Designing Multi-Commodity Flow Trees

Samir Khuller * Balaji Raghavachari † Neal Young ‡

Abstract

The traditional multi-commodity flow problem assumes a given flow network in which multiple commodities are to be maximally routed in response to given demands. This paper considers the multi-commodity flow network-design problem: given a set of multi-commodity flow demands, find a network subject to certain constraints such that the commodities can be maximally routed.

This paper focuses on the case when the network is required to be a tree. The main result is an approximation algorithm for the case when the tree is required to be of constant degree. The algorithm reduces the problem to the minimum-weight balanced-separator problem; the performance guarantee of the algorithm is within a factor of 4 of the performance guarantee of the balanced-separator procedure. If Leighton and Rao's balanced-separator procedure is used, the performance guarantee is $O(\log n)$. This improves the $O(\log^2 n)$ approximation factor obtained by a direct application of the balanced-separator method.

1 Introduction

Let a graph $G = (V, E)$ represent multicommodity flow demands: the weight of each edge $e = \{a, b\}$ represents the demand of a distinct commodity to be transported between the sites a and b. Our goal is to design a network, in which the vertices of G will be embedded, and to route the commodities in the network. The maximum capacity edge of the network should be low in comparison to the best possible in *any* network meeting the required constraints. For example, the weight of each edge could denote the expected rate of phone calls between two sites. The problem is to design a network in which calls can be routed minimizing the maximum bandwidth required; the cost of building the network increases with the required bandwidth.

We consider the case when the network is required to be a tree, called the *tree congestion problem*. Given a tree in which the vertices of G are embedded, the load on an edge e is defined as follows: delete e from T. This breaks T into two connected components. If S is

*Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Research currently supported by NSF Research Initiation Award CCR-9307462. E-mail : samir@cs.umd.edu.

†Department of Computer Science, The University of Texas at Dallas, Box 830688, Richardson, TX 75083-0688. E-mail : rbk@utdallas.edu. Part of this work was done while this author was visiting UMIACS.

‡Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. E-mail : young@umiacs.umd.edu. Research supported in part by NSF grants CCR-8906949 and CCR-9111348.

the set of vertices from G in one of the connected components, then $load(e)$ is equal to

$$W(S, \bar{S}) = \sum_{(x,y) \in E, x \in S, y \in \bar{S}} w(x, y).$$

In other words, the demand of each edge $e = \{a, b\}$ in G, maps to the unique path in T from a to b, and loads each edge on the path. The load of a single edge is the sum of the demands that load this edge.

In this paper we study two different versions of this problem.

1.1 Routing Tree Problem

The following problem was proposed and studied by Seymour and Thomas [ST].

Definition 1 *A tree T is called a* routing tree *if it satisfies the following conditions:*

- *The leaves of T correspond to vertices of G.*
- *Each internal vertex has degree 3.*

The congestion of T is the maximum load of any edge of T. The congestion of G, denoted by β_G, is defined to be the minimum congestion over all routing trees T of G.

We would like to find a routing tree T with minimum congestion (that achieves β_G).

Seymour and Thomas showed that this problem is NP-hard by showing that graph bisection can be reduced to this problem. They also showed that in the special case when G is planar, the problem can be solved optimally in polynomial time.

We provide a polynomial time approximation algorithm for the congestion problem when G is an arbitrary graph. Our algorithm computes a routing tree T whose congestion is within an $O(\log n)$ factor from the optimal congestion (Section 3). The algorithm extends to the case when the routing tree is allowed to have vertices of higher degree.

1.2 Congestion Tree Problem

We also study the case when T is required to be a spanning tree of a given feasibility graph F. We show that the problem is NP-complete (Section 4). In the special case when F is complete, we show that an optimal solution can be computed in polynomial time.

1.3 Main Ideas

Our algorithm is a simple divide-and-conquer algorithm that uses the Leighton-Rao [LR] balanced separator algorithm to split the graph. By a naive application of the LR algorithm, one obtains an $O(\log^2 n)$ approximation factor. Our main contribution is to show that by a subtle application of LR, one can actually obtain an $O(\log n)$ approximation factor. We believe that this kind of an application of LR will prove to be useful in obtaining better approximation ratios for other problems as well.

2 Preliminaries

A *cut* in a graph G is a set of edges whose removal separates G into two disconnected pieces S and $\bar{S} = V \setminus S$. A cut can be represented by the vertex set S. The *weight* of a cut S, denoted by $W(S, \bar{S})$, is the sum of the weights of those edges which have one endpoint in S and one endpoint in $\bar{S}$. We use $W(v)$ to refer to the sum of the weights of the edges incident to v. A cut S is *b-balanced* if $b \cdot n \leq |S| \leq (1-b) \cdot n$. The definition is extended to the case when vertices are weighted as follows. Let U be a non-negative weight function on the vertices and let $U(S)$ be the sum of the weights of all the vertices in S. A cut S is b-balanced if

$$b \cdot U(V) \leq U(S) \leq (1-b) \cdot U(V)$$

Definition 2 *For $b \leq 1/3$, a* λ-approximate minimum b-bisector *is a b-balanced cut whose weight is at most λ times the weight of a minimum-weight $\frac{1}{3}$-balanced cut.*

The following result was proved by Leighton and Rao ([LR], Section 1.4).

Theorem 2.1 *Let G be a graph with non-negative weights on the edges (without vertex weights). It is possible to compute an $O(\log n)$-approximate minimum $\frac{1}{4}$-bisector of G in polynomial time.*

The above theorem was extended to the case when vertices are given non-negative weights by Tragoudas [Tr].

Theorem 2.2 *Let G be a graph with non-negative weights on the edges and vertices. It is possible to compute an $O(\log n)$-approximate minimum $\frac{1}{4}$-bisector of G in polynomial time.*

Definition 3 *Let T be a tree and let u be a vertex of degree two in T. Let v and w be the neighbors of u. The following operation is said to* short-cut *u in T – delete u from T and add the edge $\{v, w\}$. To short-cut T is to delete all vertices of degree two by short-cutting them in arbitrary order.*

3 Routing Tree Problem

$W(v)$ corresponds to the total weight between v and other vertices and is called the load of a vertex. Note that the load of any vertex v is a lower bound on β_G, because the edge incident to the leaf corresponding to v in any routing tree has to handle this load.

Lemma 3.1 *For any vertex v, $W(v) \leq \beta_G$.*

Given a procedure to compute a λ-approximate minimum b-bisector, our algorithm finds a routing tree whose congestion is at most λ/b times the optimal congestion.

3.1 Lower Bounds

We show two ways of finding lower bounds on the weight of the optimal solution. First, we show that the weight of a minimum-weight balanced separator is a lower bound on β_G. Second, we show that the optimal solution for the problem in a subgraph G' induced by an arbitrary set of vertices $V' \subset V$ is a lower bound on the optimal solution of G. This implies that an optimal solution to a sub-problem costs no more than any feasible solution to the whole problem.

Lemma 3.2 *Let $G = (V, E)$ be a graph with non-negative weights on the edges. Suppose we are given a non-negative weight function $U(v)$ on the vertices. Let the weight of each vertex be at most one-half of the total weight of all the vertices. Let Q be the weight of a minimum-weight b-balanced separator of G for any $b \leq 1/3$. Then $Q \leq \beta_G$.*

Proof. Let T be a routing tree with congestion β_G. Each edge e of T naturally induces a cut in G as follows: delete e from T to obtain subtrees T_1 and T_2. Let S_e be the set of vertices in G that are leaves of T_1 (this yields a cut in G). Clearly, $W(S_e, \overline{S_e})$ is the congestion on edge e and hence $W(S_e, \overline{S_e}) \leq \beta_G$. Since T is a tree of degree three, and by the assumption on the weights of vertices, it contains at least one edge e' which yields a b-balanced separator. Since Q is the minimum b-balanced separator of G we have $Q \leq W(S_{e'}, \overline{S_{e'}}) \leq \beta_G$. □

Lemma 3.3 *Let $G = (V, E)$ be a graph. Let H be a subgraph of G. Then $\beta_H \leq \beta_G$.*

Proof. Let T be a routing tree with congestion β_G. Generate a routing tree T_H for H from T as follows. Let V_H be the vertex set of H. Mark the leaves of T corresponding to V_H. Repeatedly delete the unmarked leaves of T until it has no unmarked leaves. Delete all vertices of degree two by short-cutting the tree, thus yielding T_H. It is easily verified that T_H is a routing tree for H with congestion bounded by β_G. □

3.2 The Routing Tree Algorithm

Discussion. Our basic approach is to subdivide the graph into pieces that are smaller by a constant fraction using an approximately minimum bisector. Since computing a minimum-weight balanced separator is also NP-hard, we use approximation algorithms designed by Leighton and Rao [LR] and Tragoudas [Tr] for computing approximately minimum-weight balanced separators (or approximate minimum bisectors). The solutions for the pieces are obtained recursively. All internal vertices of the solution tree have degree three except for the root. The two trees are glued together by creating a new root and making the roots of the pieces the children of the new root. If implemented naively, this procedure leads to an $O(\log^2 n)$ factor approximation. Using balancing techniques, we improve the performance ratio to $O(\log n)$.

Suppose S, a subset of the vertices representing a subproblem, is split into two pieces S_1 and S_2 using an approximate bisector. When the problem is solved recursively on the two pieces, the main obstacle to obtaining an $O(\log n)$ approximation is the following. In the worst case, it is possible that most of the load corresponding to $W(S, \bar{S})$ may fall on S_1 or S_2. If this happens repeatedly, an edge can be overloaded proportionally to its depth in the tree. *To avoid this, it is necessary to partition the demand from $\bar{S}$ roughly equally*

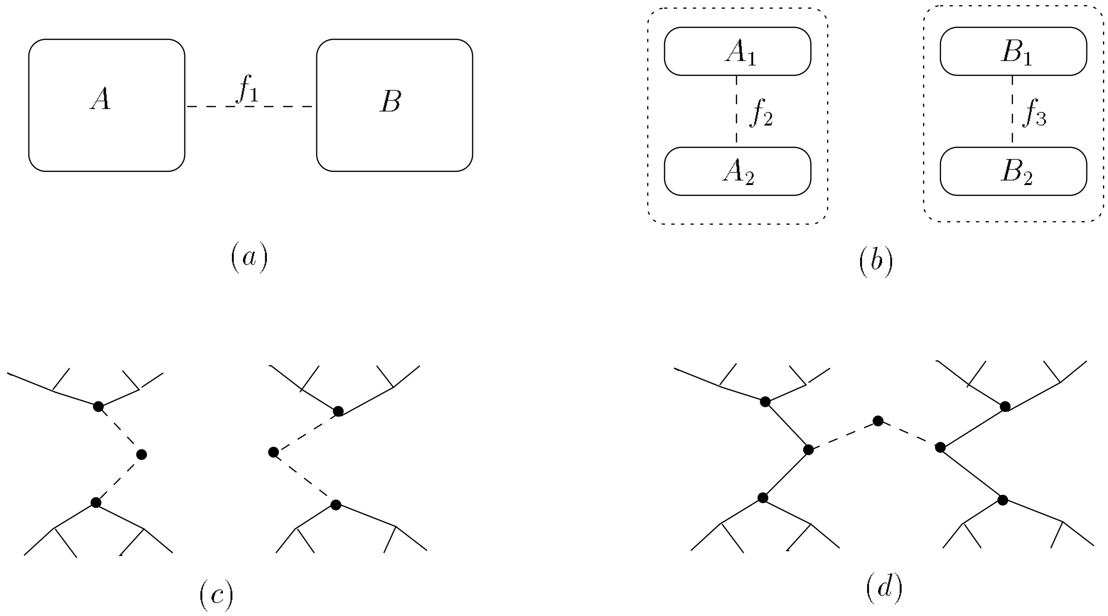

Figure 1: Example to illustrate algorithm.

among the pieces S_1 *and* S_2. The following idea solves the problem and leads to an $O(\log n)$ approximate solution. Suppose we define a weight $U(v)$ for each vertex v in S according to the amount of demand from v to the set $\bar{S}$. Now when we split S, we use a cut that splits the vertices of S into two sets of roughly equal *weight*. Lemma 3.2 guarantees that the minimum value of such a cut is a lower bound on β_S, which is a lower bound on β_G by Lemma 3.3. We illustrate the recursive step of the algorithm by an example in Fig. 1.

The algorithm first splits graph G into A, B by using an approximate bisector (without weighting the vertices). Each vertex in A is then assigned a weight equal to the total demand it has to vertices in $\bar{A}$. Similarly vertices in B are assigned weights corresponding to their demands from $\bar{B}$. The algorithm now recursively splits A and B by approximate bisectors with respect to the vertex weights. The problem is solved recursively on each piece. These recursive calls weight vertices similarly and return with respective trees as solutions for the pieces A and B as shown. By adding new edges and a new root vertex, the solution for the entire graph is obtained.

The algorithm given in Fig. 2 implements the above ideas. The procedureRoute-Tree(S) takes a subset of vertices S, and returns a routing tree for the graph induced by the vertices in S. This routing tree will either be a singleton vertex, or a tree in which each vertex has degree one or three, except for the root that has degree two. The routing tree is computed in a way so as to approximately "divide" the demand from the vertices in S to the vertices in $V - S$.

Analysis. Given a graph G, Route-Tree(V) returns a routing tree for G. To make sure that the root of the tree has degree three, we can discard the root by short-cutting it.

Let the algorithm use a λ-approximate minimum $\frac{1}{4}$-bisector in Line 6. If Leighton and Rao's [LR] balanced separator algorithm is used, $\lambda = O(\log n)$. The following theorem shows that the load of any edge is at most 4λ times the optimal congestion. We use induction to prove that our load-balancing technique splits the load properly.

Theorem 3.4 (Performance) *The algorithm in Fig. 2 finds a routing tree T for G such that $\beta_T \leq 4\lambda\beta_G$.*

Proof. The proof proceeds by induction on the level of recursion. In the first call of ROUTE-TREE, the algorithm splits G into two pieces S and $\bar{S}$ using an approximate bisector. It then finds routing trees for S and $\bar{S}$ and connects the two roots with an edge e. The load on e is $W(S, \bar{S})$. By Lemma 3.2, the weight of a minimum-weight balanced separator is a lower bound on β_G. The weight of the separator the algorithm uses is guaranteed to be at most λ times the weight of an optimal separator. Hence the load on edge e is at most $\lambda\beta_G$. This satisfies the induction hypothesis.

For the induction step, let us consider the case when we take a set S and split it into two pieces S_1 and S_2 (see Fig. 3). Let L be the load on the edge connecting the tree for S to its parent. Similarly, let L_i $(i = 1, 2)$ be the load on the edge connecting the tree for S_i to its parent. Inductively, $L \leq 4\lambda\beta_G$. We show that each $L_i \leq 4\lambda\beta_G$.

Let U be the weight function defined by the algorithm in this recursive call. Note that $L = U(S) = W(S, \bar{S})$ and $L_i = W(S_i, \bar{S_i}) = W(S_i, \bar{S}) + W(S_1, S_2)$. Also observe that $U(S_i) = W(S_i, \bar{S})$.

Case 1: If there is some vertex v in S whose weight $U(v)$ is more than $U(S)/2$, then we split S as $S_1 = \{v\}$ and $S_2 = S \setminus \{v\}$. Since $L_i = U(S_i) + W(S_1, S_2)$ and $U(S_1) > U(S)/2 > U(S_2)$ it follows that $L_1 > L_2$. This is because $U(S)$ is the sum of $U(S_1)$ and $U(S_2)$. It remains only to bound L_1. The demand from v, $W(v)$, is a lower bound on the congestion (by Lemma 3.1) and therefore $\beta_G \geq W(v) = L_1$. Hence both L_1 and L_2 satisfy the induction hypothesis.

Case 2: Otherwise, the algorithm distributed $U(S)$ into the weights of the vertices of S and then used a λ-approximate $\frac{1}{4}$-bisector of S. By the induction hypothesis, the edge from the subtree of S to its parent has a load L $(= U(S))$ of at most $4\lambda\beta_G$.

Since $W(S_i, \bar{S}) = U(S_i) \leq \frac{3}{4}U(S)$ and $W(S_1, S_2) \leq \lambda\beta_G$ (by Lemmas 3.2 and 3.3) we have:

$$L_i = W(S_i, \bar{S}) + W(S_1, S_2) \leq 3\lambda\beta_G + \lambda\beta_G.$$

□

Theorem 3.5 *The routing tree algorithm in Fig. 2 runs in polynomial time.* □

Corollary 3.6 *The algorithm in Fig. 2 finds in polynomial time a routing tree T for G such that $\beta_T = O(\log n)\beta_G$.*

Note: Our algorithm also handles the case when vertices of G are allowed to be internal vertices of the output tree. Lemmas 3.2 and 3.3 are valid in this case also. The lower bound in Lemma 3.1 weakens by a factor of 3. This lower bound is not critical to the performance ratio, so the performance ratio of the algorithm is unchanged.

Route-Tree(S) — *Find a routing tree for S.*

1 **If** $|S| = 1$ **then Return** S as a tree on a single vertex.
2 For each $v \in S$, fix its weight $U(v)$ to be $W(\{v\}, \bar{S})$.
Let the sum of the weights of the vertices in S be U_S.
3 **If** for any vertex v, $U(v) \geq U_S/2$ and $U_S \neq 0$ **then**
4 Route-Tree($S \setminus \{v\}$)
5 Create a new tree T by attaching the above tree and v as the children of a new root r. **Return** T.
6 Find an approximate minimum-weight $\frac{1}{4}$-balanced separator for the subgraph induced by S in G (if $U_S = 0$, find an unweighted balanced separator). Let this break S into pieces S_1 and S_2.
7 Route-Tree(S_1)
8 Route-Tree(S_2)
9 Create a new tree T by attaching the two trees generated above as the children of a new root vertex. **Return** T.

Figure 2: Approximation Algorithm to Find a Routing Tree

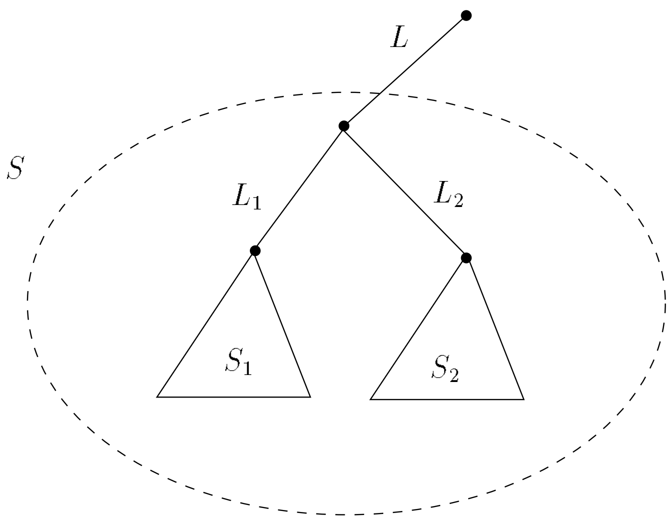

Figure 3: Inductive proof.

Our algorithm can be generalized to find routing trees when every internal vertex may have degree up to k, for any $k \geq 3$. We obtain the same $O(\log n)$ approximation factor, independent of k. An algorithm obtaining an approximation factor of n/k is straightforward and is useful as k approaches n.

4 General Congestion Problem

In this section we show that the following problem is NP-complete. The input to the problem is a demand network $G = (V, E)$, a "feasibility network" $F = (V, E')$, and an integer D. Each edge $e = \{a, b\}$ of G has a nonnegative weight $w(e)$ that represents the demand between the sites a and b. The problem is to find a tree T that is a subgraph of F, such that when the demands of the edges in G are mapped to the tree T the congestion on each edge is at most D.

The reduction is done from the k Edge-Disjoint Paths Problem, known to be NP-Complete [GJ].

k **Edge-Disjoint Paths Problem:** Given an undirected graph $H = (V, E)$, and sets $S = \{s_1, s_2, \ldots, s_k\}$ and $T = \{t_1, t_2, \ldots, t_k\}$ are there k mutually edge-disjoint paths $P_1, P_2, \ldots, P_k$ such that P_i connects s_i with t_i ?

It is easy to see that this problem can be reduced to the general tree congestion problem. For the reduction we construct F from H. For each vertex $u \in V$, if u has degree $d(u)$, we create a clique on $d(u)$ vertices, $u_1, u_2, ..., u_{d(u)}$. For each edge from v to w we introduce an edge from v_i to w_j where these are *distinct* vertices (not shared with any other edges). (Informally, each vertex is "exploded" into a clique, and the edges incident on the vertex are made incident on distinct clique vertices.) The demand graph G has edges between s_i and t_i (for all i). If there is a solution to the disjoint paths problem, clearly that yields a congestion tree with bandwidth one. The set of paths P_i can form cycles, but these cycles can be "pried" apart in F since we replaced each vertex with a clique. These can now be connected to form a congestion tree with bandwidth one.

If there is a solution to the congestion tree problem it is clear that this yields a solution to the edge-disjoint paths problem (the demand edge from s_i to s_j gets mapped to a path in the tree and causes a load of one on each edge). Since the bandwidth is restricted to one, no other path can use the same edge (even when we go from F to H).

Theorem 4.1 *The general congestion problem is NP-complete.*

An interesting open problem is to design approximation algorithms with nontrivial approximation factors for designing routing trees where the feasibility graph F is given in the input. In the special case when F is complete, it is easy to show that an optimal routing tree can be computed. In this case each edge of the routing tree is made to handle a load that is equal to the minimum cut in G separating two of its vertices. This follows from the result of Gomory and Hu [GH], who showed how to construct a tree which encodes *all* min-cuts in a graph. Gusfield [Gu] gave an algorithm to compute such trees efficiently.

Theorem 4.2 *If F is the complete graph, the problem of designing a routing tree with minimum congestion for an arbitrary demand graph G can be solved in polynomial time.*

Acknowledgments

We would like to thank the referee for useful comments.

References

[GJ] M. R. Garey and D. S. Johnson, “Computers and Intractability: A guide to the theory of NP-completeness”, *Freeman, San Francisco* (1979).

[GH] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of SIAM*, 9(4): 551–570, 1961.

[Gu] D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1): 143–155, 1990.

[LR] F. T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proc. 29th Annual Symp. on Foundations of Computer Science*, pages 422–431, October 1988, White Plains, NY.

[ST] P. Seymour and R. Thomas. Call routing and the rat catcher. *Workshop on Algorithms and Combinatorial Optimization*, March 1991. Atlanta, GA.

[Tr] S. Tragoudas. Improved approximations for the minimum-cut ratio and the flux. Technical Report 93-02, Computer Science Department, Southern Illinois University, 1993. Submitted to *Math. Syst. Theory.*