An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms

Tom Leighton
Department of Mathematics and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Satish Rao
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

In this paper, we consider a multicommodity flow problem where for each pair of vertices, (u, v), we are required to send f half-units of commodity (u, v) from u to v and f half-units of commodity (v, u) from v to u without violating capacity constraints. Our main result is an algorithm for performing the task provided that the capacity of each cut exceeds the demand across the cut by a $\Theta(\log n)$ factor. The condition on cuts is required in the worst case, and is trivially within a $\Theta(\log n)$ factor of optimal for any flow problem.

The result is of interest because it can be used to construct the first poly-log times optimal approximation algorithms for a wide variety of problems, including minimum quotient separators, $\frac{1}{3}-\frac{2}{3}$ separators, bifurcators, crossing number and VLSI layout area. The result can also be used to efficiently route packets in arbitrary distributed networks. For example, we can prove that any n-node bounded degree graph, G, with minimum edge expansion α can be configured off-line to simulate any n-node bounded degree graph H in $O(\frac{\log n}{\alpha})$ steps using constant size queues. By letting H be a universal network, we can then use G to simulate a PRAM on-line with delay $O(\frac{\log^2 n}{\alpha})$ steps, resolving a conjecture in the thoery of distributed computation. These results compare well with the universal lower bound of $\Omega(\frac{1}{\alpha})$ and the existential lower bound of $\Omega(\frac{\log n}{\alpha})$.

1 Introduction

1.1 Multicommodity Flow Problems

A *multicommodity flow problem* (MFP) consists of a graph $G = (V, E)$, a set of commodities, Γ, a capacity function on the edges $C : E \rightarrow R^+$, a supply function for vertices $S : V \times \Gamma \rightarrow R^+$, and a demand function for vertices $D : V \times \Gamma \rightarrow R^+$. The object is to construct flows for the commodities that satisfy the demand for each commodity at each vertex without violating the constraints imposed by the supply function and the capacity function. It is well known that any MFP can be solved in polynomial time using linear programming.

In this paper, we consider a special kind of MFP that we call a *uniform multicommodity flow problem* (UMFP). In a UMFP, we have a separate commodity for each ordered pair of vertices (i.e., $\Gamma = V \times V$), and we insist that f half-units of commodity (u, v) flow from u to v and that f half-units of commodity (v, u) flow

This research was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-80-C-0622 the Office of Naval Research under Contract N00014-86-K-0593, the Air Force under Contract OSR-86-0076, and the Army under Contract DAAL-03-86-K-0171. Tom Leighton is also supported in part by an NSF Presidential Young Investigator Award with matching funds from IBM and AT&T.

0272-5428/88/0000/0422$01.00 © 1988 IEEE

from v to u for each $u, v \in V$. The underlying graph and capacities can be arbitrary, although we will only need the special case when the edge capacities are all equal for the application. The object of a UMFP is to maximize f subject to the constraints imposed by the edge capacities. This can be done in polynomial time using linear programming, and can be approximately done using faster algorithms by Shahrokhi and Matula [19].

As in the case for the single commodity flows, an MFP cannot be solved if there is a cut of the graph for which the demand across the cut exceeds the capacity of the cut. In the case of a UMFP, the demand across a cut $(U, \overline{U})$ is simply $f|U|\left|\overline{U}\right|$ where $U \subseteq V$. The capacity of the cut is simply the sum of the capacities of the edges crossing the cut. In the case when all the capacities are equal, the capacity of the cut is proportional to the number of edges in the cut. This motivates the definition of the *minimum cut* for a UMFP to be

$$\mathcal{S} = \min_{U \subseteq V} \frac{\sum^{e \in (U,\overline{U})} C(e)}{|U|\left|\overline{U}\right|} \tag{1}$$

Note that for any feasible solution to a UMFP, we must have $f \leq \mathcal{S}$.

The value of $\mathcal{S}$ defined in equation 1 also has interpretations in graph theory. If $C(e) = 1$ for all $e \in E$, then $\mathcal{S}$ is the value of the *sparsest cut* of the graph. In the case of arbitrary edge capacities, then $\mathcal{S}$ is the weighted sparsest cut value for the graph with edge weights given by C. Even in the simple case where all edge weights (capacities) are 1, computing the sparsest cut of a graph is NP-hard [14].

A closely related quantity to the sparsest cut of a graph is the *minimum edge expansion* or *flux* of a graph, defined by

$$\alpha = \min_{U \subseteq V} \frac{\sum^{e \in (U,\overline{U})} C(e)}{\min(|U|, \left|\overline{U}\right|)} \tag{2}$$

In other words, a graph has flux α if every subset U with at most half of the vertices 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 constant flux. A cut that achieves the flux is called the *minimum quotient separator* and is related by a constant factor to the sparsest cut since

$$\frac{n}{2}\mathcal{S} \leq \alpha \leq n\mathcal{S} \tag{3}$$

for any n-vertex graph since the number of vertices in the big half of a cut must lie between $\frac{n}{2}$ and n. Computing the minimum quotient separator is also NP-complete. (The proof is identical to that of graph bisection [7].)

Unlike the case for single commodity flow problems, there is no max-flow, min-cut theorem for the UMFP. In particular, the condition that $\mathcal{S} \geq f$ is necessary but not always sufficient for a UMFP to have a flow of size f. As a particularly nasty example, consider a UMFP on a bounded-degree expander graph with identical edge capacities $C(e) = C_0$. Since most pairs of vertices are separated by distance $\Omega(\log n)$ in such a graph, the sum of the edge capacities must be $\Omega(fn^2 \log n)$ in order for there to be a flow of size f in the UMFP. This means that $C_0 = \Omega(fn \log n)$ which is equivalent to requiring that $\mathcal{S} = \Omega(f \log n)$. Hence, in this case, the capacity across each cut must exceed the demand across the cut by a $\Theta(\log n)$ factor in order for there to be a solution to the UMFP.

The preceding example is as bad as things can get, however. This fact is summarized in the following theorem.

Theorem 1 *There is a feasible flow with* $f = \Omega(\frac{\mathcal{S}}{\log n})$ *for any* n*-vertex UMFP.*

The preceding result provides an approximate max-flow min-cut relationship for UMFPs that is tight in the worst case. In particular, we have that

$$\Omega(\frac{\mathcal{S}}{\log n}) \leq f \leq \mathcal{S}, \tag{4}$$

for any UMFP. In other words, the maximum flow is always within a $\Theta(\log n)$ factor of the minimum cut. Moreover, there are examples of both $f = \Theta(\mathcal{S})$ and $f = \Theta(\mathcal{S}/\log n)$, so the approximation is existentially tight.

The proof of Theorem 1 is the first of two main results in the paper and is included in Section 2. In the remainder of this section, we describe the method of the proof, the second main result, and the applications of the results to a wide variety of problems. Section 3 contains the proof of Theorem 2. We conclude with some remarks, acknowledgements, and references in Sections 4 - 6.

1.2 Proof Method

To prove Theorem 1, we work with the dual of a UMFP. The dual UMFP problem consists of a weighted graph where the weight of an edge (denoted by $d(e)$) is treated as a distance and is selected by the adversary subject to the condition that $\sum^{e \in E} d(e)C(e) = 1$. The object is to route the commodities along paths in a way that minimizes the total distance of all paths used. In general, a UMFP has a feasible flow f if and only if for every legal distance function, the sum of all path distances is at most $\frac{1}{f}$. (This fact can be proved using the duality theory of linear programming [9].) In other words,

$$f = \frac{1}{\max_d \{\text{minimum total path length}\}}. \quad (5)$$

Given an n-node graph with capacities C and weighted flux α, and any distance function d such that $\sum^{e \in E} d(e)C(e) = 1$, we show in Section 2 how to route paths between all pairs of vertices so that the total path weight is at most $O(\frac{n \log n}{\alpha}) = O(\frac{\log n}{\mathcal{S}})$. This will be sufficient to prove Theorem 1 since we can then conclude that there is a feasible flow of size $\Omega(\frac{\mathcal{S}}{\log n})$.

It is worth noting that it is, of course, well-known how to compute shortest paths in a network in polynomial time. The tricky part of proving Theorem 1 is showing that paths with small total weight exist no matter how the adversary places the weights. It is also necessary for some of the applications to find the shortest paths in a way that reveals a cut whose size is inversely proportional to the total path length. Hence we must use a nonstandard short paths type of algorithm.

We suspect that our method for proving the existence of paths with short total length may be directly useful in other contexts such as routing paths around faults or costly (e.g., congested) components in a distributed network. In this paper, however, we use this technique only indirectly – in the context of the dual problem of finding sparsest cuts.

1.3 Approximating Flux

From Equations 3 and 4, we can quickly deduce that

$$\frac{nf}{2} \leq \alpha \leq O(fn \log n) \quad (6)$$

for any weighted graph. Hence, we can approximate the flux of any graph up to a $\Theta(\log n)$–factor simply by finding the optimal value of f for the associated UMFP. Unfortunately, this approach is not sufficient to actually locate an approximate minimum quotient separator, however.

In order to find an actual cut, we first solve the corresponding UMFP and its dual in order to construct the worst possible distance function $d : E \rightarrow R^+$. We then use the path routing algorithm described in Section 2 to find paths between all pairs of vertices with small total path length. As the algorithm progresses, we keep track of the minimum edge expansion cut found up to that point. At the end, we output the best such cut ever encountered. The algorithm has the nice property that the total path length is at most $O(\frac{n \log n}{\alpha_{obs}})$ where α_{obs} is the sparsest cut observed during the running of the algorithm. Hence by Equations 5 and 6 we have

$$\alpha_{obs} \leq O(\alpha \log n),$$

and thus our minimum observed sparsest cut is within a $\Theta(\log n)$ factor of optimal.

1.4 Approximating Separators, Bifurcators, Crossing Numbers and VLSI Layout Area

The algorithm for finding an approximate minimum quotient cut can be used to find polylog approximation algorithms for a variety of NP-hard combinatorial problems. For example, by recursively applying the approximate minimum quotient algorithm as in [18], we can obtain a $O(\log^2 n)$ times optimal approximation algorithm for finding minimum $\frac{1}{k}, \frac{k-1}{k}$ graph separators for any $k \geq 3$. Alternatively, we can obtian obtain a $O(\log n)$ times optimal approximation algorithm for finding approximate $\frac{1}{k}, \frac{k-1}{k}$ separators. (The difference in the latter result is that we pay the price for a $\frac{1}{k}, \frac{k-1}{k}$ separator but only get a $\frac{1}{ak}, \frac{ak-1}{ak}$ separator as output where $a > 1$ is a small constant.)

These algorithms can then be combined with the reductions of Bhatt and Leighton [1] to obtain:

1. a $O(\log^4 n)$ times optimal approximation algorithm for $c + n$ where c is the crossing number of an n–node bounded degree graph,
2. a $O(\log^{5/2} n)$ times optimal approximation algorithm for the $\sqrt{2}$–bifurcators of an n-node bounded degree graph, and
3. a $O(\log^6 n)$ times optimal approximation algorithm for the layout area of an n-node bounded degree graph.

At first glance, the crossing number approximation algorithm seems fairly weak since we are only approximating $c + n$ and only for bounded degree graphs. However, $c > \omega(n)$ for any graph with bisection width $\omega(\sqrt{n})$ as well as any graph with $4n$ or more edges [11]. These conditions are satisfied by most graphs. Moreover, the result can be partially extended to graphs with arbitrary degree. For example, the same result holds for regular (or near regular) graphs of arbitrary degree.

We should note that some minor effort is required to adapt the arguments of Bhatt and Leighton [1] to obtain these results. Full descriptions of the algorithms and their analysis will be included in the full paper.

1.5 Application to Packet Routing in Distributed Networks

Communication in a distributed network is an important area of research for which relatively little is known. 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 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 [15]. 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 this paper, we provide the first step towards a general solution to the problem by showing how any n-node bounded-degree graph, G, with flux α can simulate any other n-node bounded-degree graph, H, with delay $O(\frac{\log n}{\alpha})$ 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 H onto nodes of G. The main drawback of

the simulation result is that it requires off-line computation. However, once the off-line embedding is performed for one H, simulation of another H' can be performed on-line by using H to simulate H'. For example, by embedding a butterfly or other universal network into G off-line, we can then use G to simulate any CRCW PRAM algorithm in an on-line fashion with delay $O(\frac{\log^2 n}{\alpha})$ 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 (in this case, we embed the AKS-based sorting network described by Leighton in [13]), but if the sorting is to be done many times, the algorithm will be asymptotically much better than currently known techniques for general graphs. (Alternatively, we could use $O(\log^3 n)$ times optimal steps and obtain a smaller constant factor by off-line embedding a shuffle-exchange graph.)

In order to prove that any n-node bounded-degree graph G with flux α can simulate any other n-node bounded–degree graph H with delay $O(\frac{\log n}{\alpha})$, we first prove that H can be embedded in G with congestion and dilation $O(\frac{\log n}{\alpha})$. This is Theorem 2 and is the second of our two main results. (The *congestion* of an embedding is the maximum number of edges of H that traverse any edge of G. The *dilation* of an embedding is the maximum distance separating adjacent nodes of H when embedded in G.) The proof is then completed by applying the recent result of Leighton, Maggs and Rao [12] 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 nonconstructive. We can circumvent this drawback by introducing randomness and queues, however. In particular, we can use Theorem 2 to embed K_n in G with congestion $O(\frac{n \log n}{\alpha})$ and dilation $O(\frac{\log n}{\alpha})$. This can be performed in polynomial time off-line. We can then route a random permuation in $O(\frac{\log^2 n}{\alpha})$ expected steps using queues of size $O(\frac{\log n}{\alpha})$ by introducing a random delay selected uniformly from $[1, \frac{\log n}{\alpha}]$ for each packet and multiplexing the packets as they arrive at each edge. The details are not difficult to work out and will be included in the final paper.

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

1.6 Previous Work

Aside from the expander-related results cited in Section 1.5, there are relatively few previous results (that we are aware of) on the problems addressed in this paper. Although there are many max-flow min-cut results for multicommodity flow problems [14] [6], they tend to be exact results (or sometimes "plus one" results) for specialized problems, and we know of nothing resembling the approximate result in this paper. Duals to max-flow problems that involve min cuts have been constructed (see [19]), but they involve distance functions and are not natural min–cut problems in the normal sense.

Although there are numerous heuristic algorithms for separator related problems [5][10], as well as numerous average-case algorithms [4] [2] [3], there a very few previous results in ap-

proximation algorithms. For example, the best previously known algorithms for separators are due to Plaisted [16], who has a randomized $O(\sqrt{n})$ times optimal approximation algorithm and a randomized $O(\log n)$ times the square of optimal approximation algorithm. There has been substantial recent progress on the related problem of bandwidth minimization, however. In particular, Gonzalez and Kurki-Gowdara have used bundle-capacity flow techniques to construct a 2 times optimal approximation algorithm for this problem as well as for a variety of VLSI layout problems in [8]. Whether or not these techniques can be combined with those presented in this paper to obtain better approximation algorithms for separator related problems is a potentially interesting area of research.

2 Proof of Theorem 1

Following the analysis of Section 1.2, we need only show that for any n–node graph $G = (V, E)$ with weights $C : E \rightarrow R^+$ and distance function $d : E \rightarrow R^+$ such that $\sum^{e \in E} d(e)C(e) = 1$, it is possible to route paths between every pair of vertices with total distance $O(\frac{n \log n}{\alpha})$ where α is the weighted flux of G. This equivalent to showing that there is some vertex u of G for which

$$\sum^{v \in V} d(u, v) = O(\frac{\log n}{\alpha}),$$

where $d(u, v)$ denotes the distance along the selected path from u to v. For later reference we state this result as Lemma 1.

Lemma 1 *There is some vertex u of G for which*

$$\sum^{v \in V} d(u, v) = O(\frac{\log n}{\alpha}),$$

where $d(u, v)$ denotes the distance along the selected path from u to v.

The algorithm for finding such a node u consists of three Stages. The first Stage is a preprocessing stage. We start by placing an integral number of "distance tokens" on each edge. In particular, we place $\max\left\{1, \left\lceil \frac{d(e)}{s} \right\rceil\right\}$ tokens on edge e for every $e \in E$ where $s = \frac{1}{\sum_{e \in E} C(e)}$. Each token corresponds to distance s and serves to discretize the distance function. By using the tokens, we can break up long edges into discrete pieces, each with the same amount, s, of distance. Note that the total distance of each edge is not decreased by this process.

We next assign a weight to each token. In particular, tokens on edge e are assigned weight $C(e)$. Note that the total weight of all the tokens is at most

$$\sum^{e \in E} C(e)(1 + \frac{d(e)}{s}) = \frac{2}{s}$$

since $\sum^{e \in E} C(e)d(e) = 1$ and $\sum^{e \in E} C(e) = \frac{1}{s}$.

We now proceed with the second Stage of the algorithm. The purpose of Stage 2 is to grow a shallow depth region that contains a majority of the nodes. We do this by insisting that every time we grow the region by a token, we make the weight of the region heavier by an amount that is proportional to the expansion ratio. In particular, we execute the following steps:

1. Select a vertex $v \in G$, to initiate the region.

2. Extend the region by one token.

3. Let i denote the depth (i.e., radius) of this region as measured in tokens. (This is the number of times we have executed Step 2.) Let W_{i-1} denote the weight of tokens in the first $i - 1$ levels of the region. If $W_i \geq (1 + \epsilon)W_{i-1}$ where $\epsilon = \frac{\alpha n s}{9}$ then go to Step 2.

4. If the region contains $\frac{n}{2}$ or more nodes, halt and output the region.

5. Otherwise, remove the nodes in the region from the graph and go to Step 1.

We first prove that the algorithm eventually halts and outputs a region containing at least $\frac{n}{2}$ nodes. For the purposes of contradiction, assume otherwise and consider the union of the removed regions at the first instant that it contains at least $\frac{n}{4}$ nodes. Since, by assumption, no single removed piece contains $\frac{n}{2}$ removed nodes, the union of the removed regions has l nodes where $\frac{n}{4} \leq l \leq \frac{3n}{4}$ at this point. By the definition of α, this means that at least $\frac{\alpha n}{4}$ token weight leaves the union of the regions. From the condition at Step 3, we know that the ratio of the internal token weight to the boundary token weight of each removed piece is greater than $\frac{1}{\epsilon}$. Hence the same is true of the union of the removed regions, and we conclude that the union of the removed regions contains at least $\frac{\alpha n}{4\epsilon}$ total token weight. This is impossible, however, since $\frac{\alpha n}{4\epsilon} = \frac{9}{4s} > \frac{2}{s}$ which is the total token weight. Hence the algorithm halts and successfully outputs a region with at least $\frac{n}{2}$ nodes.

We next show that the depth of the output region (as measured in numbers of tokens) is at most $O(\frac{\log n}{\epsilon})$. To see why, we need only observe that $W_i \geq (1+\epsilon)W_{i-1}$ for each level of the output region, and hence that a region of depth d must have total token weight exceeding $(1+\epsilon)^{d-1}c_{min}$, where c_{min} is the minimum total capacity of the edges incident to any node. Since there is only $\frac{2}{s} = 2\sum^{e\in E} C(e)$ total token weight we can conclude that

$$(1+\epsilon)^{d-1}c_{min} \leq 2\sum^{e\in E} C(e),$$

and thus that

$$d = O(\frac{\log \frac{\sum^{e\in E} C(e)}{c_{min}}}{\log(1+\epsilon)}).$$

Without loss of generality we can assume that the capacity of any edge is at most nc_{min} (anything larger could never be used in a UMFP) and thus

$$\sum^{e\in E} C(e) \leq n^3 c_{min},$$

so

$$\log \frac{\sum^{e\in E} C(e)}{c_{min}} = O(\log n).$$

In addition,

$$\epsilon = \frac{ns\alpha}{9} = \frac{n\alpha}{9\sum^{e\in E} C(e)} \leq \frac{2}{9} < 1$$

since by averaging over all nodes, there must be some single node for which the weighted minimum edge expansion is at most $\frac{2\sum^{e\in E} C(e)}{n}$. Thus $\log(1+\epsilon) = \Theta(\epsilon)$, and $d = O(\frac{\log n}{\epsilon})$ as claimed.

At this point, we have constructed a region which contains at least $\frac{n}{2}$ nodes, and for which the sum of the distances from the root to all nodes in the region is at most $O(\frac{sn\log n}{\epsilon}) = O(\frac{\log n}{\alpha})$. It remains to extend the region to the remaining nodes with short total path length. This takes place during Stage 3.

Stage 3 consists of simply extending the region constructed in Stage 2 in a breadth first manner (one level of tokens at a time) until all nodes are reached. Although individual paths constructed during Stage 3 can be much longer than those constructed in Stage 2 (and this can be required in certain examples), their total length will still be small.

To prove that the total path length is small for the full region, we let n_i denote the number of nodes still not in the region after it is grown by i tokens during Stage 3. For example, n_0 is the number of nodes not in the region at the end of Stage 2. Since precisely, n_i paths will traverse the ith level of Stage 3, the total path length contributed by nodes in Stage 3 is at most

$$O(\frac{\log n}{\alpha}) + s\sum_{i=0}^{\infty} n_i.$$

The first term comes from the portion of the paths in the Stage 2 portion of the region, and the second term comes from the portion of the paths routed in Stage 3.

The sum $\sum_{i=0}^{\infty} n_i$ cannot be too large, however, since if after level i, there are still n_i unseen nodes, the weight of all the tokens used at level i must be at least αn_i. Since the total token weight is at most $\frac{2}{s}$, we can conclude that $\sum_{i=0}^{\infty} n_i \leq \frac{2}{s\alpha}$ and thus that

$$s \sum_{i=0}^{\infty} n_i \leq \frac{2}{\alpha}.$$

Hence the sum of all path lengths is at most $O(\frac{\log n}{\alpha})$, as claimed.

This completes the proof of Lemma 1 and hence Theorem 1. In order to construct the algorithm for finding approximate sparsest cuts as described in Section 1.3, however, we still need to check that the total weight of the paths in the region is at most $O(\frac{\log n}{\alpha_{obs}})$ where α_{obs} is the sparsest cut observed during the construction of the region. This is easily done, since we simply replace α with α_{obs} in all of the analysis for the performance of the algorithm. Note the algorithm is run the same way (ϵ doesn't change) but the analysis can be improved by using α_{obs} instead of α.

3 Proof of Theorem 2

Our second main result can be formally stated as follows.

Theorem 2 *Given any n-node bounded degree graph H, and any $1-1$ embedding of the nodes of H onto the nodes of G, the edges of H can be routed as paths in G with congestion and dilation $O(\frac{\log n}{\alpha})$.*

By Lemma 1 and standard duality arguments we can deduce that the commodities induced by the edges of H can be satisfied in G with congestion $O(\frac{\log n}{\alpha})$. (Here congestion corresponds to capacity in the UMFP.) We can then make use of Raghavan's [17] method of converting fractional flows into single paths to show that H can be embedded in G with $O(\frac{\log n}{\alpha})$ congestion. These peripheral details will appear in the full paper.

In order to produce paths with simultaneous congestion and dilation $O(\frac{\log n}{\alpha})$, however, we have to extend the proof of Lemma 1 so that all paths used have at most $O(\frac{\log n}{\alpha})$ edges in addition to total (token) distance $O(\frac{\log n}{\alpha})$. The result will then follow by restricting the linear program so that only paths with distance $O(\frac{\log n}{\alpha})$ are allowed.

To construct a region with small *edge depth* as well as small total path distance (measured as in Section 2 with the distance function), we need to modify the region construction algorithm of Lemma 1. In particular, we need only modify stage 3 since the edge depth of the region produced by Stage 2 is bounded by the token depth, which was shown to be at most $O(\frac{\log n}{\epsilon}) = O(\frac{\log n}{ns\alpha})$. This is at most $O(\frac{\log n}{\alpha})$ since $s = \Theta(\frac{1}{n})$ for an n–node bounded–degree graph with edge capacities all of which are one.

To modify Stage 3, we examine each level to see whether or not the number of nodes on the level exceeds the fraction $\frac{1}{2d_{max}}$ of the number of tokens, where d_{max} is the maximum degree of the graph G. If not, then we insert additional tokens in front of all the nodes that were on the level in order to "push" them to the next level. Otherwise, we continue as before.

Now we show that we do not insert too much token weight. Consider that at any level where tokens are inserted there are less than $\frac{\text{the number of tokens}}{2d_{max}}$ nodes in the level, and we need to use at most d_{max} tokens to push out any node. Thus we use less than $\frac{\text{the number of tokens}}{2}$ inserted tokens to push out the nodes. In addition, each node that is pushed out can push out at most d_{max} tokens with it so at least half the tokens that start in the level stay in this level. Hence, we can add only as many tokens as we already have which means that the number of tokens at most dou-

bles.

Thus the sum of the path lengths can at most double since (as in Lemma 1) the total distance of path segments formed in Stage 3 is $O(\frac{\log n}{\alpha})+ s\sum_{i=0}^{\infty} n_i$ and $\alpha\sum_{i=0}^{\infty} n_i$ is upper bounded by the number of tokens.

To see that the edge depth of the region is $O(\frac{\log n}{\alpha})$, we restrict our attention to those levels for which the number of nodes encountered exceeds $\frac{1}{2d_{max}}$ of the number of tokens (i.e., edges) encountered. All nodes are on these levels by our construction. Since the total number of nodes reached decreases by a factor of $(1 - \frac{\alpha}{2d_{max}})$ every time such a level is encountered during this stage, we can conclude that the edge depth of the part of the region constructed during Stage 3 is at most $O(\frac{d_{max}\log n}{\alpha})$. Finally for bounded degree graphs this is $O(\frac{d_{max}\log n}{\alpha}) = O(\frac{\log n}{\alpha})$.

This concludes the proof of Theorem 2.

4 Remarks

We are in the process of investigating possible extensions of this work. Potential questions of interest include:

1. Can the routing result of Section 1.5 be expanded to include unbounded degree networks by adding a diameter constraint?
2. Can our methods be extended to find approximate node separators?
3. Can the results of Section 1.4 be extended to approximate $\frac{1}{2} - \frac{1}{2}$ separators?
4. Can the technique of Section 1.5 be extended to improve the Peleg-Upfal disjoint paths result for expanders [15]?
5. Are there other combinatorial problems that can be approximated using these techniques?

We suspect that many of these questions can be answered affirmatively, but have not yet worked through the details.

5 Acknowledgements

We are grateful to David Shmoys and Eva Tardos for their helpful conversations. We would also like to acknowledge the work of Shahrokhi and Matula which inspired our research.

References

[1] S.N. Bhatt and F.T. Leighton. A framework for solving vlsi graph layout problems. *Journal of Computer and System Sciences*, 28(2), 1984.

[2] B. Bollobas, T.I. Fenner, and A. M. Frieze. An algorithm for finding hamiltonian cycles in random graphs. In *17th Annual Symposium on Theory of Computing, ACM*, page 430, 1985.

[3] R. Boppana. Eigenvalues and graph bisection: an average case analysis. In *28th Annual Symposium on Foundations of Computer Science, IEEE*, page 280, 1987.

[4] T. Bui, S. Chaudhuri, F.T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. In *FOCS*, pages 181–192, 1984.

[5] C.M. Fiduccia and R.M. Mattheyes. A linear time heuristic for improving network partitions. In *Design Automation Conference*, 1982.

[6] A. Frank. *Graph Connectivity and Network Flows.* Budapest, Hungary, 1979.

[7] M.R. Garey and D.S. Johnson. *Computers and Intractibility : A guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[8] T. F. Gonzalez and S. Kurki-Gowdara. *An Approximation Algorithm for the Via Placement and Related Problems.* Technical Report, University of California at Santa Barbara, December 1987.

[9] M. Iri. On an extension of the maximum-flow minimum cut theorem to multicommodity flows. *Journal of Operations Research Society of Japan*, 5(4):697–703, December 1967.

[10] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Tech. Journal*, 291–307, February 1970.

[11] F. T. Leighton. *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks.* MIT Press, 1983.

[12] F. T. Leighton, Bruce Maggs, and Satish Rao. Universal routing algorithms. In *Appearing 28th Annual Symposium on Foundations of Computer Science, IEEE*, 1988.

[13] F.T. Leighton. Tight bounds on the complexity of sorting. *IEEE Trans. on Computers*, 344–354, April 1985.

[14] D.W. Matula and F. Shahrokhi. *The Maximum Concurrent Flow Problem and Sparsest Cuts.* Technical Report, Southern Methodist University, March 1986.

[15] D. Peleg and E. Upfal. The token distrubution problem. In *27th Symposium on Foundations of Computer Science*, pages 418–427, 1986.

[16] D. Plaisted. Personal Communication.

[17] P. Raghavan. Probabilistic constuction of deterministic algorithms: approximating packing integer programs. In *27th Symposium on Foundationd of Computer Science*, pages 10–18, 1986.

[18] S. Rao. Finding near optimal separators in planar graphs. In *28th Annual Symposium on Foundations of Computer Science, IEEE*, page 225, 1987.

[19] F. Shahrokhi and D.W. Matula. *The Maximum Concurrent Flow Problem.* Technical Report, New Mexico Tech, March 1986.