{5.54}$$

Using even the crudest estimates, it is easy to verify that this implies that $\hat{r} < 1/2$.

The identical calculation also shows that if the second phase does not end until the current radius yields a volume greater than $\phi/2$, then the final radius for the second phase is less than 1/2. However, between the two phases, more than ϕ units of volume have been captured. Since there are only ϕ units of volume overall, some part of the pipe system must be captured in both phases. But then, there must exist some vertex $\hat{u}$ such that $dist_x(\hat{v}, \hat{u}) < 1/2$ and $dist_x(\hat{u}, \hat{v}) < 1/2$. This contradicts the fact that x is a feasible solution to the linear relaxation. Hence, in one of the two phases, we must indentify a good cut.

Finally, we must argue that we can find a good cut. For any cut S, it is straightforward to check if there exists some ϵ for which conditions (5.46)-(5.48) hold. We have already shown that there exists a good cut of the form $B^-(\hat{v}, \rho)$ or $B^+(\hat{v}, \rho)$. However, for any given $\hat{v}$, there are at most n distinct cuts the form $B^+(\hat{v}, \rho)$: one need only consider $\rho = dist(\hat{v}, v)$ for some $v \in V$. Of course, the same is true for $B^-(\hat{v}, \rho)$. Hence, the algorithm Feedback-Cut, used in the algorithm Feedback, need only enumerate these cuts until it finds a good one (see Figure 5.7).

Note that we can give a much better implementation of Feedback-Cut by considering the vertices v in order of their distance from $\hat{v}$; this implies that the algorithm is just a slight modification to Dijkstra's algorithm for the single-source shortest path problem (see, for example, the textbook of Cormen, Leiserson, and Rivest [CLR90]): Dijkstra's algorithm repeatedly identifies the next closest vertex to a given vertex $\hat{v}$, and hence continually maintains a cut S of the appropriate form; after finding the next closest vertex to $\hat{v}$, we need only check if the current S satisfies the conditions of Lemma 5.7.

5.5 FINDING BALANCED CUTS AND OTHER APPLICATIONS

While the sparsest cut problem is of interest in its own right, it is just the starting point for a host of applications. In particular, we will show how a subroutine that produces sparse cuts can be used to compute near-optimal balanced cuts; the approximation algorithm for the balanced cut problem can then be used in a variety of settings to derive divide-and-conquer approximation algorithms.

function Feedback-Cut(G, x)
    fix $\hat{v} \in V$;
    for each $v \in V - \{\hat{v}\}$
        $S \leftarrow \{u \in V : dist_x(\hat{v}, u) \leq dist_x(\hat{v}, v)\}$;
        if S satisfies (5.46)-(5.48) **then return** S;
        $T \leftarrow \{u \in V : dist_x(u, \hat{v}) \leq dist_x(v, \hat{v})\}$;
        if T satisfies (5.46),(5.47), & (5.48) **then return** T.

FIGURE 5.7

The algorithm Feedback-Cut

5.5.1 FINDING BALANCED CUTS

In the *α-balanced cut problem*, we are given an undirected graph $G = (V, E)$, where each edge $e \in E$ has a positive cost $c(e)$; an α-balanced cut is a subset of vertices $S \subseteq V$ such that $\alpha n \leq |S| \leq (1-\alpha)n$, where $n = |V|$, and the objective is to find such a cut for which its total cost, $\sum_{e\in\delta(S)} c(e)$, is minimized. Let C_α denote the total cost of the optimal α-balanced cut. The special case in which $\alpha = 1/2$ is sometimes referred to as the *graph bisection problem*. We have already discussed in Section 5.1 that an approximation algorithm for this problem can be quite useful in designing divide-and-conquer approximation algorithms. We shall next show that the algorithm of Section 5.3 for the sparsest cut problem can be used in this setting.

Given an instance of the α-balanced cut problem, we can construct an instance of the sparsest cut problem by letting there be $\binom{n}{2}$ terminal pairs, one for each pair of vertices $u, v \in V$, and setting the demand of each commodity to be 1; we shall call this the *all-pairs unit-demand problem*. For any cut $S \subseteq V$, its sparsity ratio $\rho(S)$ is equal to the ratio of its total cost $\sum_{e\in\delta(S)} c(e)$ to $|S||V - S|$, since the latter quantity is the total demand of terminal pairs that are disconnected by deleting the edges in $\delta(S)$. This denominator is made large by keeping S and $V - S$ of roughly the same size; that is, the cut is balanced. In fact, if S is α-balanced, then the denominator is at least $n^2\alpha(1-\alpha)$, and hence the sparsity ratio is at most $C_\alpha/[n^2\alpha(1-\alpha)]$. Of course, the sparsity ratio need not be minimized by finding the cheapest balanced cut: there might be very imbalanced cuts for which the total cost is sufficiently small so as to offset the fact that the denominator is smaller.

The greedy set covering algorithm computes a cover by repeatedly choosing a set which is the cheapest per element covered (see Chapter 3 for an analysis of this algorithm). The following algorithm Greedy-Ratio for the α-balanced cut problem is analogous: we repeatedly choose additional vertices to add to a cut $\bar{S}$ by choosing additional vertices, for which the cost of the new edges cut per vertex selected is small. We initialize $V_1 = V$; in general, V_i will denote the set of vertices remaining at the start of the ith iteration of the algorithm. In iteration i, we apply our sparsest cut approximation algorithm to all-pairs unit-demand problem for the graph induced by V_i; the cut found by this algorithm is a partition of the vertices into two sets S_i and $V_i - S_i$, where we shall assume that S_i is the smaller of the two. If $\bar{S} = \cup_{\ell=1}^{i} S_\ell$ is such that $|\bar{S}| > \alpha|V|$, then the algorithm terminates. Otherwise, we set $V_{i+1} = V_i - S_i$, and the algorithm continues.

LEMMA 5.8 For any $\alpha \leq 1/3$, the algorithm Greedy-Ratio finds an α-balanced cut.

Proof. The algorithm must clearly halt at some point; let ℓ denote the iteration in which it halts. Let $k = |\cup_{i=1}^{\ell-1} S_i|$. The set $\bar{S}$ found by the algorithm has size

$$k + |S_\ell| \leq k + (n-k)/2 \leq n/2 + k/2 < (1+\alpha)n/2 \leq (1-\alpha)n.$$

■

We next turn to showing that the cut found by Greedy-Ratio is near-optimal. We will not show that it is an approximation algorithm in the traditional sense, since we will not compare the cost of the cut found to the optimal α-balanced cut, C_α. Instead, we will compare the cost of the α-balanced cut found to the cost of the optimal β-balanced cut, C_β, where $\beta > \alpha$. Since it is more restrictive to require a cut to be β-balanced than to be α-balanced, it is possible that C_β is much greater than C_α.

Before completing the analysis of the performance of this algorithm, we first introduce some notation. The input graph evolves over the course of the execution of the algorithm, as more and more of its vertices are deleted. Thus, a notation such as $\delta(S)$ is ambiguous, since it is not clear to which graph we are referring. For any set of vertices $S \subseteq V_i$, we shall let

$$c_i(S) = \sum_{e \in \delta_i(S)} c(e),$$

where $\delta_i(S)$ denote the set of edges (u, v) such that $u \in S$ and $v \in V_i - S$. Since $V = V_1$, we will define $c(S) = c_1(S)$ for any $S \subseteq V$. If ℓ denotes the number of iterations that the algorithm makes, we shall set $n_i = |V_i|$ and $s_i = |S_i|$, for each $i = 1, \ldots, \ell$.

THEOREM 5.6 If the Greedy-Ratio algorithm uses a ρ-approximation algorithm to find a sparse cut in each iteration, then, for any $\beta > \alpha$, where $\alpha \le 1/3$, the algorithm finds an α-balanced cut of cost at most $\frac{3\rho}{\beta-\alpha} C_\beta$.

Proof. Let R denote an optimal β-balanced cut in the input $G = (V, E)$. Consider the cut S_i found in iteration i, $i = 1, \ldots, \ell$. We know that $R_i = R - \cup_{j=1}^{i-1} S_j$ contains more than $(\beta - \alpha)n$ vertices; in fact, we also know that $V_i - R_i$ also has this many vertices, and since one of these two sets has size greater than $(1-\alpha)n/2 \ge n/3$, we know that the product of their sizes is greater than $(\beta-\alpha)n^2/3$. Hence, for the graph considered in iteration i, the sparsest cut value is at most $\frac{c_i(R_i)}{(\beta-\alpha)n^2/3}$. This implies that

$$\frac{c_i(S_i)}{s_i(n_i - s_i)} \le \rho \cdot \frac{c_i(R_i)}{(\beta-\alpha)n^2/3} \le \rho \cdot \frac{c(R)}{(\beta-\alpha)n^2/3},$$

and hence,

$$c_i(S_i) \le \frac{\rho C_\beta}{(\beta-\alpha)n/3} \cdot s_i.$$

Summing this inequality for $i = 1, \ldots, \ell$, we see that

$$c(\bar{S}) \le \sum_{i=1}^{\ell} c_i(S_i) \le \sum_{i=1}^{\ell} \frac{\rho C_\beta}{(\beta-\alpha)n/3} \cdot s_i \le \frac{3\rho C_\beta}{(\beta-\alpha)}.$$

■

The relationship between the sparsest cut problem and obtaining a balanced cut was independently discovered by Leighton & Rao [LR88] and Plaisted [Pla90]. By combining Theorem 5.6 with the $O(\log k)$-approximation algorithm for the sparsest cut problem that was given in Section 5.3, we obtain the following corollary.

COROLLARY 5.3 For any fixed ϵ, $0 \le \epsilon \le 1/6$, there is a polynomial-time algorithm that finds a $1/3$-balanced cut of cost within an $O(\log n)$ factor of the cost of the optimal $1/3 + \epsilon$-balanced cut.

5.5.2 APPLICATIONS OF BALANCED CUT THEOREMS

In Section 5.1, we showed that an algorithm to find near-optimal balanced cuts could be used to derive an approximation algorithm for the minimum cut linear arrangement problem, via a divide-and-conquer approach. In this section, we will consider two other applications of this philosophy. Each of these applications will rely on a good balanced-cut algorithm, but in each case we will need a different sort of balanced cut. However, the techniques discussed in this chapter can be extended to yield the required subroutines.

We first consider a problem related to the solution of a linear system of equations $Ax = b$, where A is a positive-definite symmetric matrix, and we wish to compute x for a given input (A, b). A well-known method for solving such a system of equations is to apply Gaussian elimination. Since we are applying this method to a symmetric positive-definite matrix, we may restrict attention to the variant of Gaussian elimination in which we use only diagonal pivot entries. Furthermore, in each iteration of the algorithm, we maintain a symmetric matrix as we gradually transform the system of equations into an equivalent system (A', b') in which A' is the identity matrix. In most applications, the matrix A is extremely sparse; that is, most of its entries are zeroes. The processing of A can be made much more efficient if its sparsity can be maintained as it is transformed to A'. One very stringent requirement of this type is to require that each 0 in the matrix A should remain a 0 throughout the matrix's transformation. For example, in the following sequence, we perform Gaussian elimination without introducing any non-zeroes, where the starred element is the pivot element in the current iteration:

$$\begin{bmatrix} 4 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1^* \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 1 & 0 \\ 1 & 1^* & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 2^* & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1^* & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

This is a consequence of the order in which we chose the pivot elements. Had we started by pivoting on the 4, we would have obtained the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & -1 \\ 0 & -1 & 3 \end{bmatrix}.$$

If we pivot on the diagonal element a_{tt}, then the i, jth entry of the new matrix is $a_{ij} - a_{it}a_{tj}/a_{tt}$, $i, j \neq t$. Hence, we are assured that no new non-zero is created if we select an index t such that

$$a_{ij} = 0 \text{ implies that } a_{it} = 0 \text{ or } a_{tj} = 0, \text{ for each } i, j \neq t. \tag{5.55}$$

For simplicity, we shall assume that any non-zero remains a non-zero throughout the computation; in other words, if Gaussian elimination computes an element of the form $c - c$, where $c \neq 0$, then we shall still treat this new element as a non-zero.

We shall first focus on the following question: for which symmetric positive-definite matrices A does there exist a sequence of pivots so that no new non-zeroes are created throughout Gaussian elimination. This question has a rather elegant answer. Symmetric matrices can be easily modeled by graphs. For each index $1, \ldots, n$ create a vertex. For each non-zero entry a_{ij}, create the edge (i, j). Note that the fact that the matrix A is symmetric implies that this construction yields an undirected graph. If we consider the contrapositive of condition (5.55) and translate this into this graph setting, we see that a pivot can be performed on a_{tt} without creating a non-zero if there is an edge connecting i and j whenever there are edges (t, i) and (t, j); in other words, the neighbors of vertex t induce a clique in the graph. Hence, a matrix can be reduced to the identity without introducing non-zeroes, whenever the corresponding graph has the following property: the vertices can be ordered $v_1, \ldots, v_n$, such that, for each $j = 1, \ldots, n-1$, the neighbors of v_j in $\{v_{j+1}, \ldots, v_n\}$ induce a clique in the original graph. Such an ordering of the vertices is a called a *perfect elimination ordering*. Hence, we are interested in those graphs which admit a perfect elimination ordering. These graphs can be nicely characterized. They are precisely the class of *chordal graphs*: those graph for which any cycle of length at least 4 has an edge connecting some pair of vertices that are not consecutive in the cycle (a so-called *chord* of the cycle). These graphs are also commonly called *triangulated graphs*. It is beyond the scope of this chapter to prove this characterization of chordal graphs, and the reader is referred to the textbook of Golumbic [Gol80].

However, not all graphs are chordal, and hence in some cases, non-zeroes must be introduced in the process of performing Gaussian elimination. We shall be interested in finding an ordering of pivot elements so that the resulting number of non-zeroes is minimized. We shall this the *minimum fill-in problem* for symmetric positive-definite matrices. By the previous discussion, this problem is equivalent to the following graph theoretic question: given a graph $G = (V, E)$, find a superset of edges $F \supseteq E$ such that the graph (V, F) is a chordal graph, so that the size of F is as small as possible. We shall call this the *minimum chordal extension problem*. Yannakakis [Yan81] showed that this problem is *NP*-complete.

Agrawal, Klein, & Ravi [AKR93] focus on the case in which the maximum degree in the given graph is a constant Δ, or equivalently, there are a constant number of non-zeroes in each row of the given matrix. They gave the first approximation algorithm with a non-trivial performance guarantee for this problem, and we shall present the main ideas of this result. This result relies on several sophisticated results on the structure of chordal graphs, and it is beyond the scope of this chapter to prove the correctness and performance of this algorithm. For these results, and for a concise history of research on this problem, the reader is referred to [AKR93].

For a given ordering of the vertices $v_1, \ldots, v_n$, we compute a sequence of graphs G_i, $i = 0, \ldots, n$. The graph G_0 is the input graph G. The graph G_i is computed from G_{i-1} by adding each edge (v_j, v_k), $i < j < k$, such that v_j and v_k are non-adjacent neighbors of v_i in G_{i-1}. The graph G_n is a chordal graph; it can be viewed as the unique chordal graph corresponding to this *elimination ordering* of the vertices of G.

Suppose that we can find a small subset of nodes S in G whose deletion separates G into two substantially smaller connected components with vertex sets V_1 and V_2. Consider any elimination ordering of G in which we first order V_1, then V_2, and then S. Consider the sequence of graphs G_i, $i = 0, \ldots, n$, for this ordering. At any step i in which $v_i \in V_1$, we introduce only edges induced by $V_1 \cup S$; in any step in which $v_i \in V_2$, we introduce only edges induced by $V_2 \cup S$; finally in any step in which $v_i \in S$, we introduce only edges induced by S. Hence, we never introduce edges between V_1 and V_2, and hence we have decomposed the original problem into two substantially smaller problems. This approach has long been applied in this context; this divide-and-conquer philosophy has been referred to as *nested dissection*. The key to the performance of any nested dissection algorithm is the manner in which the set S is computed.

Agrawal, Klein, and Ravi proposed a nested dissection algorithm in which the separator is a near-optimal balanced node separator. Such a separator was known to be computable (Leighton & Rao [LR94] and Tragoudas [Tra91]). A subset of nodes S is an α-*balanced node separator* of an n-vertex graph $G = (V, E)$ if each connected component of $V - X$ has at most $(1-\alpha)n$ vertices. Let C_α denote the minimum size of an α-balanced node separator.

THEOREM 5.7 There exists a polynomial-time algorithm to find a 1/3-balanced node separator of a graph of size $O(\log n C_{1/2})$.

While the algorithm of Agrawal, Klein, & Ravi [AKR93] is a quite natural one, its analysis relies on a number of structural characterizations of chordal graphs that are beyond the scope of this chapter. For example, one key ingredient for the performance guarantee of this approximation algorithm is a result of Gilbert, Rose, and Edenbrandt [GRE84] that states that for each m-edge chordal graph, $C_{1/2} \leq \sqrt{2m}$. Agrawal, Klein, & Ravi show that for graphs in which the maximum degree is bounded by a constant, the resulting nested dissection algorithm has a polylogarithmic performance guarantee.

THEOREM 5.8 For any graph G of maximum degree Δ, the nested dissection algorithm based on near-optimal node-separators computes a chordal extension of G in which the number of edges is within an $O(\sqrt{\Delta}\log^4 n)$ factor of optimal.

We shall consider one final application of this divide-and-conquer approach. In the *storage-time product problem*, the input consists of an directed acyclic graph, which corresponds to the dependency graph of a computation; that is, each node v corresponds to an intermediate result that is computed, and if there are arcs entering v from $u_1, \dots, u_k$, this means that the result computed for each of the predecessors is needed to compute the result for v. Each node v requires a specified processing time $p(v)$, and for each edge $(u, v) = e$, there is a cost $c(e)$ which corresponds to the amount of storage needed for the intermediate result computed at node u for node v. One can interpret the input graph as defining a precedence relation $\prec$ on the nodes (where $u \prec v$ if there is a path from u to v), and then a feasible solution for this problem is a total ordering $v_1, v_2, \dots, v_n$ of the nodes that is consistent with $\prec$ (that is, $v_i \prec v_j$ implies that $i < j$). Then the cost associated with this solution is

$$\sum_{e=(v_i,v_j)\in E} c(e)\; (\sum_{k=i}^{j-1} p(v_k)), \tag{5.56}$$

that is, the total storage-time product needed to maintain intermediate results. That is, the storage-time product problem is to compute the ordering of V consistent with $\prec$ for which the cost (5.56) is minimized.

Ravi, Agrawal, & Klein [RAK91] have given an $O(\log n \log P)$-approximation algorithm for this problem, where P denotes the total processing time. For ease of exposition, we will present their algorithm for the special case in which each processing time $p(v) = 1$. By focusing on this case, we reduce the problem to one quite similar to the minimum cut linear arrangement problem discussed in Section 5.1. As in that case, let $\sigma(i)$ denote the vertex assigned to be processed in the ith position, for each $i = 1 \dots, n$.

Furthermore, let S_i denote the set of edges in E of the form $(\sigma(j), \sigma(k))$, where $j \leq i$ and $k > i$. Then the problem can be rephrased as follows: find σ consistent with $\prec$ such that $\sum_{i=1}^{n-1} \sum_{e \in S_i} c(e)$ is minimized. In effect, the minimum cut linear arrangement problem is the bottleneck (or min-max) version of this min-sum problem (but we are also constrained here to permutations σ that are consistent with $\prec$).

The idea underlying the approximation algorithm of Ravi, Agrawal, & Klein is exactly as for the minimum cut linear arrangement problem. That is, we need to compute a balanced partition of the nodes, and recursively solve the two subproblems. Of course, for the partition to make sense, we need to find a partition of V into S and $V - S$ such that there does not exist an edge (v, u) in G where $u \in S$ and $v \in V - S$; that is, $\delta^-(S) = \emptyset$. Thus, we shall focus on such *$\prec$-consistent cuts* S and their corresponding edge sets $\delta^+(S)$. Once again, we will say that such a cut is α-balanced if $\alpha n \leq |S| \leq (1-\alpha)n$. The cost of such a cut S is $\sum_{e \in \delta^+(S)} c(e)$. Ravi, Agrawal, & Klein have observed that the approach of Leighton & Rao can be extended to yield the following theorem.

THEOREM 5.9 There is a polynomial-time algorithm to find a 1/4-balanced $\prec$-consistent cut in a directed acyclic graph of cost within an $O(\log n)$ factor of the minimum cost of a 1/3-balanced $\prec$-consistent cut.

The algorithm for the storage-time product minimization problem is quite natural: we apply the algorithm of Theorem 5.9 to the graph to partition the problem into two subproblems. Let $\mathcal{A}(G)$ denote the cost of the solution found by the algorithm on input G, and let $\mathcal{B}(G)$ denote the cost of the cut S found by the algorithm of Theorem 5.9. Furthermore, let G_1 denote the graph induced by S, and let G_2 denote the graph induced by $V - S$. Since each edge in G is either in G_1, in G_2, or in the cut $\delta^+(S)$, we can derive the following recurrence relation:

$$\mathcal{A}(G) \leq \mathcal{A}(G_1) + \mathcal{A}(G_2) + (n-1)\mathcal{B}(G),$$

where the last term is a consequence of the fact that each edge in $\delta^+(S)$ might occur in each of the $n-1$ sets S_i, $i = 1, \ldots, n-1$.

The crux of the analysis of the performance guarantee of this algorithm lies in devising a good lower bound on the optimum. Consider an optimal solution σ^*. If we consider any set S_i, $i = \lceil n/3 \rceil, \ldots, \lfloor 2n/3 \rfloor$, then we see that it defines a 1/3-balanced $\prec$-consistent cut. Hence, the cost of each of these is at least B^*, where B^* is the minimum cost of a 1/3-balanced $\prec$-consistent cut. Hence $nB^*/3$ is a lower bound on the optimal value, $OPT_{S \times T}(G)$, and hence we have the recurrence:

$$\mathcal{A}(G) \leq \mathcal{A}(G_1) + \mathcal{A}(G_2) + O(\log n) \cdot OPT_{S \times T}(G).$$

By relying on the fact that each G_i has at most $(3/4)n$ vertices, it is easy to show by induction that $\mathcal{A}(G) = O(\log^2 n) OPT_{S \times T}(G)$. It is not hard to extend this argument to give the bound claimed for the general case in which there are arbitrary processing times. This somewhat more general approach to using a balanced cut as a lower bound was first introduced in a different context by Hansen [Han89].

Although we have only given a few examples, there are quite a number of applications for which this approach has led to the first approximation algorithm with polylogarithmic performance guarantee. Leighton & Rao considered several applications from the domain of VLSI design: among these are results for the crossing number of a graph (i.e., embedding a graph in the plane so as to minimize the number of pairs of edges that cross each other in this embedding); and for the problem of laying out a graph in the plane so as to minimize the area required for the layout. Many of these applications were first proposed by Bhatt & Leighton [BL84] who focused attention on obtaining approximation algorithms for the graph bisection problem.

5.6 CONCLUSIONS

In this chapter, we have tried to present the highlights of an approach to the design of approximation algorithms that seeks to capitalize on good algorithms for cut problems in order to design and analyze divide-and-conquer algorithms. We have not attempted to catalog all of the work done in this area, and therefore have not discussed a number of closely related results. However, we will briefly mention two of the most prominent areas that we have omitted.

The first success in applying divide-and-conquer based on finding good cuts was for the restricted case of planar graphs; as one of the initial applications of their planar separator theorem, Lipton & Tarjan [LT80] observed that one could derive polynomial approximation schemes for a wide variety of combinatorial problems. However, in the subsequent years, Baker [Bak94] has given a better approach for exploiting planarity, and her work is discussed in Chapter 9. Furthermore, we have also omitted discussion of work on better performance guarantees of approximation algorithms for these cut problems when restricted to planar graphs, which includes results of Rao [Rao87], Garg, Saran, & Vazirani [GSV94], Tardos & Vazirani [TV93], and Klein, Plotkin, & Rao [KPR91].

Randomized rounding has also been applied in the context multicut approximation algorithms; Bertsimas, Teo, & Vohra [BV94, BTV95] have given other mathematical programming formulations which are more amenable to this approach. However, the bounds obtained from this approach do not dominate the ones presented here, and in some sense, build on the earlier work. Randomized rounding is extensively discussed in Chapter 11.

This area of research is still extremely active, and there is still much work to be done. For example, very little is known about multicommodity max-flow min-cut bounds when the input graph is directed. Even more importantly, for each of the problems discussed, there is no complexity-theoretic evidence that a significantly better performance guarantee cannot be obtained; in fact, there may well be approximation algorithms with constant performance guarantees. In fact, Chung & Yau [CY94] gave an algorithm for which they claimed such a performance guarantee for the balanced cut problem, but the proof contained in [CY94] is not correct, and a correct proof of this claim has not been published. Finally, it is ironic that the pivotal result in this chapter, Corollary 5.3, which finds a good balanced cut, is *not* an approximation algorithm in the traditional sense. While this deficiency is unimportant for any of the applications, it would, nonetheless, be an important advance to find a good approximation algorithm for this problem.

ACKNOWLEDGEMENTS

First and foremost, I would to thank Éva Tardos, from whom I first learned most of the results in this chapter. I would also like to thank Satish Rao for explaining his insights into Seymour's algorithm at a time they were still evolving as part of his own research. This chapter benefited greatly from the perceptive comments of Yuval Rabani, David Williamson, Yuri Rabinovich, and this volume's editor, Dorit Hochbaum. This work was partially supported by NSF grant CCR-9307391.

REFERENCES

[AKR93] A. Agrawal, P. Klein, and R. Ravi. Cutting down on fill using nested dissection: provably good elimination orderings. In: J.A. George, J.R. Gilbert, and J. Liu, editors. *Sparse Matrix Computations: Graph Theory Issues and Algorithms, IMA Volumes in Mathematics and Its Applications*, Springer-Verlag, New York, 1993, pages 31–55.

[AMO93] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[AR95] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, to appear.

[AvisD91] D. Avis and M. Deza. The cut cone, L^1 embeddability, complexity, and multicommodity flows. *Networks*, 21:595–617, 1991.

[Bak94] B. S. Baker. Approximation algorithms for *NP*-complete problems on planar graphs. *J. Assoc. Comput. Mach.*, 41:153–180, 1994.

[BTV95] D. Bertsimas, C. Teo, and R. Vohra. Nonlinear formulations and improved randomized approximation algorithms for multicut problems. In E. Balas and J. Clausen, editors, *Integer Programming and Combinatorial Optimization*, *Lecture Notes in Computer Science 920*, pages 29–40. Springer, 1995.

[BV94] D. Bertsimas and R. Vohra. Linear programming relaxations, approximation algorithms, and randomized rounding: a unified approach to covering problems. Working paper #-3654-94 MSA, Sloan School of Management, MIT, Cambridge, MA, 1993.

[BL84] S.N. Bhatt and F.T. Leighton. A framework for solving VLSI graph layout problems. *J. Comput. Sys. Sciences*, 28:300–343, 1984.

[Bou85] J. Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel J. Math.*, pages 46–52, 1985.

[CY94] F. R. K. Chung and S. T. Yau. A near optimal algorithm for edge separators. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 1–8, 1994.

[Chv83] V. Chvátal. *Linear programming*. W.H. Freeman, New York, 1983.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw Hill, Cambridge, MA and New York, 1990.

[DJP+92] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiway cuts. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 241–251, 1992.

[ENRS95] G. Even, J. Naor, S. Rao, and B. Schieber. Divide-and-conquer approximation algorithms via spreading metrics. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 62–71, 1995.

[ENSS95] G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multi-cuts in directed graphs. In E. Balas and J. Clausen, editors, *Integer Programming and Combinatorial Optimization*, *Lecture Notes in Computer Science 920*, pages 14–28. Springer, 1995.

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

[Gar95] N. Garg. A deterministic $O(\log k)$-approximation algorithm for the sparsest cut problem. Preprint, 1995.

[GSV94] N. Garg, H. Saran, and V.V. Vazirani. Finding separator cuts in planar graphs within twice the optimal. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1994.

[GK94] M. D. Grigoriadis and L. G. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM J. Optimization*, 4:86–107, 1994.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer-Verlag, Berlin, 1988.

[Gol80] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[GRE84] J.R. Gilbert, D.J. Rose, and A. Edenbrandt. A separator theorem for chordal graphs. *SIAM J. Alg. Discrete Methods*, 5:306–313, 1984.

[GVY93] 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.

[Han89] M. Hansen. Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 604–609, 1989.

[Hu63] T.C. Hu. Multicommodity network flows. *Operations Res.*, 11:344–360, 1963.

[Kah93] N. Kahale. On reducing the cut ratio to the multicut problem. Technical Report 93–78, DIMACS, 1993.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica* 4:373–395, 1984.

[Kha79] L.G. Khachiyan. A polynomial algorithm in linear programming (in Russian). *Doklady Akademiia Nauk SSSR*, 244:1093–1096, 1979. English translation: *Soviet Mathematics Doklady* 20:191–194.

[KPR91] P. Klein, S. Plotkin, and S. Rao. Planar graphs, multicommodity flow, and network decomposition. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 682–690, 1991.

[KPST94] P. Klein, S. Plotkin, C. Stein, and É. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM J. on Computing*, 23:310–321, 1994.

[KRAR95] P. Klein, S. Rao, A. Agrawal, and R. Ravi. An approximate max-flow min-cut relation for undirected multicommodity flow, with applications. *Combinatorica*, 15:187–202, 1995.

[LLR95] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–246, 1995.

[LMP+95] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *J. Comput. Sys. Sciences*, 50:228–243, 1995.

[LR88] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[LR94] F. T. Leighton and S. Rao. An approximation max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. Unpublished manuscript, 1994.

[LT80] R.J. Lipton and R.E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9:615–627, 1980.

[MR95] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, Cambridge, 1995.

[Pla90] D. Plaisted. A heuristic algorithm for small separators in planar graphs. *SIAM J. Comput.*, 19:267–280, 1990.

[PST95] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math. Oper. Res.*, 20:257–301, 1995.

[PT95] S. Plotkin and É. Tardos. Improved bounds on the max-flow min-cut ratio for multicommodity flows. *Combinatorica*, 15:425–434, 1995.

[RAK91] R. Ravi, A. Agrawal, and P. Klein. Ordering problems approximated: single-processor scheduling and interval graph completion. In *Proceedings of the 18th International Colloquium on Automata, Languages, and Processing, Lecture Notes in Computer Science 510*, pages 751–762, 1991.

[Rao87] S. Rao. Finding near optimal separators in planar graphs. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 225–237, 1987.

[Sey95] P.D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15:281–288, 1995.

[SM90] F. Shahrokhi and D.W. Matula. The maximum concurrent flow problem. *J. Assoc. Comput. Mach.*, 37:318–334, 1990.

[Tra91] S. Tragoudas. *VLSI partitioning approximation algorithms based on multicommodity flow and other techniques*. PhD thesis, University of Texas, Dallas, 1991.

[TV93] É. Tardos and V. V. Vazirani. Improved bounds for the max-flow min-multicut ratio for planar and $K_{r,r}$-free graphs. *Inform. Proc. Lett.*, 47:77–80, 1993.

[Yan81] M. Yannakakis. Computing the minimum fill-in is *NP*-complete. *SIAM J. Alg. Discrete Methods*, 2:77–79, 1981.