Multicommodity Max-Flow Min-Cut Theorems and Their Use in Designing Approximation Algorithms

TOM LEIGHTON

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

SATISH RAO

NEC Research Institute, Princeton, New Jersey

Abstract. In this paper, we establish max-flow min-cut theorems for several important classes of multicommodity flow problems. In particular, we show that for any n-node multicommodity flow problem with uniform demands, the max-flow for the problem is within an $O(\log n)$ factor of the upper bound implied by the min-cut. The result (which is existentially optimal) establishes an important analogue of the famous 1-commodity max-flow min-cut theorem for problems with multiple commodities. The result also has substantial applications to the field of approximation algorithms. For example, we use the flow result to design the first polynomial-time (polylog n-times-optimal) approximation algorithms for well-known NP-hard optimization problems such as graph partitioning, min-cut linear arrangement, crossing number, VLSI layout, and minimum feedback arc set. Applications of the flow results to path routing problems, network reconfiguration, communication in distributed networks, scientific computing and rapidly mixing Markov chains are also described in the paper.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms: Algorithms

Additional Key Words and Phrases: Approximation algorithms, divide and conquer, graph bisection, graph partitioning, maximum flow, minimum cut, muticommodity flow, routing, VLSI layout

1. *Introduction*

In this paper, we study the relationship between the maximum flow and the minimum cut in multicommodity flow problems. Our research is motivated in

---

This paper was supported in part by ARPA Contracts N00014-92-J-1799 and N00014-95-1-1246, Air Force Contract AFOSR F49620-92-J-0125, and Army Contract DAAH04-95-1-0607.

Authors' present addresses: T. Leighton, Department of Mathematics and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139; S. Rao, Computer Science Division, Soda Hall, University of California, Berkeley, CA 94720-1776.

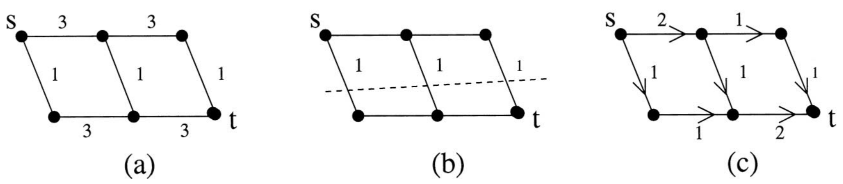

FIG. 1. A 1-commodity flow problem (a) for which the min-cut (b) and max-flow (c) are both 3.

part by the seminal work of Ford and Fulkerson [1956], which showed that for 1-commodity flow problems, the max-flow and min-cut are always equal.

1.1. SINGLE COMMODITY FLOW PROBLEMS. In a 1-commodity flow problem, there is an underlying network with n nodes V and m edges E. Each edge $e \in E$ is provided with a nonnegative capacity $C(e)$, which represents the maximum amount of flow that can pass through the edge. (Unless otherwise stated, we assume that edges are undirected, in which case flow can pass through edges in either direction.) One of the nodes is designated as the *source* s and one as the *sink* t. The objective is to route as much flow as possible from the source to the sink without violating the capacity of any edge. The maximum amount of flow that can be so routed is called the *max-flow*.[1] The *min-cut* is the minimum amount of capacity that needs to be removed from the network in order to disconnect the source from the sink. More formally, the min-cut is

$$\min_{\{U \subset V | s \in U, t \in \bar{U}\}} \sum_{e \in \langle U, \bar{U} \rangle} C(e)$$

where U is any subset of V that contains the source but not the sink, $\bar{U} = V - U$ is the set of nodes not in U, and $\langle U, \bar{U} \rangle$ denotes the set of edges that link a node in U to a node in $\bar{U}$. (The set of edges from any set U to $\bar{U}$ is referred to as a *cut* of the network since the removal of those edges separates U from the rest of the network.)

It is easy to see that the min-cut of a 1-commodity flow problem is always an upper bound on the max-flow. This is because for any $U \subseteq V$ that contains the source but not the sink, all flow from s to t must be routed through edges in $\langle U, \bar{U} \rangle$. Hence, the total flow is limited by the capacity in the min-cut. For example, see Figure 1. Showing that the min-cut bound is always achievable (i.e., that the max-flow equals the min-cut) is more difficult and remains as one of the earliest and most important advances in the field of algorithm design [Ford and Fulkerson 1956].

1.2. MULTICOMMODITY FLOW PROBLEMS. In a multicommodity flow problem, there are $k \geq 1$ commodities, each with its own source s_i, sink t_i, and demand D_i. The objective is to simultaneously route D_i units of commodity i from s_i to t_i for each i so that the total amount of all commodities passing through any edge is no greater than its capacity. (In the case of undirected edges, the sum of the flows in both directions cannot exceed the capacity of the edge.) For example, a solution to a 2-commodity flow problem is illustrated in Figure 2.

More generally, it is often useful to maximize the amount of flow that can be routed for each commodity. Because of the shared capacity constraints and the

[1] For a formal definition of max-flow, see Tarjan [1983].

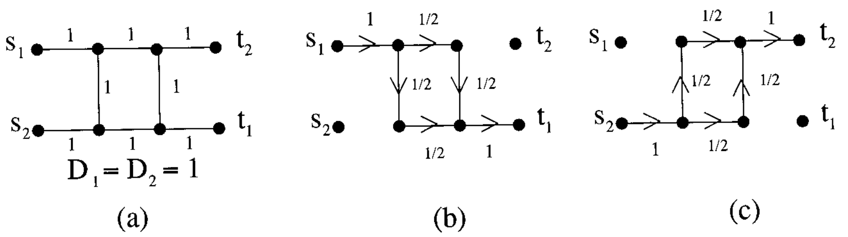

FIG. 2. Solution to a 2-commodity flow problem (a) in which all edge capacities are 1. The routing of the first commodity is shown in (b) and the second commodity is shown in (c).

multiplicity of commodities, there are several ways that one can define the max-flow for a multicommodity flow problem. In this paper, we will concentrate on a normalized definition of max-flow (known as *concurrent max-flow*) wherein we attempt to maximize a common fraction f of each commodity that is routed. In other words, the *max-flow* for a multicommodity flow problem is defined to be the maximum value of f such that fD_i units of commodity i can be simultaneously routed for each i without violating any capacity constraints. (For example, the max-flow for the 2-commodity flow problem in Figure 2 is one.) This commonly-used definition provides for a "fairness" property that ensures that proportionally more of one commodity will not be routed at the expense of another. The definition is also representative of other definitions in the sense that several of the techniques that we will derive in conjunction with our definition can also be used with alternative definitions.

The *min-cut* (a.k.a. *sparsest cut* $\mathcal{S}$) of a (undirected) multicommodity flow problem is defined to be the minimum over all cuts of the ratio of the capacity of the cut to the demand of the cut. More formally, the min-cut is

$$\mathcal{S} = \min_{U \subseteq V} \frac{C(U, \bar{U})}{D(U, \bar{U})},$$

where

$$C(U, \bar{U}) = \sum_{e \in \langle U, \bar{U} \rangle} C(e)$$

is the sum of capacities of the edges linking U to $\bar{U}$ and

$$D(U, \bar{U}) = \sum_{\{i | s_i \in U \wedge t_i \in \bar{U} \text{ or } t_i \in U \wedge s_i \in \bar{U}\}} D_i$$

is the sum of the demands whose source and sink are on opposite sides of the cut that separates U from $\bar{U}$. For example, the min-cut of the flow problem shown in Figure 2 is one. (Without loss of generality, we assume that the underlying graph is connected so that $C(U, \bar{U}) > 0$ for all U.)

It is not difficult to check that the definition of the min-cut for a multicommodity flow problem is a generalization of the definition given earlier in the special case of one commodity (where we assume that the demand of the single commodity is one without loss of generality). It is also not difficult to check that the max-flow is always upper bounded by the min-cut for any multicommodity flow problem. To see why, let $i_1, i_2, \ldots, i_r$ denote the commodities whose source and sink are separated by some cut $\langle U, \bar{U} \rangle$. Since all flow for these commodities must cross the cut $\langle U, \bar{U} \rangle$, we know that

$$\sum_{j=1}^{r} f D_{i_j} \le C(U, \bar{U}).$$

Since

$$\sum_{j=1}^{r} D_{i_j} = D(U, \bar{U}),$$

this means that

$$f \le \frac{C(U, \bar{U})}{D(U, \bar{U})}$$

and that the max-flow is upper bounded by the min-cut.

1.3. Prior Work on Max-Flows and Min-Cuts for Multicommodity Flow Problems. The relationship between the max-flow and min-cut of a multicommodity flow problem has been the subject of substantial interest since Ford and Fulkerson's famous result for 1-commodity flows. Hu [1963] showed that the max-flow and min-cut are always equal in the case of two commodities. More generally, by combining results of Lomonosov [1985] and Papernov [1990], Schrijver [1990] showed that if the graph formed by the set of demands (i.e., the graph with edges $\{s_i, t_i\}$) does not contain either three disjoint edges or a triangle and a disjoint edge, then the max-flow and min-cut are equal. The max-flow and min-cut are also known to be equal or near-equal for certain special types of flows in planar graphs (see Frank [1990] and Schrijver [1990] for a survey of such results).

In the case when there is a commodity for every pair of nodes and all demands are equal, Shahrokhi and Matula [1990] proved that the max-flow and min-cut are equal provided that the dual of the flow problem satisfies a certain cut condition (namely, that the edges with nonzero length in the dual form a cut of the underlying graph into three or fewer components). They also used information contained in the dual to design heuristics for finding small cuts in graphs.

The max-flow and min-cut are not always equal for all patterns or numbers of commodities, however. For example, Figure 3 illustrates a simple 4-commodity flow problem described in Okamura and Seymour [1981] for which the max-flow is 3/4 and the min-cut is 1.

For general networks, little was known about the relationship between the max-flow and the min-cut. In fact, for general networks, it was known only that the max-flow is within a factor of k of the min-cut since each commodity can be optimized separately using $1/k$ of the capacity of each edge. Unfortunately, this result is not very good for large numbers of commodities.

1.4. Our Max-Flow Min-Cut Results. In this paper, we establish a much tighter relationship between the max-flow and the min-cut for several important

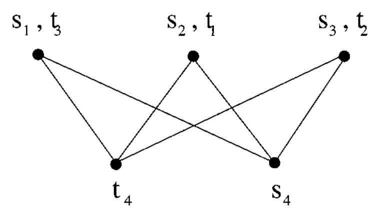

FIG. 3. The Okamura–Seymour [1981] example of a 4-commodity flow problem for which the max-flow is 3/4 and the min-cut is 1. In this example, all demands and capacities are one. The max-flow is attained by routing 1/4 unit of commodity 4 on each of the three paths between s_4 and t_4, and 3/8 unit of commodity i on each of the two paths between s_i and t_i for $1 \leq i \leq 3$.

classes of multicommodity flow problems. For the most part, we focus our attention on a special kind of flow problem that we call the uniform multicommodity flow problem. In a *uniform* multicommodity flow problem (UMFP), there is a commodity for every pair of nodes and the demand for every commodity is the same. (Without loss of generality, the demand for every commodity is set to one.)[2] The underlying network and capacities are arbitrary. As we will later see, uniform multicommodity flow problems arise in many important applications and their structure is sufficiently robust that the methods that we develop in this paper have already proven useful in the study of more general multicommodity flow problems [Klein et al. 1989; 1993; 1997; Garg et al. 1996; Plotkin and Tardos 1993].

Our primary result in this paper is an approximate max-flow min-cut theorem for uniform multicommodity flow problems. In particular, we show that for uniform multicommodity flow problems, the max-flow is within a $\Theta(\log n)$-factor of the min-cut.[3] We also show that this bound is tight in the sense that there exist uniform flow problems for which the max-flow is precisely a factor of $\Theta(\log n)$ smaller than the min-cut for any n. To our knowledge, these are the first results of their kind. In particular, the approximate max-flow min-cut theorem is the first such nontrivial result known to hold for flow problems with many commodities and arbitrary underlying networks. The examples for which the max-flow and min-cut differ by a $\Theta(\log n)$ factor also provide the first evidence that the separation between the max-flow and min-cut can be arbitrarily large.

We also prove approximate max-flow min-cut theorems for related classes of flow problems in which the edges in the underlying graph may be directed, the flow paths may be required to be short, and/or the constraint on the uniformity of the demands is relaxed. All of the results are constructive in the sense that we can convert any of the known polynomial-time algorithms for max-flow [Vaidya 1989; Kamuth and Palmon 1995] or approximate max-flow [Klein et al. 1994; Leighton et al. 1992; Awerbuch and Leighton 1994] into a polynomial-time algorithm for finding an approximate min-cut. As we will soon see, this fact has surprisingly powerful consequences in the domain of approximation algorithms.

1.5. APPLICATIONS TO APPROXIMATION ALGORITHMS. In a uniform multicommodity flow problem, the demand across a cut $\langle U, \bar{U} \rangle$ is simply the product of

[2] Equivalently, we can have two commodities for every pair of nodes u and v, with 1/2 unit of flow from u to v and 1/2 unit of flow from v to u in a uniform multicommodity flow problem.
[3] Throughout this paper, $\log n$ is used to denote $\log_2 n$ and $\ln n$ is used to denote $\log_e n$.

the number of nodes in U and the number of nodes in $\bar{U}$. In other words,

$$D(U, \bar{U}) = |U||\bar{U}|.$$

Hence, the min-cut of a uniform flow problem is

$$\mathcal{S} = \min_{U \subseteq V} \frac{C(U, \bar{U})}{|U||\bar{U}|}, \tag{1}$$

where $C(U, \bar{U}) = \Sigma_{e \in \langle U, \bar{U} \rangle} C(e)$. In the case when all capacities are 1, the min-cut is simply

$$\min_{U \subseteq V} \frac{|\langle U, \bar{U} \rangle|}{|U||\bar{U}|}. \tag{2}$$

The min-cut values in Eqs. 1 and 2 provide good measures of the number of edges (or the total weight of edges) that need to be removed in order to partition the underlying network into pieces of various sizes. As a consequence, we will be able to use our algorithm for finding an approximate min-cut to design the first nontrivial polynomial-time approximation algorithms for several important NP-hard graph partitioning problems.

In part because graph partitioning is itself a key component in many divide-and-conquer-based algorithms, we will also be able to use the flow result to design the first polynomial-time approximation algorithms for a wide variety of other well-known NP-hard optimization problems. For example, approximation algorithms for crossing number, VLSI layout, minimum feedback arc set, and search number will be described in the paper. Applications of the flow results to path routing problems, network reconfiguration, communication in distributed networks, rapidly mixing Markov chains, and scientific computing will also be described in the paper.

Altogether, the max-flow min-cut result has been used to provide the first polynomial time approximation algorithms for over two dozen optimization problems, many of which have no readily-apparent connection to multicommodity flow.

1.6. Subsequent Work. Several of the key results in this paper first appeared in an extended abstract published in 1988 [Leighton and Rao 1988]. Since that time, significant progress has been made on many of the problems studied in this paper. In what follows, we will give brief descriptions of and forward pointers to the most important subsequent advances in this area.

The methods described in this paper were first used by Klein et al. [1995] to establish an $O(\log C \log D)$ max-flow min-cut theorem for arbitrary multicommodity flow problems in undirected networks, where C denotes the total capacity of the network and D denotes the total demand of the commodities. (In other words, Klein et al. [1995] showed that the max-flow is always within an $O(\log C \log D)$ factor of the min-cut for undirected multicommodity flow problems.) This bound was improved to $O(\log^2 k)$ in a succession of papers by Tragoudas [1990], Plotkin and Tardos [1993], and Garg et al. [1996]. An existentially tight gap of $\Theta(\log k)$ was recently established by Linial et al. [1995] and Aumann and Rabani

[1998]. The latter result differs from its predecessors by its elegant use of Bourgain's techniques [Bourgain 1985] that embed metric spaces on graphs into geometric spaces.

In addition, Garg et al. [1996] established an $O(\log k)$ max-flow min-cut theorem for arbitrary (undirected) multicommodity flow problems where the sum of the flows is maximized (as opposed to our situation in which a common fraction of each commodity's demand is maximized). Klein et al. [1993] discovered a $\Theta(1)$ max-flow min-cut theorem for uniform multicommodity flow problems in undirected planar graphs and graphs with small excluded minors. Klein et al. [1997] and Even et al. [1998] have established an $O(\log^3 k)$ max-flow min-cut theorem for multicommodity flow problems in directed networks for which the demand from any node u to any node v is equal to the demand from v to u. Whether or not there is a polylog max-flow min-cut theorem for arbitrary directed multicommodity flow problems still remains as an interesting open question.

Additional applications of the max-flow min-cut results described in the paper have also been discovered. For example, Klein et al. [1995] used the flow results to design approximation algorithms for 2-CNF satisfiability. Agarwal et al. [1993] and Ravi et al. [1991] gave approximation algorithms for minimizing fill when solving sparse linear systems of equations, and register allocation. Garg et al. [1996] used their flow results to find an approximation algorithm for the minimum multicut problem. Some of these applications will be described in greater detail in Section 3 of the paper.

The construction used in Lemma 3 is similar in nature to the neighborhood cover constructions defined by Awerbuch and Peleg [1990]. Neighborhood covers have also proven to be very useful in the design of approximation algorithms for such problems as shortest paths [Awerbuch et al. 1999; Cohen 1993] finding separators in planar graphs [Rao 1992], and producing compact routing tables in distributed networks [Awerbuch and Peleg 1990].

For a survey of all the work on max-flow min-cut theorems and their applications to approximation algorithms, we refer the reader to the excellent article by Shmoys [1996].

1.7. ORGANIZATION OF THE PAPER. The remainder of the paper is organized as follows: The max-flow min-cut results are described in Section 2. Applications of these results to the design of approximation algorithms are described in Section 3. Remarks and open questions are included in Section 4. We conclude with acknowledgments and references in Sections 5 and 6, respectively.

2. *Max-Flow Min-Cut Theorems*

In this section, we prove max-flow min-cut theorems for several classes of multicommodity flow problems. We begin in Section 2.1 by giving an example of a uniform multicommodity flow problem (UMFP) for which the max-flow is a $\Omega(\log n)$-factor smaller than the min-cut.

Our main result is proved in Section 2.2, where we show that the max-flow is *always* within a $\Theta(\log n)$-factor of the min-cut for any UMFP. This result is generalized in Section 2.3, where we show that the max-flow is within a $\Theta(\log n)$-factor of the min-cut for any *product* multicommodity flow problem

(PMFP).[4] (In a PMFP, the demands between any pair of nodes u and v is $\pi(u)\pi(v)$ where π is a function on the nodes of the graph.) Our results are further extended to the case of directed graphs in Section 2.4 and to flow problems in which the flow paths are required to be short in Section 2.5.

2.1. A BAD EXAMPLE. We begin by showing that the max-flow and min-cut of a UMFP are separated by a $\Theta(\log n)$ gap whenever the underlying network has certain expansion properties. In particular, let G be a 3-regular n-node graph with unit edge capacities for which

$$|\langle U, \bar{U}\rangle| \geq c \ \min\{|U|, |\bar{U}|\}$$

for some constant $c > 0$ and all $U \subseteq V$. Families of such graphs are well known to exist provided that c is a sufficiently small constant [Margulis 1973]. (In fact, a randomly selected 3-regular graph will satisfy this property with high probability.)

By the definition of G, we know that the min-cut of the corresponding UMFP is

$$\begin{aligned}\mathcal{S} &= \min_{U\subseteq V} \frac{|\langle U, \bar{U}|}{|U||\bar{U}|}\\ &\geq \min_{U\subseteq V} \frac{c}{\max\{|U|, |\bar{U}|\}}\\ &= \frac{c}{n-1}.\end{aligned}$$

As we will show in what follows, however, the max-flow for the UMFP is at most $6/(n-1)(\log n - 1)$, which is a $\Theta(\log n)$-factor smaller than the min-cut.

Since G is 3-regular, there are at most $n/2$ nodes within distance $\log n - 3$ of any particular node $v \in V$. Hence, for at least half of the $\binom{n}{2}$ commodities, the shortest path connecting the source and sink in G has at least $\log n - 2$ edges. In order to sustain a flow of f for such a commodity, at least $f(\log n - 2)$ capacity must be used by the commodity. Thus, in order to sustain a flow of f for all $\binom{n}{2}$ commodities, the capacity in the network must be at least

$$\frac{1}{2}\binom{n}{2} f(\log n - 2).$$

Since the graph is 3-regular and has unit capacity edges, the total capacity is at most $3n/2$. Hence,

$$f \leq \frac{3n}{\binom{n}{2}(\log n - 2)}$$

[4] The fact that our techniques apply to this more general case was pointed out to us by Shmoys and Tardos, and Sinclair.

$$= \frac{6}{(n-1)(\log n - 2)}$$

$$\le \frac{6\mathcal{S}}{c(\log n - 2)}$$

$$= O\left(\frac{\mathcal{S}}{\log n}\right)$$

In other words, the max-flow for the UMFP on G is at least a $\Theta(\log n)$-factor smaller than the min-cut. This result is summarized in the following theorem.

THEOREM 1. *For any n, there is an n-node uniform multicommodity flow problem with max-flow f and min-cut $\mathcal{S}$ for which $f \le O(\mathcal{S}/\log n)$.*

The example described above can be generalized to nonexpander graphs by replacing each edge of G by a path of length p. In this manner, it is possible to construct UMFPs with a small min-cut but for which the $\Omega(\log n)$ gap between max-flow and min-cut still holds. The gap can never be greater than $\Theta(\log n)$, however, as we show in the next section.

2.2. FINDING SMALL CUTS IN UMFPS. The example in Section 2.1 demonstrates that there is a UMFP for which the min-cut is at least $\Theta(\log n)$ times as large as the max-flow. In this section we will prove that this example is worst-case in the sense that the min-cut of a UMFP can never be more than a $\Theta(\log n)$-factor larger than the max-flow. In fact, we will describe a polynomial-time algorithm that will find a cut $\langle U, \bar{U}\rangle$ for any UMFP for which

$$\frac{C(U, \bar{U})}{|U||\bar{U}|} \le O(f \log n), \tag{3}$$

where f is the max-flow of the UMFP. (Henceforth, we will refer to the quantity $C(U, \bar{U})/|U||\bar{U}|$ as the *ratio cost* of a cut $\langle U, \bar{U}\rangle$.) Since the min-cut of a UMFP is

$$\mathcal{S} = \min_{U \subseteq V} \frac{C(U, \bar{U})}{|U||\bar{U}|},$$

Eq. 3 results in the following theorem.

THEOREM 2. *For any uniform multicommodity flow problem,*

$$\Omega\left(\frac{\mathcal{S}}{\log n}\right) \le f \le \mathcal{S},$$

where f is the max-flow and $\mathcal{S}$ is the min-cut of the UMFP.

The algorithm for finding a cut with small ratio cost is based on the linear programming dual of the UMFP. In general, the *dual* of a multicommodity flow problem for a graph G is the problem of apportioning a fixed amount of weight (where weights are thought of as distances) to the edges of G so as to maximize the cumulative distance between the source/sink pairs. Alternatively, the dual can be thought of as apportioning the smallest amount of total distance so that the cumulative distances between the source/sink pairs is not too small. More precisely, the dual of a k-commodity flow problem consists of finding a nonnegative distance $d(e)$ for each edge $e \in E$ so that

$$\sum_{i=1}^{k} D_i d(s_i, t_i) \geq 1 \tag{4}$$

and so that

$$\sum_{e \in E} C(e) d(e)$$

is minimized, where $d(s_i, t_i)$ is the distance between the source and sink for the ith commodity in G with respect to the distance function. (The proof of this result is based on the standard notion of duality from linear programming [Chvatal 1983]. The proof was originally discovered by Iri [1967] and subsequently observed by Shahrokhi and Matula [1986].)

In what follows, we will refer to Eq. 4 as the *distance constraint* of the dual. In the case of a uniform multicommodity flow problem, the distance constraint is simply

$$\sum_{u,v \in V} d(u, v) \geq 1, \tag{5}$$

where the sum is taken over all unordered pairs of nodes in G. We will also refer to

$$W = \sum_{e \in E} C(e) d(e)$$

as the *total weight* of the distance function. From the duality theory of linear programming, we know that an optimal distance function results in a total weight that is equal to the max-flow of the UMFP. Hence, by solving the dual UMFP, we can find distances $d(e)$ that satisfy Eq. 5 and for which $W = f$. This is the first step toward finding a cut with small ratio cost.[5]

The fact that our algorithm uses the dual UMFP to find a small cut should not be completely surprising, given the well-known relationship between the min-cut and the dual of a single commodity flow problem. Indeed, in one optimal solution for the dual of a single-commodity flow problem, the edges in the

[5] Although linear programming problems can be solved in polynomial time, linear programming is not very fast for large multicommodity flow problems in practice. Thus, Shahrokhi and Matula proposed approximation algorithms that seemed promising in experiments. Subsequently, researchers have discovered provably fast methods for finding a near-optimal distance function. For example, Klein et al. [1994] and Leighton et al. [1992] describe fast polynomial-time algorithms for finding a distance function for which $W \leq (1 + \varepsilon) f$ for arbitrarily small $\epsilon > 0$. Such approximation algorithms are quite suitable for our purposes (even with $\epsilon > 1$) since we will only endeavor to find cuts with ratio cost $O(W \log n) = O(f \log n)$.

min-cut are assigned distance 1 and all other edges are assigned distance 0. A similar relationship between the min-cut and nonzero distance edges also holds for certain very special UMFPs. For example, consider the UMFP where G consists of two copies of $K_{n/2}$ that are connected by a single edge e_0 and where all edges have unit capacity. In this example, the edge e_0 is assigned distance $4/n^2$ and all other edges are assigned distance 0 in the optimal solution to the dual. The min-cut is simply the edge e_0.

Unfortunately, the relationship between the min-cut and the non-zero distance edges appears to break down for general UMFPs. Although edges that are assigned large distances in the dual tend to be included in good cuts for many examples (see Shahrokhi and Matula [1990]), it does not appear as though a simple thresholding type of procedure can be relied upon to always find a good cut.

In what follows, we describe a different approach that is guaranteed to find a good cut with ratio cost at most $O(W \log n) = O(f \log n)$. Although the algorithm will tend to place edges with large distance in the cut, the distance that is assigned to an edge is not the primary factor in deciding whether to place the edge in the cut. Rather, we rely on a more global property of the distance function that is related to the problem of decomposing a graph into regions of small radius. This property is captured in the following lemmas.

LEMMA 3. *For any graph G with arbitrary edge capacities, any $\Delta > 0$, and any distance function with total weight W it is possible to partition G into components with radius at most Δ so that the capacity of the edges connecting nodes in different components is at most $4W \log n/\Delta$.*

PROOF. Let $C = \Sigma_{e \in E} C(e)$ denote the *total capacity* on the edges of G. When $\Delta \le 4W \log n/C$, we can use the partition that leaves every node in a different component. Each such component will have radius $0 \le \Delta$ and the capacity of the edges running between different components is at most $C \le (4W \log n/\Delta)$, as desired.

If $\Delta > (4W \log n/C)$, then we construct a second graph G^+ from G by replacing each edge e of G with a path of $\lceil Cd(e)/W \rceil$ edges. Each edge along the path is assigned capacity $C(e)$ and distance 1. In what follows, we will show how to partition G by forming components in G^+. (The reason for using G^+ is that it is simpler to work with a graph in which each edge corresponds to the same amount of distance.)

The components of G^+ are formed as follows: We begin by selecting an arbitrary node $v \in G^+$ that corresponds to a node in G. For each $i \ge 0$, define G_i^+ to be the subgraph of G^+ consisting of nodes and edges within distance i of v. (For G^+, the distance between two nodes is defined to be the number of edges that are traversed in the shortest path connecting the nodes.) Let $C_0 = (2C/n)$, and for $i > 0$, define C_i to be the total capacity of the edges in G_i^+. Let j denote the smallest value of $i \ge 0$ for which $C_{i+1} < (1 + \epsilon)C_i$ where $\epsilon = (W \log n/\Delta C) < 1/4$. (There must be such a j since for large enough i, $G_{i+1}^+ = G_i^+$.)

The nodes and edges in G_j^+ form the first component of the partition. The remaining components are found by removing G_j^+ from G^+ and then repeating the entire process. The process is repeated until there are no longer any nodes $v \in G^+$ that correspond to nodes in G.

Let C^+ denote the total initial capacity of G^+. From the construction of G^+, we know that

$$\begin{aligned} C^+ &= \sum_{e \in E} C(e) \left\lceil \frac{Cd(e)}{W} \right\rceil \\ &\le \sum_{e \in E} C(e) + \frac{C}{W} \sum_{e \in E} C(e)d(e) \\ &= 2C. \end{aligned}$$

From the method by which the components were formed, we know that the capacity of the edges leaving any component is at most ϵC_j, where $C_j = (2C/n)$ if the component consists of a single node and C_j is the sum of the capacities on the edges contained in G_j^+, otherwise. Since the G_j^+ are disjoint, this means that the total capacity on all edges leaving all components in G^+ is at most

$$\epsilon(C^+ + nC_0) \le 2\epsilon C^+ 2\epsilon C = 4\epsilon C. \tag{6}$$

The partition for G is derived from the components of G^+ in the natural way. In particular, two nodes of G are placed in the same component for G if and only if they were in the same component for G^+. From the construction of the components, we thus know that any edge $e \in G$ that links two components in G must correspond to a path of capacity $C(e)$ edges in G^+ that was cut to form at least one of the corresponding components in G^+. Hence, the total capacity of the edges linking different components in G is at most $4\epsilon C = (4W \log n/\Delta)$, as desired.

It remains only to show that each component has small radius. This can be verified as follows: Consider a component with radius j in G^+. Provided that $j > 0$, this component must have total capacity at least $(1 + \epsilon)^j C_0 = (1 + \epsilon)^j C_0 \epsilon$ $(1 + \epsilon)^j (2C/n)$. Since the total capacity of G^+ is $C^+ \le 2C$, this means that

$$(1 + \epsilon)^j \frac{2C}{n} \le 2C$$

and thus (since $\epsilon < 1/4$) that

$$j \le \frac{\log n}{\log(1 + \epsilon)} \le \frac{\log n}{\epsilon}.$$

Given a path of length l in G^+, the corresponding path in G has length at most Wl/C. Hence, the radius of each component in G is at most

$$\frac{W \log n}{C\epsilon} = \Delta,$$

as desired. □

COROLLARY 4. *For any graph G and any distance function with total weight W, we can either*

(1) *find a component with radius $1/2n^2$ that contains at least 2/3 of the nodes in G, or*

(2) *find a cut of G with ratio cost $O(W \log n)$.*

PROOF. We apply the result of Lemma 3 with $\Delta = 1/2n^2$. If one of the components formed during the construction of Lemma 3 contains at least 2/3 of the nodes of G, then we are done. Otherwise, we can divide the components into two sets so that each set contains at least $n/3$ nodes. This division forms a cut with edge capacity at most

$$4W \log n/\Delta = 8Wn^2 \log n.$$

Since both sides of the cut have at least $n/3$ nodes, the ratio cost of this cut is at most

$$\begin{aligned}\frac{8wn^2 \log n}{(2n/3)(n/3)} &= 36W \log n \\ &= O(w \log n),\end{aligned}$$

as desired. □

LEMMA 5. *For any graph G, if there is a distance function d with total weight W and a subset of nodes $T \subseteq V$ with $|T| \geq 2n/3$ and*

$$\sum_{u \in V-T} d(T, u) \geq \frac{1}{2n},$$

then we can find a cut with ratio cost $O(W)$.

PROOF. The proof uses many of the same arguments that were used to prove Lemma 3. In particular, we start by defining the graph G_i^+ to be the subgraph of G^+ consisting of all nodes and edges that are within distance i of a node in T. (Recall that distance in G^+ is measured in terms of the number of edges that are traversed since every edge has distance 1 in G^+.) We also define V_i to be the set of nodes of G that correspond to nodes in G_i^+, $n_i = |V - V_i|$, R_i to be the ratio cost of the cut $\langle V_i, V - V_i \rangle$ in G, and $R = \min\{R_i\}$.

Since n_i nodes of G are at distance at least $i + 1$ from T in G^+ for all i, we know that

$$\sum_{u \in V-T} d_{G^+}(T, u) = \sum_{i \geq 0} n_i.$$

By the construction of G^+, we also know that $d_{G^+}(T, u) \geq (C/W) d_G(T, u)$. Hence, we can conclude that

$$\sum_{i \geq 0} n_i \geq \frac{C}{W} \sum_{u \in V-T} d_G(T, u) \geq \frac{C}{2nW}. \tag{7}$$

Since $|T| \geq 2n/3$, the capacity in the cut $\langle V_i, V - V_i \rangle$ of G is at least $R_i n_i (2n/3) \geq (2nRn_i/3)$. Hence, the capacity of the corresponding cut for G_i^+

in G^+ is also at least this amount. Since the total capacity in G^+ is at most $C^+ \leq 2C$ (from the proof of Lemma 3), this means that

$$\sum_{i \geq 0} \frac{2nRn_i}{3} \leq 2C$$

and thus that

$$R \leq \frac{3C}{n \sum_{i \geq 0} n_i}.$$

Plugging in the lower bound from Eq. 7, we find that

$$R \leq 6W = O(W),$$

as desired. □

LEMMA 6. *Given a graph G and a distance function with total weight W that satisfies the distance constraint, we can find a cut with ratio cost $O(W \log n)$.*

PROOF. We begin by partitioning the graph as in Lemma 3 with $\Delta = 1/2n^2$. By Corollary 4, we can then either find a cut with ratio cost $O(W \log n)$ (in which case, we are done) or we can find a component T with radius $1/2n^2$ that contains at least $2n/3$ nodes. In the latter case, we will apply Lemma 5 to find a cut with ratio cost $O(W)$ (thereby concluding the proof), but we must first show that

$$\sum_{u \in V-T} d(T, u) \geq \frac{1}{2n}.$$

The proof makes use of the fact that for any pair of nodes $u, v \in V$

$$d(u, v) \leq d(T, u) + d(T, v) + \frac{1}{n^2}$$

since T has radius $1/2n^2$. In particular,

$$\begin{aligned}
\sum_{\{u,v\}} d(u, v) &= \frac{1}{2} \sum_{(u,v)} d(u, v) \\
&\leq \frac{1}{2} \sum_{(u,v)} \left(d(T, u) + d(T, v) + \frac{1}{n^2} \right) \\
&< n \sum_{u \in V-T} d(T, u) + \frac{1}{2}.
\end{aligned}$$

Because of the distance constraint, this means that

$$\sum_{u \in V-T} d(T, u) > \frac{1}{2n},$$

as desired. □

The proof of Theorem 2 follows immediately from Lemma 6 and the fact that $W = f$. (In fact, the result follows even if an approximate distance function with total weight $W = O(f)$ is used.) It is worth noting that although high-distance edges in G are not necessarily placed in the cut with small ratio cost, they are more likely to be placed in the cut since they correspond to long paths in G^+. It is also worth noting that each of the proofs in this section can be easily adapted to yield simple polynomial time algorithms for finding cuts and regions with low diameter.

2.3. Relaxing the Uniformity of Demands. The results of the preceding section can be easily generalized to work for more general multicommodity flow problems. For example, in this section, we show how to prove analogous results for *product* multicommodity flow problems (PMFPs).[6] In a PMFP, we associate a nonnegative weight $\pi(u)$ with each node $u \in V$. The demand for the commodity between nodes u and v is then set to be $\pi(u)\pi(v)$.[7] The UMFP is a special case of a PMFP for which $\pi(u) = 1$ for all nodes $u \in V$.

In what follows, we use $\mathcal{P}$ to denote the subset of nodes for which π is nonzero, and we set $p = |\mathcal{P}|$. Without loss of generality, we will assume that $\Sigma_{u \in V} \pi(u) = p$ for each PMFP.

The min-cut for a PMFP is

$$\min_{U \subseteq V} \frac{C(U, \bar{U})}{\pi(U)\pi(\bar{U})},$$

where $\pi(U) = \Sigma_{u \in U} \pi(u)$. In what follows, we will show how to find a cut $\langle U, \bar{U} \rangle$ for which the *weighted ratio cost*

$$\frac{C(U, \bar{U})}{\pi(U)\pi(\bar{U})}$$

is at most $O(f \log p)$ where f is the value of the maximum flow for the PMFP. Since the number of commodities with nonzero demand is $k = \binom{p}{2}$, this will be sufficient to prove the following theorem.

Theorem 7. *For any product multicommodity flow problem with k commodities*,

$$\Omega\left(\frac{\mathcal{S}}{\log k}\right) \le f \le \mathcal{S},$$

[6] The fact that our techniques apply to this more general case was pointed out to us by Shmoys and Tardos, and Sinclair.
[7] Equivalently, we can have two commodities for each pair of nodes u and v with demand $\frac{1}{2}\pi(u)\pi(v)$ from u to v and demand $\frac{1}{2}\pi(u)\pi(v)$ from v to u.

where f is the max-flow and $\mathcal{S}$ is the min-cut of the PMFP.

The algorithm for finding a cut with small weighted ratio cost is quite similar to the algorithm for UMFPs described in Section 2.2. In particular, the algorithm is based on a distance function formed from the dual of the PMFP. In this case, however, the distance constraint from Eq. 4 becomes

$$\sum_{\{u,v\}\in\mathcal{P}^2} \pi(u)\pi(v)d(u,v) \geq 1.$$

In what follows, we will briefly explain the other changes that need to be made in Lemmas 3–6 to find a cut with small weighted ratio cost.

LEMMA 8. *For any graph G, any $\Delta \geq 0$, any distance function with total weight W, and any set of p nodes $\mathcal{P}$ with nonzero node weight, it is possible to partition G into components so that*

(1) any component containing a node of $\mathcal{P}$ has radius at most Δ, and
(2) the capacity of the edges linking nodes in different components is at most $4W \log p/\Delta$.

PROOF. The proof is identical to that of Lemma 3, except that:

(1) n is replaced by p everywhere,
(2) the components G_j^+ are grown only from nodes in $\mathcal{P}$, and
(3) we only compute the bound on radius for components containing a node from $\mathcal{P}$. □

COROLLARY 9. *For any graph G, any distance function with total weight W, and any set $\mathcal{P}$ of p nodes with nonzero node weight, we can either*

(1) *find a component T with radius $1/2p^2$ for which $\pi(T) \geq 2p/3$, or*
(2) *find a cut of G with weighted ratio cost $O(W \log p)$.*

PROOF. The proof follows from Lemma 8 in a manner analogous to the proof of Corollary 4. □

LEMMA 10. *For any node-weighted graph G, if there is a distance function d with total weight W and a subset of nodes $T \subseteq V$ for which $\pi(T) \geq 2p/3$ and*

$$\sum_{u\in\mathcal{P}-T} \pi(u)d(T,u) \geq \frac{1}{2p},$$

then we can find a cut with ratio cost $O(W)$.

PROOF. The proof is identical to that of Lemma 5 except that

(1) measures of the number of nodes such as n_i and $|T|$ are replaced by the weight of those nodes (such as $\pi(V - V_i)$ and $\pi(T)$),
(2) we define R_i to be the *weighted* ratio cost of the cut $\langle V_i, V - V_i\rangle$ in G, and
(3) $d_G(T,u)$ and $d_{G^+}(T,u)$ are scaled by a factor of $\pi(u)$. □

LEMMA 11. *Given a node-weighted graph G and a distance function with total weight W that satisfies the (weighted) distance-constraint, we can find a cut with weighted ratio cost $O(W \log p)$.*

PROOF. Similar to the proof of Lemma 6. In particular, we first partition the graph as in Lemma 8 with $\Delta = 1/2p^2$. By Corollary 9, we can then either find a cut with weighted ratio cost $O(W \log p)$ (in which case we are done) or we can find a component T with radius $1/2p^2$ for which $\pi(T) \geq 2p/3$. In the latter case, we apply Lemma 10 to find a cut with weighted ratio cost $O(W)$ (thereby concluding the proof), but we must first show that

$$\sum_{u \in \mathcal{P}-T} \pi(u) d(T, u) \geq \frac{1}{2p}.$$

Arguing as in Lemma 6, we find that

$$\begin{aligned}\sum_{\{u,v\} \in \mathcal{P}^2} \pi(u)\pi(v)d(u, v) &= \frac{1}{2} \sum_{u \neq v} \pi(u)\pi(v)d(u, v) \\ &\leq \frac{1}{2} \sum_{u \neq v} \pi(u)\pi(v)\left(d(T, u) + d(T, v) + \frac{1}{p^2}\right) \\ &< p \sum_{u \in \mathcal{P}-T} \pi(u)d(T, u) + \frac{1}{2}.\end{aligned}$$

(Here we have used the facts that $\Sigma_{u \in V} \pi(u) = p$ and $\Sigma_{u \neq v} \pi(u)\pi(v) < p^2$.) Because of the distance constraint, this means that

$$\sum_{u \in \mathcal{P}-T} \pi(u) d(T, u) > \frac{1}{2p},$$

as desired. □

2.4. DEALING WITH DIRECTED GRAPHS. The results of Sections 2.2 and 2.3 can also be extended to hold for directed multicommodity flow problems. In a *directed* MFP, each edge has a specified direction, and flow is restricted to move only in the direction of each edge.

In a directed UMFP, we will assume that the demand from u to v is 1 for each $u \neq v$. (Hence, there is a flow of 1 from u to v as well as from v to u.) For a directed PMFP, the demand from u to v is $\pi(u)\pi(v)$, although we will restrict our attention to the case of UMFPs for the sake of simplicity in what follows.

The min-cut of a directed UMFP is defined to be

$$\min_{U \subseteq V} \frac{C(U, \bar{U})}{|U||\bar{U}|},$$

where $C(U, \bar{U})$ is defined to be the sum of the capacities of the edges in the cut $\langle U, \bar{U}\rangle$ that are directed from U to $\bar{U}$. The max-flow and min-cut of a directed UMFP are related by the following theorem.

THEOREM 12. *For any directed UMFP with n nodes,*

$$\Omega\left(\frac{\mathcal{S}}{\log n}\right) \leq f \leq \mathcal{S},$$

where f is the max-flow and $\mathcal{S}$ is the min-cut of the UMFP.

The fact that $f \leq \mathcal{S}$ follows immediately from the definitions. The algorithm for finding a directed cut $\langle U, \bar{U}\rangle$ with small ratio cost is somewhat more complicated than the algorithm for UMFPs described in Section 2.2, however. The main difference is that we need to keep track of two different notions of radius, one using edges directed *inwards* towards a common root and one using edges directed *outwards* from a common root. In particular, we define $\mathcal{N}^{\Delta}_{\text{in}}(v, G)$ to be the set of nodes from which v can be reached by a directed path of length at most Δ in G and $\mathcal{N}^{\Delta}_{\text{out}}(v, G)$ to be the set of nodes that are reachable from v by a directed path of length at most Δ in G.

LEMMA 13. *For any directed graph $G = (V, E)$ with arbitrary edge capacities, any $\Delta > 0$, and any distance function with total weight W, it is possible to partition V into subsets $V_1, V_2, \ldots, V_r$ so that*

(1) *the total capacity of all edges in the set $\{(u, v) | u \in V_i,\ v \in V_j,\ 1 \leq j < i < r\}$ is at most $8W \log n/\Delta$, and*
(2) *for each $i \leq r$, there is a node $v \in V_i$ for which $|\mathcal{N}^{\Delta}_{\text{in}}(v, G)| \geq |V_i|$ and $|\mathcal{N}^{\Delta}_{\text{out}}(v, G)| \geq |V_i|$.*

PROOF. The proof is similar to that of Lemma 3. In particular, C, G^+, and $C^+ \leq 2C$ are defined as before, and we assume without loss of generality that $\Delta \geq (8W \log n/C)$. The sets $V_1, V_2, \ldots, V_r$ are formed as follows: We begin by selecting an arbitrary node $v \in G^+$ that corresponds to a node in G. For each $i \geq 0$, define $G^+_{i,\text{in}}$ (respectively, $G^+_{i,\text{out}}$) to be the subgraph of G^+ induced by $\mathcal{N}^{i}_{\text{in}}(v, G^+)$ (respectively, $\mathcal{N}^{i}_{\text{out}}(v, G^+)$). In other words, $G^+_{i,\text{in}}$ is the subgraph of G^+ induced by the nodes from which v can be reached by a directed path of length at most i in G^+.

Let $C_0 = (2C/n)$, and for $i > 0$, define $C_{i,\text{in}}$ to be the total capacity of the edges in $G^+_{i,\text{in}}$ and $C_{i,\text{out}}$ to be the total capacity of the edges in $G^+_{i,\text{out}}$. Let j denote the smallest i for which either

Case 1. $G^+_{i,\text{in}}$, has no more nodes of G than $G^+_{i,\text{out}}$ and $C_{i+1,\text{in}} < (1 + \epsilon)C_{i,\text{in}}$, or

Case 2. $G^+_{i,\text{out}}$ has no more nodes of G than $G^+_{i,\text{in}}$ and $C_{i+1,\text{out}} < (1 + \epsilon)C_{i,\text{out}}$, where $\epsilon = (2W \log n/\Delta C) \leq 1/4$.

In the event of Case 1, we define U^+_1 to be the set of nodes in $G^+_{j,\text{in}}$ and we say that U^+_1 is an *in-set.* In the event of Case 2, we define U^+_1 to be the set of nodes in $G^+_{j,\text{out}}$ and we say that U^+_1 is an *out-set.* In either event, we remove U^+_1 and all edges incident to a node in U^+_1 from G^+. The process is then repeated, creating

$U_2^+, U_3^+, \ldots, U_r^+$, until there are no longer any nodes in G^+ that correspond to nodes in G.

Once the process is terminated, the sets $U_1^+, U_2^+, \ldots, U_r^+$ are reordered to produce $V_1^+, V_2^+, \ldots, V_r^+$ as follows:

(1) all in-sets precede all out-sets in $V_1^+, V_2^+, \ldots, V_r^+$,
(2) the relative order of in-sets in $U_1^+, U_2^+, \ldots, U_r^+$ is preserved in $V_1^+, V_2^+, \ldots, V_r^+$,
(3) and the relative order of out-sets in $U_1^+, U_2^+, \ldots, U_r^+$ is reversed in $V_1^+, V_2^+, \ldots, V_r^+$.

(For example, if U_1^+ is an in-set, then $V_1^+ = U_1^+$. If U_1^+ is an out-set, then $V_r^+ = U_1^+$.) We then define $V_i = V \cap V_i^+$ for $1 \le i \le r$. In what follows, we will show that $V_1, V_2, V_3, \ldots, V_r$ have the properties claimed in the statement of the Lemma. The proof is similar to that for Lemma 3.

We say that an edge of G^+ is a *boundary* edge if it has one endpoint in V_i^+ and the other endpoint outside V_i^+ for some i. Moreover, we say that a boundary edge is *relevant* if either

(1) the head of the edge is contained in some in-set V_j^+ and the edge was present in G^+ when V_j^+ was being created or
(2) the tail of the edge is contained in some out-set V_i^+ and the edge was present in G^+ when V_i^+ was being created.

Arguing as in the proof of Lemma 3, we can conclude that the capacity of all relevant boundary edges is at most

$$\epsilon(C^+ + nC_0) \le 4\epsilon C.$$

Because of the way that $U_1^+, U_2^+, \ldots, U_r^+$ were reordered to form $V_1^+, V_2^+, \ldots, V_r^+$, we also know that for every edge $e \in \{(u, v) | u \in V_i, v \in V_j, 1 \le j < i \le r\}$, we can identify a distinct *relevant* boundary edge in G^+ with the same capacity as e. Hence, the total capacity of the edges in $\{(u, v) | u \in V_i, v \in V_j, 1 \le j < i \le r\}$ is at most $4\epsilon C = (8W \log n/\Delta)$, as desired.

We next show that for each $i \le r$, there is a node $v \in V_i$ for which both $|\mathcal{N}_{\text{in}}^{\Delta}(v, G)|$ and $|\mathcal{N}_{\text{out}}^{\Delta}(v, G)|$ are at least $|V_i|$. We do this by upper-bounding the radius of the subgraph of G^+ that was used to create V_i^+. For simplicity, we will consider only the case when V_i^+ is an in-set. (A symmetric analysis can be used when V_i^+ is an out-set.)

Let $G_{j,\text{in}}^+$ denote the radius j subgraph of G^+ that was removed from G^+ to form V_i^+. By the construction of $G_{j,\text{in}}^+$, we know that for all $0 \le s < j$

(1) $C_{s+1,\text{in}} \ge C_{s,\text{in}}$ and $C_{s+1,\text{out}} \ge C_{s,\text{out}}$ and
(2) $C_{s+1,\text{in}} \ge (1 + \epsilon)C_{s,\text{in}}$ or $C_{s+1,\text{out}} \ge (1 + \epsilon)C_{s,\text{out}}$.

Define $C_j = \max\{C_{j,\text{in}}, C_{j,\text{out}}\}$. By the preceding facts, we know that

$$\begin{aligned} C_j &\ge (1 + \epsilon)^{j/2} C_0 \\ &= (1 + \epsilon)^{j/2} 2C/n. \end{aligned}$$

Also, $C_j \le C^+ \le 2C$, and so

$$j \le \frac{2 \log n}{\epsilon}.$$

Let v denote the first node that was placed in $G_{j,\mathrm{in}}$. By the preceding analysis, we know that

$$V_i^+ \subseteq \mathcal{N}_{\mathrm{in}}^{(2 \log n)/\epsilon}(v, G^+).$$

Given a path of length l in G^+, the corresponding path in G has length at most Wl/C. Since $W/C(2 \log n/\epsilon) = \Delta$, this means that

$$v_i \subseteq \mathcal{N}_{\mathrm{in}}^{\Delta}(v, G),$$

and thus that $|\mathcal{N}_{\mathrm{in}}^{\Delta}(v, G)| \ge |V_i|$.

Since $G_{j,\mathrm{in}}^+$ has no more nodes of G than $G_{j,\mathrm{out}}^+$ by construction, we also know that $|\mathcal{N}_{\mathrm{out}}^{\Delta}(v, G)| \le |V_i|$, as desired. $\square$

COROLLARY 14. *For any directed graph G and any distance function with total weight W, we can either*

(1) *find a node v for which $|\mathcal{N}_{\mathrm{in}}^{1/4n^2}(v, G)| \ge 2n/3$ and $|\mathcal{N}_{\mathrm{out}}^{1/4n^2}(v, G)| \ge 2n/3$, or*
(2) *find a directed cut of G with ratio cost $O(W \log n)$.*

PROOF. We apply the result of Lemma 13 with $\Delta = 1/4n^2$. If one of the sets V_i in the partition of Lemma 13 has at least $2n/3$ nodes, then we are done. Otherwise, we can find an $s < r$ such that

$$\frac{n}{6} \le |V_1 \cup V_2 \cup \cdots \cup V_s| \le \frac{5n}{6}.$$

The desired directed cut is $\langle V_{s+1} \cup \cdots \cup V_r, V_1 \cup \cdots \cup V_s \rangle$. By Lemma 13, the ratio cost of this cut is at most

$$\frac{8W \log n}{\Delta(n/6)(5n/6)} \le O(W \log n),$$

as desired. $\square$

LEMMA 15. *For any graph G and distance function d with total weight W, if there is a subset of nodes $T \subseteq V$ for which $|T| \ge 2n/3$ and either*

$$\sum_{u \in V-T} d(T, u) \ge \frac{1}{4n}$$

or

$$\sum_{u \in V-T} d(u, T) \ge \frac{1}{4n},$$

then we can find a directed cut with ratio cost $O(W)$.

PROOF. The proof is virtually identical to that of Lemma 5. In the case when

$$\sum_{u \in V-T} d(T, u) \geq \frac{1}{4n},$$

we consider edges directed away from T. In the case when $\Sigma_{u \in V-T} d(u, T) \geq 1/4n$, we consider edges directed towards T. □

LEMMA 16. *Given a directed graph G and a distance function with total weight W that satisfies the distance constraint, we can find a directed cut with ratio cost $O(W \log n)$.*

PROOF. By Corollary 14, we can either find the desired cut or a node v for which $|\mathcal{N}_{\text{in}}^{1/4n^2}(v, G)| \geq 2n/3$ and $|\mathcal{N}_{\text{out}}^{1/4n^2}(v, G)| \geq 2n/3$.

Let $T_{\text{in}} = \mathcal{N}_{\text{in}}^{1/4n^2}(v, G)$ and $T_{\text{out}} = \mathcal{N}_{\text{out}}^{1/4n^2}(v, G)$. If $\Sigma_{u \in V-T_{\text{in}}} d(u, T_{\text{in}}) \geq 1/4n$ or $\Sigma_{u \in V-T_{\text{out}}} d(T_{\text{out}}, u) \geq 1/4n$, then we can find a cut of size $O(W)$ by Lemma 15. One of these conditions must hold since otherwise

$$\begin{aligned}
\sum_{(u,v) \in V^2} d(u, v) &< \sum_{u \in V} \sum_{v \in V} \left(d(u, T_{\text{in}}) + d(T_{\text{out}}, v) + \frac{1}{2n^2} \right) \\
&= n \sum_{u \in V} d(u, T_{\text{in}}) + n \sum_{v \in V} d(T_{\text{out}, v}) + \frac{1}{2} \\
&< \frac{1}{4} + \frac{1}{4} + \frac{1}{2} \\
&= 1,
\end{aligned}$$

which violates the distance constraint. □

The proof of Theorem 12 follows immediately from Lemma 16. The result can also be easily extended to directed PMFPs by using the techniques developed in Section 2.3. We state this result without proof as Theorem 17.

THEOREM 17. *For any directed product multicommodity flow problem with k commodities,*

$$\Omega\left(\frac{\mathcal{S}}{\log k} \right) \leq f \leq \mathcal{S},$$

where f is the max-flow and $\mathcal{S}$ is the directed min-cut of the PMFP.

2.5. FLOWS WITH SHORT PATHS. Thus far, we have focused on finding cuts with small ratio cost in graphs. As a consequence of this work, we have found that the max-flow of a UMFP or PMFP is nearly as large as the limit implied by the min-cut. In this section, we will show that the max-flow of a UMFP or PMFP is large even if the flow paths are restricted to have short length in the underlying graph. This result will be very useful later when we describe algorithms for routing paths with low congestion and dilation in communication networks. In particular, we will make use of the following theorem.

THEOREM 18. *Given, any n-node uniform multicommodity flow problem for which the min-cut has size $\mathcal{S}$, we can find a flow of size $f \geq \Omega(\mathcal{S}/\log n)$ for which every flow path has length at most $L \leq O(C_{\max}\log n/n\mathcal{S})$, where $C_{\max}$ is the maximum total capacity of the edges incident to any single node.*

The fact there is a flow of size $\Omega(\mathcal{S}/\log n)$ follows from Theorem 2. The fact that the flow can be routed using only short flow paths involves some additional work. In what follows, we will show that such a flow exists. The flow itself can be found using linear programming or other polynomial-time algorithms.

To formulate the flow problem as a linear program, we use a separate variable for each demand, each edge, and each distance that the flow has traveled through the network. We will then maximize the amount of flow that has traveled distance at most $L = \Theta(C_{\max}\log n/n\mathcal{S})$. Although we do not know the precise value of L, a good approximation can be obtained by computing f or using binary search.

The dual of the short-path UMFP is the same as before except that the distance between two nodes in the distance constraint is computed using only paths with at most L edges in G. (Note that it is possible for the shortest path between two nodes using the distance metric to use more than L edges of G. Such paths are not allowed in this case. As a consequence, the restricted distance between two nodes in the dual may be infinite.)

The proof that the short-path flow exceeds $\Omega(\mathcal{S}/\log n)$ is similar to the proof of Theorem 2. The main difference is that we now need to keep track of path lengths according to two metrics: the distance measure and the number of edges. In addition, we only need an existence proof and so it will suffice to show that $W \geq \Omega(\mathcal{S}/\log n)$ for any distance function that satisfies the distance constraint. We begin with a strengthening of Lemma 3.

LEMMA 19. *For any graph G with total capacity C, any $\Delta > 0$ and any distance function with total weight W, it is possible to partition G into components with edge-radius $\Delta C/W$ and distance-radius Δ so that the capacity of the edges connecting nodes in different components is at most $4W \log n/\Delta$.*

PROOF. The proof is identical to that of Lemma 3. We need only observe that the edge-radius of a component of G is upper bounded by the radius of a component of G^{+}, which is at most $(\log n/\epsilon) = (\Delta C/W)$. □

COROLLARY 20. *For any graph G with min-cut $\mathcal{S}$ and any distance function with total weight $W \leq \mathcal{S}/36 \log n$, we can find a component of G with edge-radius $C/2Wn^2$ and distance-radius $1/2n^2$ that contains at least* 2/3 of the nodes of G.

PROOF. The proof is nearly identical to that of Corollary 4. We apply Lemma 19 with $\Delta = 1/2n^2$. If there is a component in the partition with at least $2n/3$ nodes, then we are done. Otherwise, we can find a cut with ratio cost strictly less than $36W \log n \leq \mathcal{S}$, which contradicts the minimality of $\mathcal{S}$.[8] □

[8] We can obtain a cut with ratio cost *strictly* less than $36W \log n$ since either (1) we can divide the components into two sets where each set has strictly more than $n/3$ nodes, or (2) there are three components each with exactly $n/3$ nodes in which case we can arrange the components into 2 sets (each with at least $n/3$ nodes) for which the capacity in the corresponding cut is strictly less than $4W \log n/\Delta$.

LEMMA 21. *For any graph G with min-cut $\mathcal{S}$, any distance function with total weight W, any length $L \geq (12C_{\max} \ln n/n\mathcal{S})$, and any subset of nodes $T \subseteq V$ for which $|T| \geq 2n/3$,*

$$\sum_{u \in V-T} d(T, u) \leq \frac{6W}{n\mathcal{S}},$$

where $d(T, u)$ is computed using only paths with at most $L/4$ edges.

PROOF. The proof is similar to that of Lemma 5 except that we need to be more careful about how G_i^+ is defined in order to ensure that we only use paths with at most $L/4$ edges when upper bounding the sum of distances.

In particular, we will now define G_i^+ iteratively, beginning (as before) with G_0^+ as the subgraph of G^+ induced on T. Given G_i^+, we produce G_{i+1}^+, as follows: Define S_i to be the set of edges that would be cut if G_i^+ were to be removed from G^+, C_i to be the total capacity of the edges in S_i, and C_i' to be the total capacity of the edges in S_i that link G_i^+ to a node of G not in G_i^+. If $C_i' \geq C_i/2$, then G_{i+1}^+ is the graph induced on G_i^+ and all nodes of G^+ within distance one of G_i^+ in G^+. If $C_i' < C_i/2$, then G_{i+1}^+ is defined to be the graph induced on G_i^+ and all nodes of $G^+ - G$ that are within distance one of G_i^+ in G^+. (In the latter case, we do not add nodes of G that are adjacent to G_i^+ when forming G_{i+1}^+. This will help ensure that the edge-radius of the corresponding components in G do not grow too quickly.)

We will begin by showing that there are not many levels i for which nodes of G were added to G_i^+ to form G_{i+1}^+. Let n_i denote the number of nodes of G not in G_i. By the minimality of $\mathcal{S}$, we know that $C_i \geq (2nn_i\mathcal{S}/3)$. If nodes of G were added to G_i^+ to form G_{i+1}^+, then

$$C_i' \geq \frac{C_i}{2} \geq \frac{nn_i\mathcal{S}}{3}.$$

Let $C_{\max}$ denote the maximum amount of total capacity on the edges incident to any single node. Then at least

$$\frac{C_i'}{C_{\max}} \geq \frac{nn_i\mathcal{S}}{3C_{\max}}$$

nodes of G are added to G_i^+ to form G_{i+1}^+. In other words,

$$n_{i+1} \leq \left(1 - \frac{n\mathcal{S}}{3C_{\max}}\right) n_i.$$

There are at most $(3C_{\max} \ln n/n\mathcal{S}) \leq L/4$ levels for which n_i can be decreased in this fashion without running out of nodes of G. Hence, there are at most $L/4$ levels i for which nodes of G are added to G_i^+ to form G_{i+1}^+.

We next upper-bound the sum of the n_i. By the construction of G_{i+1}^+, we know that at least $(C_i/2) \geq (nn_i\mathcal{S}/3)$ capacity is contained in edges that are entirely contained in G_{i+1}^+ but not G_i^+. Since the total capacity of G^+ is at most $C^+ \leq$

$2C$, this means that

$$\sum_{i \geq 0} \frac{n n_i \mathcal{S}}{3} \leq 2C$$

and thus that

$$\sum_{i \geq 0} n_i \leq \frac{6C}{n\mathcal{S}}.$$

We can now upper bound $\Sigma_{u \in V-T} d(T, u)$. Let $P^+(u)$ denote the path in G^+ from T to u formed by the process of growing the G_i^+. If u first appears in G_i^+, then this path has length $d^*_{G}{}^+(T, u) = i$ in G^+. Let $P(u)$ denote the corresponding path in G. By construction, $P(u)$ contains at most $L/4$ edges. Let $d^*_G(T, u)$ denote the distance of this path in G. Then

$$\begin{aligned}\sum_{u \in V-T} d(T, u) &\leq \sum_{u \in V-T} d^*_G(T, u) \\ &\leq \frac{W}{C} \sum_{u \in V-T} d^*_{G^+}(T, u) \\ &\leq \frac{W}{C} \sum_{i \geq 0} n_i \\ &\leq \frac{6W}{n\mathcal{S}},\end{aligned}$$

as claimed. □

Proof of Theorem 18. Set $L = (36 C_{\max} \log n / n\mathcal{S}) = O(C_{\max} \log n / n\mathcal{S})$. It remains to show that $W \geq \Omega(\mathcal{S}/\log n)$ if the distance constraint (restricted to paths of length at most L) is satisfied. The proof is by contradiction. Suppose that $W \leq \mathcal{S}/36 \log n$. Then, there is a feasible solution with $W = \mathcal{S}/36 \log n$ (by simply increasing the distances). Then, by Corollary 20, we can find a component T of G with edge-radius $(C/2Wn^2) \leq L/4$ and distance-radius $1/2n^2$ that contains $2n/3$ nodes.

By Lemma 21, we know that

$$\begin{aligned}\sum_{u \in V-T} d(T, u) &\leq \frac{6W}{n\mathcal{S}} \\ &= \frac{1}{6n \log n},\end{aligned}$$

where $d(T, u)$ is computed using paths with at most $L/4$ edges. Hence

$$\begin{aligned}
\sum_{u,v} d(u, v) &= \frac{1}{2} \sum_{(u,v)} d(u, v) \\
&\le \frac{1}{2} \sum_{(u,v)} \left(d(T, u) + d(T, v) + \frac{1}{n^2} \right) \\
&< n \sum_{u \in V-T} d(T, u) + \frac{1}{2} \\
&\le \frac{1}{6 \log n} + \frac{1}{2} \\
&< 1
\end{aligned}$$

where $d(u, v)$ is computed using paths of length at most L. This violates the distance constraint. Thus the flow with paths of length L is at least $\Omega(\mathcal{S}/\log n)$ as claimed. □

It is not difficult to show that the bound on L in Theorem 18 is tight up to a constant factor for many flow problems. For example, if G is a τ-regular expander (for any τ) with uniform capacities, then $L = \Theta(\log n)$, which is optimal up to a factor of $\Theta(\log \tau)$.

Theorem 18 can also be extended to product MFPs and directed UMFPs and PMFPs, but we will not present all the details here. The results can be worked out by using the methods of Sections 2.3 and 2.4 in combination with the proof of Theorem 18. In the case of PMFPs with $\binom{p}{2}$ commodities and total node weight p, the path length is defined as the number of nodes of nonzero weight in the path and a bound on the length is

$$O\left(\frac{\max\{C_{\max}, (2C/p)\} \log p}{p \mathcal{S}} \right),$$

where $C_{\max}$ is the maximum value over the nodes with nonzero weight of the capacity incident to the node divided by the weight of the node.

3. *Applications to Approximation Algorithms*

The max-flow min-cut theorems proved in Section 2 can be applied to develop good approximation algorithms for a surprisingly wide variety of NP-hard problems. We describe several of these algorithms in this section, beginning with approximation algorithms for most variants of the graph partitioning problem in Sections 3.1–3.7.

Graph partitioning is a critical component of many divide-and-conquer algorithms in practice. Unfortunately nearly every variation of the problem is NP-hard and little in the way of approximation algorithms has previously been discovered. In what follows, we describe $O(\log n)$-times optimal approximation algorithms for most variants of the problem. We then build upon these results to derive approximation algorithms for a wide variety of more complex problems such as minimum feedback arc set, crossing number, and PRAM emulation in a distributed network.

The performance guarantees of the algorithms described in this section are closely tied to the performance guarantee for the sparsest cut algorithm described in Section 3.1. If the performance guarantee of this algorithm is improved (as it has been for certain classes of graphs such as planar graphs or graphs with excluded minors [Park and Phillips 1993; Rao 1992; Klein et al. 1993] and dense graphs [Arora et al. 1995]), then it will be possible to derive corresponding improvements in the performance guarantees in the other algorithms described in the section.

All of the algorithms described in the section run in polynomial time. The fastest implementations can be derived by using algorithms developed by Klein et al. [1994], Leighton et al. [1992], Awerbuch and Leighton [1994], and Leong et al. [1991] for finding approximately optimal flows and/or distance functions. The fastest implementation of the sparsest cut algorithm runs in $\tilde{O}(n^2)$ steps and is due to Benczur and Karger [1996].[9] Finally, we mention that algorithms based on our approach were found experimentally to be useful for graph partitioning in a class of geometric graphs in Lang and Rao [1993], for certain VLSI benchmark circuits in Lang and Rao [1993] and Yeh et al. [1992], and certain benchmark sparse matrices [Klein et al. 1991].

3.1. Sparsest Cuts. The *sparsest cut* of a graph $G = (V, E)$ is a partition $\langle U, \bar{U}\rangle$ for which

$$\frac{|\langle U, \bar{U}\rangle|}{|U||\bar{U}|}$$

is minimized where $|\langle U, \bar{U}\rangle|$ denotes the number of edges connecting U to $V - U$. Computing the sparsest cut of a graph is NP-hard [Matula and Shahrokhi 1986]. The sparsest cut can be approximated to within an $O(\log n)$ factor using the algorithm of Section 2.2. In this case, we simply set all demands and capacities to be 1 and we find a cut with ratio cost $O(f \log n)$.

Edge-weighted, node-weighted, and directed versions of the sparsest cut problem can also be solved to within an $O(\log p)$ factor where p is the number of nodes with nonzero weight. In its most general form, we desire to find a partition $\langle U, \bar{U}\rangle$ for which

$$\frac{C(U, \bar{U})}{\pi(U)\pi(\bar{U})}$$

is minimized where $C(U, \bar{U})$ is the sum of the weights of the edges that lead from U to $\bar{U}$, and $\pi(U)$ is the sum of the weights of the nodes in U. The sparsest cut can be approximated to within a factor of $O(\log p)$ using the methods in Sections 2.2 and 2.3. In this case, we set the capacity of an edge to equal its

[9] The notation $\tilde{O}$ is similar to O except that it ignores logarithmic factors in addition to constant factors.

weight and $\pi(v)$ to be the node weight of v for all $v \in V$. We then find a cut with directed weighted ratio cost $O(f \log n)$.

3.2. FLUX, EXPANSION, AND MINIMUM QUOTIENT SEPARATORS. A closely related quantity to the sparsest cut of a graph is the *minimum edge expansion* or *flux* of a graph, defined by

$$\alpha = \min_{U \subseteq V} \frac{C(U, \bar{U})}{\min(|U|, |\bar{U}|)}.$$

In other words, a graph has flux at least α if every subset U with at most half of the nodes is connected to the rest of the graph with edges of total weight at least $\alpha|U|$. For example, unweighted (i.e., $C(e) = 1$ for all $e \in E$) expander graphs have flux at least $\Omega(1)$. A cut that achieves the flux is called the *minimum quotient separator* and is related by a constant factor to the sparsest cut since

$$\frac{n}{2}\mathcal{S} \le \alpha \le n\mathcal{S}$$

for any n-node graph. (This is because $n/2 \le \max(|U|, |\bar{U}|) \le n$).)

Computing the minimum quotient separator is NP-hard, even for unweighted graphs. (The proof is identical to that for graph bisection [Garey and Johnson 1979].)

The sparsest cut algorithms mentioned in Section 3.1 provide $O(\log n)$ approximation algorithms for the flux and minimum quotient separator problems, even in the case where edges and nodes are weighted and where edges are directed.

3.3. BALANCED CUTS AND SEPARATORS. In some applications, we desire to find a small cut in a graph $G = (V, E)$ that partitions the graph into nearly equal-size pieces. In general, we say that a cut $\langle U, \bar{U}\rangle$ is *b-balanced* or a *$(b, 1 - b)$-separator* (for $b \le 1/2$) if

$$b\pi(V) \le \pi(U) \le (1 - b)\pi(V)$$

where $\pi(U)$ denotes the sum of the node weights in U.

Finding b-balanced cuts of minimum edge weight is NP-hard, even in the special case when all node and edge weights are 1. In what follows, we describe a kind of approximation algorithm for this problem. In particular, if G has a b-balanced cut of size S, then we will show how to find a b'-balanced cut of size $O(S \log n/b - b')$ for any b' where $b' < b$ and $b' \le 1/3$.[10]

[10] In Leighton and Rao [1988], it was erroneously claimed that there is a polylogarithmic times optimal approximation algorithm for the case when $b' = b$. Such a result is not known. The best bound known for b-balanced cuts is $O(\sqrt{S|E|\log n})$. This result (which holds only for unweighted graphs) can be obtained by setting b' so that $b - b' = \sqrt{S \log n/|E|}$ and then finding a b'-balanced cut of size $O(S \log n/b - b') = O(\sqrt{S|E|\log n})$. This cut can be transformed into a b-balanced cut by taking $n^* \le (b - b')n$ nodes from the larger side of the cut and moving them to the smaller side. By selecting the n^* nodes carefully, this step adds at most $O((n^*/n)|E|) \le O((b - b')|E|) \le O(\sqrt{S|E|\log n})$ edges to the cut. To select the n^* nodes, we partition the nodes on the larger side into sets of size n^* and then select the set which results in the smallest cut.

The algorithm is very simple and does not need to know the value of S or b. We start by finding an approximate sparsest cut $\langle U, \bar{U}\rangle$ for $G_0 = G$ using the algorithm described in Section 3.1. We then remove the smaller (in node weight) of U or $\bar{U}$ from G, and call the residual graph G_1. Given a residual graph G_i, we find an approximate sparsest cut for G_i and remove the smaller set (call it U_i) to produce G_{i+1}. The algorithm continues until $\pi(G_{j+1}) \le (1 - b')\pi(G)$ for some j. The desired cut is then $\langle U_0 \cup U_1 \cup \cdots \cup U_j, \overline{U_0 \cup U_1 \cup \cdots \cup U_j}\rangle$.

The cut is easily seen to be b'-balanced since $b' \le 1/3$. In order to bound the weight of the edges in the cut, we first observe that for each $i \in [0, j]$, G_i always has a cut with ratio cost at most

$$\frac{S}{(b - b')\pi(G)(1 - b)\pi(G)} \le \frac{2S}{(b - b')\pi(G)^2}.$$

This is because G has a b-balanced cut of size S and we have removed at most $b'\pi(G)$ node weight from G to form G_i.

Let $\alpha_i\pi(G)$ denote the node weight removed from G_i to form G_{i+1}, and let S_i denote the edge weight in the corresponding cut. Since S_i is derived from an approximate sparsest cut, we know that

$$\frac{S_i}{\alpha_i\pi(G)(\pi(G_i) - \alpha_i\pi(G))} \le O\left(\frac{S \log n}{(b - b')\pi(G)^2}\right)$$

and thus that

$$S_i \le O\left(\frac{\alpha_i S \log n}{(b - b')}\right).$$

By construction, $\Sigma_{0 \le i \le j}\alpha_i \le 2/3$. Hence the total weight of the edges in the cut is at most

$$\sum_{0 \le i \le j} S_i \le O\left(\frac{S \log n}{b - b'}\right),$$

as claimed.

In a recent paper, Even et al. [1997] give an alternative approach to finding balanced separators using a generalized version of the dual to the UMFP. In addition, M. Goemans, B. Scheiber, M. Sudan, and D. Williamson (personal communications) have extended the balanced cut algorithm to hold when $b' < 1/2$. An adaptation of their extension is explained in what follows.

Set $\epsilon = (b - b')/2$ and let S' denote a threshold parameter that starts at 1 and doubles until a $(b', 1 - b')$-separator is found. Assume without loss of generality that $\pi(V) = n$. Recursively partition $G = (V, E)$ into pieces $G_1 = (V_1, E_1), \ldots, G_r = (V_r, E_r)$ in accordance with the following rules:

(1) if $\pi(V_i) \ge \epsilon n$ for any i, then find a $(\epsilon, 1 - \epsilon)$-separator for G_i using the algorithm described for the case when $b' \le 1/3$,

(2) if the cost of the $(\epsilon, 1 - \epsilon)$-separator is at most S', then partition G_i into two pieces and proceed recursively.

The result of this process is a partition of G into pieces of size at least $\epsilon^2 n$. Hence, there are at most $1/\epsilon^2$ pieces. The cost of the partition is thus at most S'/ϵ.[11]

We next consider all possible separators that can be formed from the $r \leq 1/\epsilon^2$ pieces. If there is a $(b', 1 - b')$-separator among the possibilities, then we terminate. Otherwise, we double the value of S' and continue cutting pieces of the graph. In what follows, we show that this algorithm terminates for some $S' = O(S \log n/\epsilon)$ where S is the size of any $(b, 1 - b)$-separator for which $b' < b \leq 1/2$. Hence, the algorithm always finds a $(b', 1 - b')$-separator of size $O(S \log n/(b - b')^3)$.

Suppose that G has a $(b, 1 - b)$-separator of size S. Divide each piece G_i of the partition of G into two subpieces G_i' and G_i'' according to the $(b, 1 - b)$-separator for G (and so that $bn \leq \Sigma_{i=1}^{r} \pi(V_i') \leq (n/2)$). Note that if $\pi(V_i) \geq \epsilon n$ there is a $S' = O(S \log n/\epsilon)$, where either $\pi(V_i')$ or $\pi(V_i'')$ is at most $2\epsilon\pi(V_i)$. (Otherwise, we would have cut G_i during the partitioning phase.)

We next consider the collection $\mathscr{C}$ of G_i for which $\pi(V_i) \geq \epsilon n$ *and* $\pi(V_i') \geq \pi(V_i)/2$. We then add as many G_i as possible for which $\pi(V_i) \leq \epsilon n$ so that the overall weight of the collection is at most $n/2 + 2\epsilon n$. If $\pi(V_i')$ or $\pi(V_i'')$ is at most $2\epsilon\pi(V_i)$ whenever $\pi(V_i) \geq \epsilon n$, then the total weight of the collection will be at least $bn - 2\epsilon n$ and at most $n/2 + 2\epsilon n$. Hence, we will have produced a $(b', 1 - b')$-separator for G for some $S' = O(S \log n/\epsilon)$, as claimed.

3.4. Directed Cuts. The balanced cut algorithm from Section 3.3 can also be applied to directed graphs, but we might lose a factor of 2 in the balance b'. This is because $U_0, U_1, \ldots, U_j$ must be divided into two classes depending on whether the sparse cut that was used to identify U_i consisted of edges directed towards U_i or away from U_i. One of the two classes must contain half the node weight of the total, however, and this class forms a directed $b'/2$-balanced cut of size $O(S \log n/b - b')$ where S is the size of a $(b, 1 - b)$-balanced directed cut.

The factor of 2 for directed cuts can be recovered by using an alternative definition of a directed cut (which we refer to as a 3-*way directed cut*). Consider a partition of the nodes into 3 sets V_1, V_2, and $\overline{V_1 \cup V_2}$. The edges in a 3-way directed cut $\langle V_1, V_2, \overline{V_1 \cup V_2}\rangle$ are those leaving V_1 and those entering V_2. The balance of the 3-way directed cut is defined to be

$$\frac{\min\{\pi(V_1) + \pi(V_2),\ \pi(\overline{V_1 \cup V_2})\}}{\pi(G)}.$$

In this context, V_1 (respectively, V_2) consists of those U_i for which edges are directed away from (respectively, towards) the set. The corresponding 3-way cut has balance b' and size $O(S \log n/(b - b'))$ where S is the size of a $(b, 1 - b)$-balanced 3-way directed cut.

[11] If ϵ is not too small, this can be accomplished by exhaustive search. If ϵ is small, then we can still run in polynomial time provided that $\pi(V)$ is polynomial in n. The algorithm is not polynomial-time if $\pi(V)$ is large and ϵ is small (since we are solving a problem that is at least as hard as the knapsack problem.)

3.5. Node Cuts. Thus far, we have focused our attention on edge cuts for graphs. These same methods can also be used to derive approximation algorithms for node cuts by converting the node-cut problem for an undirected graph into an edge-cut problem for a directed graph. (A *node cut* is a subset of nodes whose removal from the graph separates the remaining nodes into two disconnected pieces.) The transformation, which is well known, is explained in what follows.

Given a graph $G = (V, E)$ with node weights π for which we want to find a small node cut, we will produce a directed graph $G^* = (V^*, E^*)$ where

$$(V^* = \{v | v \in V\} \cup \{v' | v \in V\})$$

and

$$E^* = \{(v, v') | v \in V\} \cup \{(u', v) | (u, v) \in E\}$$
$$\cup \{(v', u) | (u, v) \in E\}.$$

Weights are assigned to each edge so that $C(v, v') = \pi(v)$ and $C(u', v) = \infty$ for $u \neq v$.

If we define the weight on one side of a node cut as the weight of the nodes on the side plus half the weight of the nodes in the cut, any node cut in G corresponds to a directed edge cut in G^* with the same cost and balance. (The correspondence is as follows: Let U denote the set of nodes that cuts $V - U$ into V_1 and V_2. Then, the cut of G^* is $\langle V_1^*, V_2^* \rangle$ where $V_1^* = \{v | v \in V_1 \cup U\} \cup \{v' | v \in U\}$.) And any directed edge cut in G^* with noninfinite cost corresponds to a node cut in G with the same cost and balance. Hence, the approximation algorithm for directed cuts in Section 3.4 can be extended to give an approximation algorithm for balanced node cuts.

The algorithm can be further extended to work with two different node weight functions: one (call it π_1) for the cost of removing a node, and one (call it π_2) for the contribution of the node to the balance. In this case, we set $C(v, v') = \pi_1(v)$ and $\pi(v) = \pi(v') = \pi_2(v)$. This construction is useful for the hypergraph partitioning algorithm described in Section 3.7.

3.6. Multi-Way Cuts. The sparsest cut algorithm of Section 3.1 can also be used iteratively to find near optimal partitions of a graph into components of size m. For example, Leighton et al. [1990] show that if the sparsest cut algorithm is used repeatedly until every component has size at most m, then the total weight of the removed edges is at most $O(\log n \log n/m)OPT_{m/3}$ where $OPT_{m/3}$ is the minimum edge weight that needs to be removed to split the graph into components of size at most $m/3$. In the special case when $m = O(\log n)$, the total weight that is removed by a closely related algorithm is at most $O(\log^2 n)OPT_m$ which is $O(\log^2 n)$ times optimal [Leighton et al. 1990].

Recently, Even et al. [1997] gave an $O(\log n \log \log n)$ approximation algorithm for this problem. The algorithm of Even et al. [1997] makes use of a generalization of the dual of the uniform multicommodity flow problems used here. Better bounds are possible for planar graphs [Leighton et al. 1990].

3.7. Hypergraph Partitioning. Leighton et al. [1990] also show how to extend the graph partitioning algorithms described above to hypergraphs. The key idea behind the work is to convert a hypergraph $G_h = (V_h, E_h)$ (with node weights π and edge weights C) into a bipartite graph $G = (V_1, V_2, E)$ with two node weight functions: one (call it π_1) for the cost of removing a node, and another (call it π_2) for the contribution that a node makes to the balance. We can then apply the node-cut algorithm described in Section 3.5. We note that this transformation was also used in the experimental studies of Lang and Rao [1993] and Yeh et al. [1992] as well as numerous other studies involving VLSI circuits.

The transformation proceeds as follows:

$$\begin{aligned} V_1 &= \{u \mid u \in E_h\}, \\ V_2 &= \{v \mid v \in V_h\}, \\ E &= \{(u, v) \mid v \in u \text{ in } G_h\}, \end{aligned}$$

$$\pi_1(u) = C(u) \quad \text{and} \quad \pi_2(u) = 0 \quad \text{for} \quad u \in E_h,$$

and

$$\pi_1(v) = \infty \quad \text{and} \quad \pi_2(v) = \pi(v) \quad \text{for} \quad v \in V_h.$$

Then there is a 1–1 correspondence between edge partitions in G_h and non-infinite-cost node partitions in G.

3.8. Minimum Cut Linear Arrangement. One of the most famous NP-hard ordering/cut problems is the minimum cut linear arrangement problem. Given a graph $G = (V, E)$, we desire to find an ordering of the nodes $v_1, v_2, \ldots, v_n$ which minimizes the value of

$$\mathscr{C} = \max_{1 \le i \le n} C(\{v_1, v_2, \ldots, v_i\}, \{v_{i+1}, v_{i+2}, \ldots, v_n\}).$$

In other words, we desire to find the minimum value of $\mathscr{C}$ (which is known as the *cutwidth* of G) for which there is an ordering of the nodes of G so that at most $\mathscr{C}$ edges connect any initial segment of the nodes to the remainder of the nodes. Like graph partitioning, min-cut linear arrangement arises in numerous applications.

In what follows, we will describe an $O(\log^2 n)$-times optimal approximation algorithm for min-cut linear arrangement. The algorithm is quite simple. Given a graph G, we begin by using the algorithm described in Section 3.3 to find a 1/3-balanced cut of size $O(S \log n)$ for G, where S is the size of an optimal 1/2-balanced cut (or bisection) of G. Let G_1 and G_2 denote the subgraphs of G formed by the cut. We then find a good ordering for G_1 and G_2 recursively, placing all the nodes of G_1 before the nodes of G_2 in the final ordering.

The algorithm for ordering the nodes of G forms a "decomposition tree" of subgraphs of G with depth at most $\log_{3/2} n = O(\log n)$. For any node v_j, let G_{i_1}, $G_{i_2}, \ldots, G_{i_{O(\log n)}}$ denote those subgraphs in the decomposition tree that contain v_j. Then the number of edges in the cut $\langle\{v_1, v_2, \ldots, v_j\}, \{v_{j+1}, v_{j+2}, \ldots, v_n\}\rangle$ for the ordering $v_1, v_2, \ldots, v_n$ produced by the algorithm is at most $O(S_{i_1} \log n) + O(S_{i_2} \log n) + \cdots + O(S_{i_{O(\log n)}} \log n)$, where S_{i_r} denotes the optimal bisection width of G_{i_r} for $1 \le r \le O(\log n)$.

Let $\mathscr{C}$ denote the optimal cutwidth for G. Note that $\mathscr{C}$ is lower bounded by the optimal bisection width of any subgraph of G. Thus, $S_{i_r} \leq \mathscr{C}$ for all r. This means that the cutwidth produced by the algorithm is at most $O(\mathscr{C} \log^2 n)$, as desired.

The preceding algorithm can be extended to directed graphs by ordering G_1 and G_2 so that the smaller set of directed edges in the cut always points from left to right.

3.9. Minimum Feedback Arc Set. The minimum feedback arc set problem consists of removing the smallest number of edges F from a digraph G so that the residual graph is acyclic. The problem is NP-hard and arises in numerous applications (such as circuit testing).

An equivalent formulation of the problem is to find an ordering $v_1, v_2, \ldots, v_n$ of the nodes of G so that the number of *forward* edges F (i.e., edges of the form (v_i, v_j) where $i < j$) is minimized. In what follows, we will describe an $O(\log^2 n)$ times optimal approximation algorithm for this problem.

Somewhat surprisingly, the approximation algorithm for minimum feedback arc set is identical to the algorithm just described for directed min-cut linear arrangement. In order to show that this algorithm produces an ordering with $O(F \log^2 n)$ forward edges where F is the optimal value for the graph, we again examine the decomposition tree of subgraphs produced by the algorithm. In particular, let $G_{i,j}$ denote the jth subgraph on level i of the decomposition tree, and let $S_{i,j}$ denote the optimal directed bisection for $G_{i,j}$. Then the number of forward edges for the ordering produced by the algorithm is

$$F_{alg} \leq \Theta(\log n) \sum_{i=1}^{\Theta(\log n)} \sum_{j=1}^{2^i} S_{i,j}.$$

We next observe that

(1) the minimum number of forward edges for any graph is at least as large as the minimum directed bisection for that graph, and
(2) if G' and G'' are disjoint subgraphs of G, then $F_G \geq F_{G'} + F_{G''}$.

Hence,

$$\begin{aligned} F_{alg} &\leq \Theta(\log n) \sum_{i=1}^{\Theta(\log n)} \sum_{j=1}^{2^i} F(G_{i,j}) \\ &\leq \Theta(\log n) \sum_{i=1}^{\Theta(\log n)} F \\ &= O(F \log^2 n), \end{aligned}$$

as desired.

An $O(\log n \log \log n)$ approximation algorithm was implicitly described by Seymour [1995] for this problem using similar ideas to those presented here. Even et al. [1998] made this algorithm explicit.

3.10. Minimum Register Sufficiency. Given a DAG G, the *register sufficiency problem* is to find a topological ordering of the vertices of the DAG v_1,

$v_2, \ldots, v_n$ for which

$$M = \max_{1 \le i \le n} M_i$$

is minimized where M_i is the number of nodes in $\{v_1, v_2, \ldots, v_i\}$ incident to a node in $\{v_{i+1}, \ldots, v_n\}$. The value of M is known as the *optimal register cost*, because it specifies the number of pebbles that are needed to pebble the computation graph represented by G.

The minimum register sufficiency problem is NP-hard and is very similar to the minimum cut linear arrangement problem. The main difference is that we have a DAG and are worried about node cuts instead of edge cuts. Ravi et al. [1991] describe an $O(\log^2 n)$ times optimal approximation algorithm for this problem. The algorithm combines techniques of Sections 3.5 and 3.8 along with some other ideas to obtain approximately optimal register cost.

3.11. UNIPROCESSOR SCHEDULING. Ravi et al. [1991] have also applied the method developed in this paper to find approximation algorithms for several uniprocessor scheduling problems. For example, let G be a DAG in which each node v corresponds to a task with execution time $l(v)$ and each edge $e = (u, v)$ has a weight $w(e)$ that represents the amount of storage required to save the intermediate results generated by task u until they are consumed by task v. The goal is to find a topological ordering of the nodes $v_1, v_2, \ldots, v_n$ for which

$$\mathscr{C} = \sum_{e} s(e) w(e)$$

is minimized where $s(e)$ for edge $e = (v_i, v_j)$ is the sum of the execution times of all tasks ordered between v_i and v_j, inclusive (i.e. $s(e) = \Sigma_{i \le k \le j} l(v_k)$).

In Ravi et al. [1991], an $O(\log n \log L)$ times optimal algorithm is presented for this problem where $L = \Sigma_{1 \le i \le n} l(v_i)$ is the sum of the execution times of the tasks.

Ravi et al. [1991] provide an $O(\log n \log L)$ times optimal approximation algorithm for the *minimum weighted completion time problem.* In this problem, the goal is to produce a topological ordering $v_1, v_2, \ldots, v_n$ for which

$$\sum_{1 \le i \le n} w(v_i) \sigma(v_i)$$

is minimized, where $w(v_i)$ is the weight of task v_i and $\sigma(v_i) = \Sigma_{1 \le j \le i} l(v_j)$ is the completion time of task v_i under the schedule.

These results were recently improved by Even et al. [1995] who give an $O(\log n \log \log n)$ approximation algorithm for this (and several other) problems. Their algorithm is based on a generalization of the dual to the multicommodity flow problem and the partitioning methods developed by Seymour [1995] mentioned earlier for approximating minimum feedback arc set.

3.12. CROSSING NUMBER. The *crossing number* $\mathscr{C}$ of a graph G is the minimum number of pairwise edge crossings that must appear in any drawing of G in the plane. For example, the crossing number of a planar graph is 0 and the crossing number of K_n is $\Theta(n^4)$ for $n \ge 5$. Determining the crossing number of a graph is NP-hard [Garey and Johnson 1979] and no good approximations are known for this problem in the general case.

Bhatt and Leighton [1984] showed how to obtain a planar drawing for any bounded-degree graph with $O((\mathscr{C} + n)B^2(n)\log^2 n)$ crossings provided that a $B(n)$-times optimal approximation algorithm for the graph bisection problem can be used as a subroutine. Like most of the algorithms described in Section 3, the algorithm uses the bisection algorithm to recursively partition the graph into two subgraphs, each of which is then drawn by recursion. Although no approximation algorithm for the graph bisection problem is yet known, it is sufficient for this application to find a (1/3, 2/3)-separator for the graph with size at most $O(B \log n)$ where B is the optimal bisection of the graph. Such an algorithm is described in Section 3.3. Then it is straightforward to modify the analysis of Bhatt and Leighton [1984] to show that the resulting drawing has $O((\mathscr{C} + n)\log^4 n)$ crossings.

At first glance, this approximation algorithm seems fairly weak since we are only approximating $\mathscr{C} + n$ and only for bounded-degree graphs. However, $\mathscr{C} > \omega(n)$ for any graph with bisection width $\omega(\sqrt{n})$ as well as any graph with $4n$ or more edges [Bhatt and Leighton 1984]. These conditions are satisfied by many graphs of interest, however. Moreover, this approximation algorithm is strong enough to be the basis of polylogarithmic times optimal solutions to other problems of interest. (See Sections 3.13 and 3.14.)

3.13. Bifurcators and Recursive Separators. As we have already seen, it is often useful to be able to repeatedly decompose a graph into smaller and smaller subgraphs with smaller and smaller edge-cuts. Any recursive decomposition into smaller and smaller subgraphs may be viewed as a decomposition tree. In particular, Bhatt and Leighton [1984] say that a graph G has an $(F_0, F_1, \ldots, F_r)$-*decomposition tree* when G can be decomposed into two subgraphs G_0 and G_1 by removing no more than F_0 edges, and, in turn, both G_0 and G_1 can be decomposed into smaller subgraphs by removing no more than F_1 edges from each and so on until each subgraph is empty or an isolated node. A particularly useful type of decomposition tree is called a *bifurcator.* An N-node graph has an α-*bifurcator* of size F (or an (F, α)-bifurcator) if it has an $(F, F/\alpha, \ldots, 1)$-decomposition tree. In particular, a $\sqrt{2}$-bifurcator is interesting since it was shown in Bhatt and Leighton [1984] that it can be used in the solution of a wide variety of problems in VLSI layout: minimizing capacitive delay, producing fault tolerant layouts, producing layouts for graphs using prefabricated chips, producing regular layouts, producing layouts without too many wire crossings, and a few other problems. (See Bhatt and Leighton [1984] for a detailed discussion of these problems.) Unfortunately, finding an optimal $\sqrt{2}$-bifurcator (i.e., a $\sqrt{2}$-bifurcator for which the value of F is minimized) involves the problem of graph partitioning, or graph bisection which is NP-hard. However, our methods can be combined with those of Bhatt and Leighton [1984] to produce an $O(\log^{2.5} n)$ times optimal $\sqrt{2}$-bifurcator for any graph.

The approximation algorithm for bifurcators uses the approximation algorithm for crossing numbers described in Section 3.12 as a preprocessing step. In particular, given an n-node bounded-degree graph G, we first draw G in the plane using $O((\mathscr{C} + n)\log^4 n)$ pairwise edge crossings, where $\mathscr{C}$ is the minimum crossing number of G. We can then use the planar separator theorem [Lipton and Tarjan 1979] as in Bhatt and Leighton [1984] to find a $\sqrt{2}$-bifurcator of size $O(\sqrt{(\mathcal{C} + n)\log^4 n}) = O(\sqrt{\mathcal{C} + n}\,\log^2 n)$. Since the $\sqrt{2}$-bifurcator of any graph must have size $\Omega(\sqrt{(\mathcal{C} + n)/\log n})$ Bhatt and Leighton [1984], this means that the $\sqrt{2}$-bifurcator produced by the algorithm is $O(\log^{2.5} n)$ times optimal, as claimed.

3.14. VLSI LAYOUT PROBLEMS. When designing a layout for a VLSI circuit, it is often useful to find a layout with minimum size. This problem can often be conveniently modeled by a graph embedding problem. In particular, let G denote the underlying graph for the circuit. In the special case, when G has maximum degree 4, then the layout problem corresponds to finding an embedding of G into a $m \times m$ grid H where the nodes of G are mapped injectively to the nodes of H and where the edges of G are mapped to edge-disjoint paths of H. The goal is to find an embedding for which the layout area $A = m^2$ is minimized. (The case when G has nodes with degree greater than 4 is similar, but slightly more complicated.)

Finding the minimum layout area is *NP*-hard, even when G is a forest of trees [Dolov et al. 1983]. By combining the methods of Bhatt and Leighton [1984] with the algorithm for crossing number just presented, we can find an $O(\log^6 n)$ times optimal approximation algorithm for this problem. In fact, the algorithm is quite simple. We first find a drawing of G in the plane with $O((\mathcal{C} + n)\log^4 n)$ crossings, where $\mathcal{C}$ is the crossing number of G. We then create a planar graph G^* with $n^* = O((\mathcal{C} + n)\log^4 n)$ nodes by replacing each edge crossing of G with a "dummy" node of degree 4. Using the layout algorithm of Leiserson [1980] and Valiant [1981], we can embed G^* in an $m \times m$ grid where $m = O(\sqrt{n^*}\log n^*)$. The embedding of G^* induces an embedding of G with layout area

$$\begin{aligned} A &= m^2 \\ &= O(n^* \log^2 n) \\ &= O((\mathcal{C} + n)\log^6 n). \end{aligned}$$

Since $\mathcal{C} + n$ is a lower bound on the layout area of G, the area achieved by the algorithm is within an $O(\log^6 n)$ factor of optimal.

In some circumstances, it is necessary to restrict the embedding so that the nodes of G are located on the perimeter of the grid H. In this case, we desire to find a rectangular $m \times O(n)$ grid containing G where m is minimized. The resulting embedding is known as a *colinear layout* [Leighton 1983]. The minimum value of m can be approximated to within an $O(\log^2 n)$ factor by using the min-cut linear arrangement algorithm described in Section 3.8.

Several other VLSI-related problems such as minimum wire volume and via minimization can also be approximately solved using the methods developed in this paper. We refer the reader to Bhatt and Leighton [1984] for further information on VLSI layout problems.

3.15. GEOMETRIC EMBEDDINGS. The *geometric embedding problem* consists of an edge-weighted graph $G = (V, E)$ and a set of points P in a d-dimensional Euclidean space. The goal is to find an injection f: $V \rightarrow P$ that minimizes the total edge length D induced on P, where

$$D = \sum_{(u,v)\in E} d(f(u), f(v))w(u, v),$$

$d(x, y)$ is the Euclidean distance between points x and y, and $w(u, v)$ is the weight of edge (u, v). The geometric embedding problem is similar to the VLSI layout problem when $d = 2$ except that we do not need to worry about wire width or separation and we can embed into arbitrary point sets.

In the case when the points of P are arranged as a d-dimensional array in 480^d, Hansen [1989] has applied the algorithms described in Sections 3.1–3.3 to obtain an $O(\log^2 n)$-times optimal approximation algorithm for D for any graph. A similar result is obtained with high probability in the case when the points of P are distributed uniformly in the unit sphere of $\Re^d$.

As with Uniprocessor Scheduling, Even et al. [1995] have recently discovered an $O(\log n \log \log n)$ approximation algorithm for geometric embedding problems into d-dimensional arrays.

3.16. EMBEDDINGS IN GENERAL GRAPHS. In Sections 3.14–3.15, we mentioned algorithms for embedding arbitrary graphs into grids. The methods developed in Section 2 can also be used to find a good embedding of an arbitrary graph G into an arbitrary graph H. By a good embedding, we mean an embedding that has small congestion and dilation. (An embedding maps nodes of G to nodes in H and edges in G to paths in H. The *congestion* of the embedding is the maximum for any edge e of H of the number of paths in the embedding that contain e. The *dilation* of an embedding is the maximum number of edges in any path in the embedding.)

In what follows, we will describe how to find a good embedding in the special case that G and H have bounded degree and unweighted edges (i.e., capacity 1 edges). (This result can be generalized to handle weighted graphs with unbounded node degree, but the quality of the embedding may degrade accordingly.) In particular, we will prove the following theorem. (The definition of flux is given in Section 3.2.)

THEOREM 22. *Consider any n-node bounded degree graph G and any* 1–1 *mapping h of the nodes of G onto the nodes of an n-node bounded degree graph H with flux* α. *The edges of G can be routed as paths in H with congestion and dilation* $O(\log n/\alpha)$.

PROOF. In order to produce the embedding, we would like to view each edge $e = (u_1, u_2)$ of G as incurring a demand of one unit of flow for commodity $\rho(e)$ between $h(u_1)$ and $h(u_2)$ in H. Unfortunately, the resulting flow problem is not uniform (nor is it a PMFP) and so we must modify the demands so that they appear uniform. We do this as follows: For each edge $e = (u_1, u_2)$ of G, we create $2n$ commodities $\rho_1(e), \ldots, \rho_{2n}(e)$ each with demand $1/n$, where commodity $\rho_i(e)$ is routed between $h(u_1)$ and v_i, and $\rho_{n+i}(e)$ is routed between v_i and $h(u_2)$ for $1 \le i \le n$, where v_i is the ith node of H.

When all the commodities $\{\rho_i(e) | 1 \le i \le 2n, e \in E(G)\}$ are taken together, we will have 1 unit of flow for each $(u, v) \in E(G)$ and we will have at most $2d_{\max}/n$ demand for each pair of nodes in H where $d_{\max}$ is the maximum node degree in G. Hence, we can view the flow problem as a UMFP on H with demands of size $2d_{\max}/n$. By Theorem 18, we can find a flow that satisfies the demands using flow paths of length $L \leq O(C_{\max}\log n/n\mathcal{S})$ provided that $f = c\mathcal{S}/\log n \geq 2d_{\max}/n$, where c is the constant hidden in the Ω-bound of Theorem 18.

If every edge of H is assigned capacity Γ, then $\mathcal{S} \geq \Gamma\alpha/n$. Hence, we can find the desired flow provided that $c\Gamma\alpha/n \log n \geq 2d_{\max}/n$, which is true provided that

$$\begin{aligned}\Gamma &\geq \frac{2d_{\max}\log n}{c\alpha} \\ &= \Theta\left(\frac{\log n}{\alpha}\right),\end{aligned}$$

which is the desired bound on congestion. Hence, we can solve the original flow problem where each edge $e = (u_1, u_2)$ of G incurs demand 1 between $h(u_1)$ and $h(u_2)$ in H using flow paths of length

$$\begin{aligned}2L &\leq O\left(\frac{C_{\max}\log n}{n\mathcal{S}}\right) \\ &= O\left(\frac{\Gamma \log n}{n(2\Gamma\alpha/n)}\right) \\ &= O\left(\frac{\log n}{\alpha}\right),\end{aligned}$$

provided that every edge of H is assigned capacity $\Gamma = O(\log n/\alpha)$.

It remains to find a single flow path with unit volume for every edge of G. This can be accomplished by using the rounding method of Raghavan [1988]. This method increases the capacity by a constant multiplicative factor and an additive $\log n$ term. Since $\alpha \leq O(1)$ for bounded-degree graphs, however, the resulting rounded flow still requires capacity only $O(\log n/\alpha)$. Hence, we can embed the edges of G into H with congestion and dilation $O(\log n/\alpha)$, as claimed. □

As a corollary, we can conclude that any bounded-degree graph can be embedded into any expander with $O(\log n)$ congestion and dilation.

3.17. PRAM EMULATION AND PACKET ROUTING IN DISTRIBUTED NETWORKS. Communication in a distributed network is an important area of research in which it is very difficult to prove precise results. For example, given an arbitrary n-processor network, we might wish to know how well it can simulate a well-studied network or other parallel machine such as the butterfly or PRAM. Some progress has been made on this problem in the special case that the graph is an expander. For example, Peleg and Upfal [1989] have shown how to solve any n-packet routing problem on any n-node expander in $O(\log n)$ expected steps using queues of size $O(\log n)$ at each node. For general graphs, relatively little is known except that it has been observed by many researchers that flux and diameter are important parameters that influence performance (e.g., they are both lower bounds on the time needed to route a random permutation).

In what follows, we provide the first step towards a general solution to the problem by showing how any n-node bounded-degree graph, H, with flux α can simulate any other n-node bounded-degree graph, G, with delay $O(\log n/\alpha)$ and constant size queues. The result is optimal in the sense that there are simulation problems that require this much time, and is robust in the sense that the simulation can take place for any 1–1 embedding of the nodes of G onto nodes of H. The main drawback of the simulation result is that it requires off-line computation. However, once the off-line embedding is performed for one G, simulation of another G' can be performed on-line by using G to simulate G'. For example, by embedding a butterfly or other universal network into H off-line, we can then use H to simulate any CRCW PRAM algorithm in an on-line fashion with delay $O(\log^2 n/\alpha)$ and constant size queues.

Conceivably, such a result could have a substantive impact on the theory of distributed computation, where the complexity of algorithms for operations such as sorting is typically measured in terms of numbers of nodes and edges and (sometimes) diameter. While such algorithms are of interest for some graphs, they can be far from optimal for many others. By using our flux-based approach, however, it is possible to devise a single on-line algorithm to sort in any network in $O(\log^2 n)$ steps times the optimal for that graph. Of course, we will still need off-line computation to set up routing tables for the graph, but if the sorting is to be done many times, the algorithm will be asymptotically much better than currently known techniques for general graphs.

The proof that any n-node bounded-degree graph H with flux α can simulate any other n-node bounded-degree graph G with delay $O(\log n/\alpha)$ follows from Theorem 22 and the result of Leighton et al. [1994] (or the constructive version of Leighton and Maggs [1995] and Leighton et al. [1995] that any packet routing problem can be solved in $O(\text{congestion} + \text{dilation})$ steps using constant size queues.

Unfortunately, the LMR routing algorithm is fairly complicated. We can circumvent this drawback by introducing randomness and queues, however. In particular, we can use Theorem 18 to embed K_n in G with congestion $O(n \log n/\alpha)$ and dilation $O((\log n)/\alpha)$. This can be performed in polynomial time off-line. We can then route a random permutation using queues of size $O((\log n)/\alpha)$ by introducing a random delay selected uniformly from $[1, (\log n)/\alpha]$ for each packet and multiplexing the packets as they arrive at each edge. It is shown in Leighton et al. [1994] that the congestion on any edge is only $O(\log n)$ with high probability in a random permutation; thus one only needs to multiplex each edge $O(\log n)$ times to get the congestion down to one. So each packet travels through at most $O(\log n/\alpha)$ edges, and it takes $O(\log n)$ steps to traverse any edge. Thus, a random permutation is routed in $O(\log^2 n/\alpha)$ expected steps. See Leighton et al. [1994] for a more extensive discussion of these details.

One final comment is relevant here. The observant reader will notice that we are routing arbitrary paths in G within time $O((\log n)/\alpha)$ without regard for the diameter of G. At first glance, this would not seem to be possible if the diameter were $\omega((\log n)/\alpha)$. Such a scenario is not possible, however, since a consequence of our proof is that the diameter of any bounded-degree graph is always $O((\log n)/\alpha)$.

3.18. EMBEDDING DISJOINT OR LOW CONGESTION PATHS IN GRAPHS. Given a set of k request pairs in a network, and a constant c, *the path embedding problem* is to route a path between each pair so that at most c paths use the same wire.

Leighton and Rao [1996] use Theorem 18 to show that *the path embedding problem* can be solved in polynomial time for any sufficiently strong expander graph when $k = O(n/(\log n)^{1+\epsilon})$ for a sufficiently large constant c. Also, Leighton and Rao [1996] existentially show that the *disjoint paths problem* (a path embedding problem with $c = 1$) has a solution for any $k = \Omega(n/(\log n \log\log n)^2)$ in a sufficiently strong expander.

This can be compared to a previous algorithm based on random walks developed by Broder et al. [1992], which obtains paths in polynomial time with congestion 1 for $k = O(n/\log^{\kappa} n)$ requests where κ seems to be at least 6. Broder et al. [1997] have extended their random walks approach to obtain existential bounds for $k = \Omega(n/(\log n)^2)$. Recently, Leighton et al. [1998] matched and generalized these results using the multicommodity flow approach.

3.19. FORWARDING INDEX PROBLEM. Given an n-node graph G and an embedding of K_n in G, Chung et al. [1987] defined the *forwarding index* of the embedding to be the maximum number of paths (each corresponding to an edge of K_n) that pass through any node of G. For applications involving network communications, the goal is to find an embedding that minimizes the forwarding index. As noted by Heydemann et al. [1989], it is also desirable to minimize the maximum number of paths passing through any edge (which they define as the *edge-forwarding index*).

Several bounds on forwarding indices have been proved based on properties of the underlying graph such as connectivity, edge distribution, factorization, or some special structure [Chung et al. 1987; Heydemann et al. 1989; 1992; 1994]. Although the bounds are nearly tight for some graphs, they can be far apart for others. By applying the methods described in Sections 3.16–3.17, however, it is possible to bound the node and edge-forwarding indices to within an $O(\log n)$-factor for every graph G in terms of the flux of the graph.

3.20. ROUTING IN OPTICAL NETWORKS. Given a graph $G = (V, E)$ and a set of *requests* R that consist of pairs of nodes in V, the *optical routing problem* is to partition the set of requests into a minimum number of sets or *rounds* $\{R_1, \ldots, R_k\}$ and to find for each i a set of edge-disjoint paths connecting the nodes of each request in R_i.

Aumann and Rabani [1995] apply the methods developed in this paper to devise an algorithm that can solve the problem using $O(\log^2 n/\alpha^2)$ rounds (where α is the flux of G) for any bounded-degree graph and any set of requests for which each node is the source or destination of at most one request. This result is within an $O(\log^2 n)$ factor of optimal since there is such a problem that requires at least $\Omega(1/\alpha^2)$ rounds for any α. This improves the results of Raghavan and Upfal [1994] who use a random walk based algorithm with a worst case bound of $O(\log^2 n/\alpha^4)$ rounds.

3.21. INTERVAL GRAPH COMPLETION. Given a graph G, the *interval graph completion problem* is to find an n-node interval graph G^* with the minimum number of edges for which G is a subgraph of G^*. Ravi et al. [1991] apply a balanced node separator algorithm like that in Section 3.5 to obtain an $O(\log^2 n)$ times optimal approximation algorithm for this problem.

These results were improved in Even et al. [1995] to yield an $O(\log n \log\log n)$ times optimal approximation algorithm.

3.22. Chordal Graph Completion. Given a graph G, the *chordal graph completion problem* is to find an n-node chordal graph G^* with the minimum number of edges for which G is a subgraph of G^*. Agrawal et al. [1993] use the results of Section 2 to obtain an $O(\sqrt{d_{\max}}\log^4 n)$ times optimal approximation algorithm for this problem where $d_{\max}$ is the maximum degree of a node in G.

3.23. Planar Edge Deletion. Tragoudas [1990] uses the approximation algorithm for balanced separators described in Section 3.3 to find a set of

$$R_{ALG} \le O\left((R \log n + \sqrt{nR})\log \frac{n}{R}\right)$$

edges whose removal from a bounded-degree graph G results in a planar graph, where R is the minimum number of edges that need to be removed from G before it becomes planar. Whether or not, there is a polylog n times optimal approximation algorithm for R remains an interesting open question.

3.24. Treewidth and Pathwidth. As defined in Bodlaender et al. [1995], a *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i | i \in I\}, T = (I, F))$, where T is a tree and $\{X_i\}$ is a collection of subsets of V, such that

—$\cup_{i\in I} X_i = V$.
—For all $(v, w) \in E$, there exists an $i \in I$ with $v, w \in X_i$.
—For all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *treewidth* of a tree decomposition $(\{X_i\}, T)$ is $\max|X_i| - 1$. The *treewidth of a graph* is the minimum treewidth of any tree decomposition. The *pathwidth* of a graph G is the minimum treewidth over all tree decompositions $(\{X_i\}, T)$ of G where T is a path.

Our algorithms are used to give an $O(\log n)$ approximation algorithm for treewidth and an $O(\log^2 n)$ approximation for pathwidth in Bodlaender et al. [1995].

3.25. Elimination Ordering Problems. Agrawal et al. [1993] use the results of Section 2 to derive a near optimal elimination ordering of the variables for solving a symmetric system of linear equations without pivoting. Their ordering results in $O(Md_{\max}\log^6 n)$ multiplications being performed during the elimination, where $d_{\max}$ is the maximum number of nonzero entries in any row of the matrix and M is the number of multiplications required for the optimal ordering.

The *elimination tree height* of G is the minimum parallel depth in terms of elimination steps for any elimination algorithm on G. The *minimum front size* of a graph G is the minimum over elimination orderings of G of the maximum degree of v in the filled (chordal) graph corresponding to nodes that are later in the elimination order. (See, e.g., Bodlaender et al. [1995] for the definition of an elimination algorithm.)

The algorithm of Section 2 is used to approximate front size in Bodlaender et al. [1995] and elimination tree height in Bodlaender et al. [1995] and Agarwal et al. [1993]. The approximation factors are $O(\log n)$ for minimum front size and $O(\log^2 n)$ for elimination tree height.

3.26. Search Number. As described in Ellis et al. [1994], the search number of a graph is the number of searchers needed to capture a fugitive that can move with arbitrary speed about the edges of the graph. A *search step* is the placing of a searcher on a vertex, the movement of a searcher along an edge, or the removal of a searcher from a vertex. A *search sequence* is a sequence of search steps. All edges are initially *contaminated.* An edge $e = (x, y)$ becomes *clear* when either there is a searcher on x and a second searcher moves from x to y or all the edges other than e incident to x are clear and a searcher moves from x to y. A clear edge e can become contaminated again if the movement or removal of a searcher results in a path from a contaminated edge to e that includes no nodes containing a searcher. A *search strategy* for a graph is a search sequence that results in all edges being simultaneously clear. The *search number* of a graph is the minimum number of searchers for which a search strategy exists.

The *node search number* of a graph, defined in Kirousis and Papadimitriou [1986], is similar except that a searcher does not need to move along an edge; looking along it is sufficient to catch a fugitive.

Kirousis and Papadimitriou [1986] show that the node search number of a graph is equal to its pathwidth plus one. Ellis et al. [1994] showed that the search number of a graph is equal to its pathwidth plus two.

Thus, using the result described in Section 3.24, our results give an $O(\log^2 n)$ approximation algorithm for finding the search number and node search number of a graph.

3.27. Resistance, Conductance, and Rapidly-Mixing Markov Chains. Reversible Markov Chains are often identified with a weighted directed graph $G = (V, E)$ where the weight $\pi(v)$ of a node v is the probability of being in state v in the stationary distribution and the weight $w(e)$ of an edge $e = (u, v)$ is $\pi(u)P(u, v)$ where $P(u, v)$ is the probability of moving to state v from state u. The *conductance* C of the chain is equal to the flux of G and is useful in quantifying the most severe transition bottleneck in the chain. The *resistance* R of the chain is the minimum capacity needed on each edge in order that there exist a solution to the PMFP where $\pi(u)\pi(v)$ flow is passed between u and v for all u, $v \in V$.

As a consequence of Theorem 7, we can conclude that for any chain, C^{-1} and R are equal up to a $\Theta(\log p)$ factor, where p is the number of states in the chain with nonzero stationary probability. (Sinclair [1993] uses similar methods to prove a slightly weaker bound.)

A Markov chain is said to be rapidly-mixing if the chain reaches equilibrium quickly. The mixing rate of a chain is often characterized in terms of its second eigenvalue [Diaconis and Stroock 1991], which is known to be within a square of the conductance [Sinclair and Jerrum 1989]. Diaconis and Strook [1991] have also devised bounds on mixing rate derived directly from the resistance. Sinclair [1991] applies Theorem 7 to connect these methods and to derive improved approximation algorithms for a variety of difficult combinatorial problems.

The connection between max-flow, min-cut theorems and Markov chains is important and worthy of further exploration. (For further information on rapidly-mixing Markov chains, we refer the reader to Diaconis and Stroock [1991], Lovász [1991], Sinclair [1991; 1993], and Sinclair and Jerrum [1989].) The connection is particularly important in the domain of approximation algorithms, where it is known that the methods perform within a quadratic factor of each other. For some problems (such as path routing), however, the quadratic factor can be substantial and so it is necessary to try each method independently to obtain the best approximation algorithm. (For example, see Section 3.20.)

4. *Remarks and Open Questions*

Since the initial results of this work first appeared in 1988, dramatic progress has been made on a wide variety of problems involving multicommodity flow. Two central questions still remain unresolved, however:

(1) Is there a max-flow min-cut theorem similar to Theorem 2 for directed multicommodity flow problems with general demands?

Currently, such a result is known only if the demands are symmetric [Klein et al. 1997; Even et al. 1998], and even in this case, the bounds are not tight. Negative results are only known for restricted notions of cuts. (In general, any set of edges might be considered to be a cut, even if the set of edges does not correspond to a partition of the graph.)

(2) Is there a polynomial-time approximation algorithm for balanced separators that achieves a $o(\log n)$-times optimal performance guarantee?

No such result is currently known for general graphs, and no lower bounds are known for this problem. Arora et al. [1995] give a fully polynomial time randomized approximation scheme for the special case of dense graphs. Any improvement in the $O(\log n)$-times optimal performance guarantee provided in this paper would immediately result in improved performance guarantees for all of the approximation algorithms in Section 3.

ACKNOWLEDGMENTS The proof of Theorem 7 was suggested by Eva Tardos. The fact that Theorem 2 could be extended in this way was also pointed out by David Shmoys and Alistair Sinclair.

We would also like to thank Sanjeev Arora, Yonatan Aumann, Baruch Awerbuch, Sandeep Bhatt, Hans Bodlaender, Andrei Broder, Fan Chung, Persi Diaconis, Alan Frieze, John Gilbert, Mark Hansen, David Johnson, Nabil Kahale, David Karger, Phil Klein, Danny Lewin, Nati Linial, László Lovász, Bruce Maggs, Fillia Makedon, David Matula, David Peleg, Serge Plotkin, Yuval Rabani, Arnie Rosenberg, Paul Seymour, Farhad Shahrokhi, Dominique Sotteau, Cliff Stein, Madhu Sudan, Éva Tardos, Spyros Tragoudas, Umesh Vazirani, and Mihalis Yannakakis for helpful comments and criticisms.

Finally, we are greatly indebted to the referees for their very thorough reading of the paper, and for the hundreds of suggestions they made for its improvement.

REFERENCES

AGRAWAL, A., KLEIN, P. N., AND RAVI, R. 1993. Cutting down on fill using nested dissection: Provably good elimination orderings. In *Graph Theory and Sparse Matrix Computation*, A. George,

J. Gilbert, and J. W. H. Liu, eds. IMA Volumes in Mathematics and Its Applications, Springer-Verlag, New York, pp. 31–55.

Arora, S., Karger, D., and Karpinski, M. 1995. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing* (Las Vegas, Nev., May 29–June 1). ACM, New York, pp. 284–293.

Aumann, Y. and Rabani, Y. 1995. Improved bounds for all-optical routing. In *Proceedings of the 6th Annual ACM–SIAM Symposium on Discrete Algorithms* (San Francisco, Calif., Jan. 22–24), ACM, New York, pp. 567–576.

Aumann, Y. and Rabani, Y. 1998. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput. 27*, 1, 291–301.

Awerbuch, B., Berger, B., Cowen, L., and Peleg, D. 1999. Near-linear cost constructions of neighborhood covers in sequential and distributed environments. *SIAM J. Comput. 28*, 1, 263–277.

Awerbuch, B. and Leighton, T. 1994. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing* (Montreal, Que., Canada, May 23–25). ACM, New York, pp. 487–495.

Awerbuch, B. and Peleg, D. 1990. Sparse partitions. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science.* IEEE Computer Society Press, Los Alamitos, Calif., pp. 503–513.

Benczur, A. and Karger, D. 1996. Approximate s-t min-cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing* (Philadelphia, Pa., May 22–24). ACM, New York, pp. 47–55.

Bhatt, S. N. and Leighton, F. T. 1984. A framework for solving VLSI graph layout problems. *J. Comput. Syst. Sci. 28*, 2, 300–343.

Bodlaender, H., Hafsteinsson, H., Gilbert, J. R., and Kloks, T. 1995. Approximating tree-width, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms 18*, 238–255.

Bourgain, J. 1985. On Lipschitz embedding of finite metric spaces in Hilbert space. *Is. J. Math. 52*, 46–52.

Broder, A. Z., Frieze, A. M., and Upfal, E. 1992. Existence and construction of edge disjoint paths on expander graphs. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing* (Victoria, B.C., Canada, May 4–6). ACM, New York, pp. 140–149.

Broder, A. Z., Frieze, A. M., and Upfal, E. 1997. Static and dynamic path selection on expander graphs: A random walk approach. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing* (El Paso, Tex., May 4–6), ACM, New York, pp. 531–539.

Chung, F. K., Coffman, E. G., Reiman, M. I., and Simon, B. E. 1987. The forwarding index of communication networks. *IEEE Trans. Inf. Theory 33*, 224–232.

Chvatal, V. 1983. *Linear Programming.* Freeman, San Francisco, Calif.

Cohen, E. 1993. Fast algorithms for constructing t-spanners and paths with stretch t. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science* (Nov.), IEEE Computer Society Press, Los Alamitos, Calif., pp. 648–658.

Diaconis, P. and Stroock, D. 1991. Geometric bounds for eigenvalues of Markov chains. *Ann. Appl. Prob. 1*, 36–61.

Dolev, D., Leighton, F. T., and Trickey, H. 1983. Planar embeddings of planar graphs. Tech. rep. MIT, Cambridge, Mass.

Ellis, J. A., Sudborough, I. H., and Turner, J. S. 1994. The vertex separation and search number of a graph. *Inf. Comput. 113*, 50–79.

Even, G., Naor, J., Rao, S., and Schieber, B. 1995. Divide-and-conquer approximation algorithms via spreading metrics. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science* (Oct.), IEEE Computer Society Press, Los Alamitos, Calif., pp. 62–71.

Even, G., Naor, J., Rao, S., and Schieber, B. 1997. Fast approximate graph partitioning algorithms. In *Proceedings of the 8th Annual ACM–SIAM Symposium on Discrete Algorithms.* ACM, New York, pp. 639–648.

Even, G., Naor, J., Schieber, B., and Sudan, M. 1998. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica 20*, 2, 151–174.

Ford, L. R. and Fulkerson, D. R. 1956. Sur le probléme des courbes gauches en topologie. *Canad. J. Math 8*, 399–404.

FRANK, A. 1990. Packing paths, circuits and cuts—A survey. In *Paths, Flows, and VLSI-Layout*, B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, eds. Springer-Verlag, Berlin, Germany, pp. 47–100.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, Calif.

GARG, N., VAZARANI, V., AND YANNAKAKIS, M. 1996. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM J. Comput. 25*, 235–251.

HANSEN, M. 1989. Approximation algorithms for geometric embeddings in the plane and applications to parallel processing problems. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science* (Oct.). IEEE Computer Science Society Press, Los Alamitos, Calif., pp. 604–609.

HEYDEMANN, M. C., MEYER, J. C., AND SOTTEAU, D. 1989. On forwarding indices of networks. *Discrete Applied Mathematics 23*, 103–123.

HEYDEMANN, M. C., MEYER, J. C., AND SOTTEAU, D. 1992. Forwarding indices of k-connected graphs. *Disc. Appl. Math. 37/38*, 287–296.

HEYDEMANN, M. C., MEYER, J. C., AND SOTTEAU, D. 1994. Forwarding indices of consistent routings and their complexity. *Networks 24*, 75–82.

HU, T. C. 1963. Multicommodity network flows. *Oper. Res. 11*, 3, 344–360.

IRI, M. 1967. On an extension of the maximum-flow minimum cut theorem to multicommodity flows. *J. Oper. Res. Soc. Japan 5*, 4 (Dec.), 697–703.

KAMATH, A. AND PALMON, O. 1995. Improved interior point algorithms for exact and approximate solutions of multicommodity flow problems. In *Proceedings of the 6th Annual ACM–SIAM Symposium on Discrete Algorithms* (San Francisco, Calif., Jan. 22–24). ACM, New York, pp. 502–511.

KIROUSIS, L. M. AND PAPADIMITRIOU, C. H. 1986. Searching and pebbling. *Theoret. Comput. Sci. 47*, 205–216.

KLEIN, P., AGRAWAL, A., RAO, S., AND RAVI, R. 1989. Approximation through multicommodity flow. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science* (Oct.). IEEE Computer Society Press, Los Alamitos, Calif., pp. 726–737.

KLEIN, P., PLOTKIN, S., RAO, S., AND TARDOS, E. 1997. Bounds on the max-flow min-cut ratio for directed multicommodity flows. *J. Algorithms 22*, 241–269.

KLEIN, P., PLOTKIN, S., STEIN, C., AND TARDOS, E. 1994. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM J. Comput. 23*, 466–487.

KLEIN, P., RAO, S., AGRAWAL, A., AND RAVI, R. 1995. Approximation through multicommodity flow. *Combinatorica 15*, 187–202.

KLEIN, P., PLOTKIN, S., AND RAO, S. 1993. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (San Diego, Calif., May 16–18). ACM, New York, pp. 682–690.

KLEIN, P. N., BORGER, J. M., AND KANG, S. 1991. Approximating concurrent flow with uniform demands and capacities: An implementation. In *Proceedings of DIMACS Implementation Challenge Workshop: Network Flows and Matching* (Oct.). AMS, Providence, R.I., pp. 371–386.

LANG, K. AND RAO, S. 1993. Finding near-optimal cuts: An empirical evaluation. In *Proceedings of the 4th Annual ACM–SIAM Symposium on Discrete Algorithms.* ACM, New York, pp. 212–221.

LEIGHTON, F. T. 1983. *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks.* MIT Press, Cambridge, Mass.

LEIGHTON, F. T., MAGGS, B., AND RAO, S. 1994. Packet routing and job-shop scheduling in o(congestion + dilation) steps. *Combinatorica 14*, 2, 167–180.

LEIGHTON, F. T. AND RAO, S. 1996. Circuit switching: A multicommodity flow based approach. In *Proceedings of the 1st Workshop on Randomized Parallel Computing* (Apr.).

LEIGHTON, F. T., RAO, S. B., AND SRINIVASAN, A. 1998. Multi-commodity flow and circuit switching. In *Proceedings of the 31st Hawaii International Conference on System Sciences*, vol. 7. IEEE Computer Society Press, Los Alamitos, Calif., pp. 466–474.

LEIGHTON, F. T. AND RAO, S. 1988. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science.* IEEE Computer Society Press, Los Alamitos, Calif., pp. 256–269.

LEIGHTON, F. T., MAKEDON, F., PLOTKIN, S., STEIN, C., TARDOS, E., AND TRAGOUDAS, S. 1992. Fast approximation algorithms for multicommodity flow problems. *J. Comput. Syst. Sci. 50*, 228–243.

LEIGHTON, F. T., MAKEDON, F., AND TRAGOUDAS, S. 1990. Approximation algorithms for VLSI partition problems. In *Proceedings of the IEEE International Symposium on Circuits and Systems.* IEEE Computer Society Press, Los Alamitos, Calif.

LEIGHTON, T. AND MAGGS, B. 1995. Fast algorithms for finding O(congestion + dilation) packet routing schedules. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS)*, vol. 2. IEEE Computer Society Press, Los Alamitos, Calif., pp. 555–563.

LEIGHTON, T., MAGGS, B., AND RICHA, A. 1995. Fast algorithms for finding O(congestion + dilation) packet routing schedules. Tech. Rep. School of Computer Science, Carnegie Mellon University, Pittsburgh, Pa.

LEISERSON, C. E. 1980. Area-efficient layouts (for VLSI). In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science* (Oct.). IEEE Computer Science Press, Los Alimatos, Calif., pp. 270–281.

LEONG, T., SHOR, P., AND STEIN, C. 1991. Implementation of a combinatorial multicommodity flow algorithm. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: Volume 12, Network Flows and Matching*, D. S. Johnson and C. C. McGeoch, eds. (Oct.). AMS, Providence R.I., pp. 387–406.

LINIAL, N., LONDON, E., AND RABINOVICH, Y. 1995. The geometry of graphs and some of its algorithmic applications. *Combinatorica 15*, 215–245.

LIPTON, J. AND TARJAN, R. E. 1979. A separator theorem for planar graphs. *SIAM J. Appl. Math. 36*, 2 (Apr.), 177–189.

LOMONOSOV, M. V. 1985. Combinatorial approaches to multiflow problems. *Disc. Appl. Math. 11*, 1–94.

LOVÁSZ, L. 1991. Random walk on graphs: A survey. Tech. Rep. Dept. Computer Science, Yale Univ., New Haven, Conn.

MARGULIS, G. A. 1973. Explicit constructions of concentrators. *Prob. Inf. Trans. 9*, 325–332.

MATULA, D. W. AND SHAHROKHI, F. 1986. The maximum concurrent flow problem and sparsest cuts. Tech. Rep. Southern Methodist Univ., Dallas, Tex.

OKAMURA, H. AND SEYMOUR, P. D. 1981. Multicommodity flows in planar graphs. *J. Combin. Theory, Ser. B 31*, 75–81.

PAPERNOV, B. A. 1990. Feasibility of multicommodity flows (in Russian). In *Studies in Discrete Optimization*, A. A. Friedman, ed. New York, pp. 17–34.

PARK, J. AND PHILLIPS, C. 1993. Finding minimum quotient cuts in planar graphs. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (San Diego, Calif., May 16–18). ACM, New York, pp. 766–775.

PELEG, D. AND UPFAL, E. 1989. The token distribution problem. *SIAM J. Comput. 18*, 2 (Apr.), 229–243.

PLOTKIN, S. AND TARDOS, E. 1993. Improved bounds on the max-flow min-cut ratio for multicommodity flows. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (San Diego, Calif., May 16–18). ACM, New York, pp. 691–697.

RAGHAVAN, P. 1988. Probabilistic construction of deterministic algorithms: Approximate packing integer programs. *J. Comput. Syst. Sci. 37*, 4 (Oct.), 130–143.

RAGHAVAN, P. AND UPFAL, E. 1994. Efficient routing all-optical networks. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing* (Montreal, Que., Canada, May 23–25). ACM, New York, pp. 134–143.

RAO, S. 1992. Faster algorithms for finding small edge cuts in planar graphs. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing* (Victoria, B.C., Canada, May 4–6). ACM, New York, pp. 229–240.

RAVI, R., AGRAWAL, A., AND KLEIN, P. N. 1991. Ordering problems approximated: Single-processor scheduling and interval graph completion. In *Proceedings, 18th International Conference on Automata, Languages, and Programming.* Lecture Notes in Computer Science, vol. 510. Springer-Verlag, New York, pp. 751–762.

SCHRIJVER, A. 1990. Homotopic routing methods. In *Paths, Flows, and VLSI-Layout*, B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, eds. Springer-Verlag, Berlin, Germany, pp. 329–371.

SEYMOUR, P. D. 1995. Packing directed circuits fractionally. *Combinatorica 15*, 281–288.

SHAHROKHI, F. AND MATULA, D. W. 1990. The maximum concurrent flow problem. *J. ACM 37*, 318–334.

SHMOYS, D. B. 1996. Cut problems and their application to divide-and-conquer. In *Approximation Algorithms*, D. S. Hochbaum, ed. PWS Publishers, Boston, Mass., pp. 192–235.

SINCLAIR, A. 1991. Improved bounds for mixing rates of Markov chains and multicommodity flow. Tech. Rep. Laboratory for Foundations of Computer Science, Department of Computer Science, The University of Edinburgh, Edinburgh, Scotland.

SINCLAIR, A. 1993. *Algorithms for Random Generating and Counting.* Birkhäuser, Boston, Mass.

SINCLAIR, A. J. AND JERRUM, M. R. 1989. Approximate counting, uniform generation and rapidly mixing Markov chains. *Inf. Comput. 82*, 93–133.

TARJAN, R. E. 1983. *Data Structures and Network Algorithms.* SIAM, Philadelphia, Pa.

TRAGOUDAS, S. 1990. VLSI partitioning approximation algorithms based on multicommodity flows and other techniques. Ph.D. dissertation. Dept. Comput. Sci., Univ. Texas, Dallas, Dallas, Tex.

VAIDYA, P. M. 1989. Speeding up linear programming using fast matrix multiplication. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science.* IEEE Computer Society Press, Los Alamitos, pp. 332–337.

VALIANT, L. 1981. Universality considerations in VLSI circuits. *IEEE Trans. Comput. C-30*, 2, 135–140.

YEH, C. W., CHENG, C. K., AND LIN, T. T. 1992. A probabilistic multicommodity-flow solution to circuit clustering problems. In *Proceedings of the IEEE International Conference on Computer-Aided Design.* IEEE Computer Society Press, Los Alamitos, Calif. pp. 428–431.

RECEIVED NOVEMBER 1996; REVISED AUGUST 1998; ACCEPTED SEPTEMBER 1998