A Polynomial-time Tree Decomposition to Minimize Congestion

Chris Harrelson [*] Kirsten Hildrum [†] Satish Rao [‡]

ABSTRACT

Räcke recently gave a remarkable proof showing that any undirected multicommodity flow problem can be routed in an oblivious fashion with congestion that is within a factor of $O(\log^3 n)$ of the best *off-line* solution to the problem. He also presented interesting applications of this result to distributed computing. Maggs, Miller, Parekh, Ravi and Wu have shown that such a decomposition also has an application to speeding up iterative solvers of linear systems.

Räcke's construction finds a decomposition tree of the underlying graph, along with a method to obliviously route in a hierarchical fashion on the tree. The construction, however, uses exponential-time procedures to build the decomposition.

The non-constructive nature of his result was remedied, in part, by Azar, Cohen, Fiat, Kaplan, and Räcke, who gave a polynomial time method for building an oblivious routing strategy. Their construction was not based on finding a hierarchical decomposition, and this precludes its application to iterative methods for solving linear systems.

In this paper, we show how to compute a hierarchical decomposition and a corresponding oblivious routing strategy in polynomial time. In addition, our decomposition gives an improved competitive ratio for congestion of $O(\log^2 n \log\log n)$.

In an independent result in this conference, Bienkowski, Korzeniowski, and Räcke give a polynomial-time method for constructing a decomposition tree with competitive ratio $O(\log^4 n)$. We note that our original submission used essentially the same algorithm, and we appreciate them allowing us to present this improved version.

[*]Email: chrishtr@cs.berkeley.edu. Supported in part by a GAANN fellowship and NSF grant CCR-0105533.

[†]Email: hildrum@cs.berkeley.edu. Supported by UC MICRO #02-035.

[‡]Email: satishr@cs.berkeley.edu. Supported by NSF grant CCR-0105533.

Categories and Subject Descriptors

E.1 [**Data structures**]: Graphs and networks; F.2.2 [**Nonnumerical algorithms and problems**]: Routing and layout

General Terms

Algorithms, Theory

Keywords

Congestion, oblivious routing, tree decomposition, preconditioning

1. INTRODUCTION

In this paper we consider the following problem: we are given a weighted, undirected graph. We would like to preprocess the graph so that whenever flow needs to be routed from one vertex to another, it can be done without reference to any flow possibly being routed between any other pair of vertices. In other words, we would like to find a *fixed* flow routing that can be used for any set of demands. Such a construction is called *oblivious.*

Our goal is to find flow paths for all pairs of vertices, such that regardless of the flow demands, we can route with a low maximum congestion on any edge. Our quantitative measure of "low maximum congestion" is to compare ourselves with the best possible (non-oblivious, offline) routing scheme under the same flow demands.

Räcke [23] proved the existence of such a good oblivious routing scheme by hierarchically decomposing the graph, and then showing how the graph can approximately simulate the corresponding hierarchical tree.[1] This hierarchical tree can route flow with congestion at most that of the optimal offline flow on the original graph. Therefore, approximate simulation of the hierarchical tree by the original graph implies an approximately optimal routing scheme for the original graph itself. He also gives other applications of his technique. His construction, however, was not polynomial-time.

Maggs et al. [21] showed that Räcke's result yields a good family of preconditioners that provably speed up the iterative solution of linear systems. However, since Räcke's con-

[1]We would like to point out that these trees are very similar in spirit to the fat-trees developed by Leiserson [19], who used fat-trees to give similar results for grid graphs. This was in the context of finding universal networks for VLSI circuits, which are modeled as grid graphs.

struction is not polynomial-time, neither the preconditioners nor the oblivious routing strategies could be efficiently found.

For routing, this situation was remedied. Azar, Cohen, Fiat, Kaplan and Räcke [6] used linear programming to show that one could, in polynomial time, find the *optimal* oblivious routing scheme for any network. Their result applied to directed graphs as well as undirected graphs, and they relied on Räcke's result to give an upper bound on the relationship between the best offline and oblivious solutions for undirected graphs. They also showed that for *directed* graphs that oblivious routing schemes could be off by a factor of $\Omega(\sqrt{n})$ from the best offline schemes.

Their results, however, do not seem to apply to finding the preconditioners developed by Maggs et al. using Räcke's result. The preconditioners make essential use of hierarchical structure, which the Azar et al. construction does not provide.

We improve Räcke's result by making it constructive, and show that our decomposition achieves an $O(\log^2 n \log\log n)$ competitive ratio against the best possible off-line routing strategy. Since our algorithm actually constructs a hierarchical tree decomposition, it can be used to efficiently construct the preconditioners presented by Maggs et al.

Again we note that, independent of our result, Bienkowski, Korzeniowski and Räcke [8] present in this conference a polynomial time algorithm for constructing the decomposition tree. Moreover, they point out the competitive ratio is parameterized by the max-flow min-cut gap of the input graph.

1.1 Related Work

A totally distributed approach to routing multicommodity flows was developed by Awerbuch and Leighton [5]. Indeed, they gave a method that routes flows with a rate that is within a $(1+\epsilon)$ factor of optimal. The flows were routed by injecting packets at sources and removing packets at the corresponding sinks, and generally flowing packets "downhill" according to the number of packets queued at each node. Their approach could clearly incur quite some delay to build the "hills" that guide the packets to their ultimate destinations. Moreover, their approach was not oblivious, *i.e.*, a packet's path could be influenced by other flows.

Aspnes et al. [2] gave an $O(\log n)$-competitive, online algorithm, which is not oblivious. Awerbuch and Azar [4] gave a less centralized algorithm with the same bounds.

The technique of using a tree decomposition for routing has also found application in a data management problem [22, 26].

The work presented by Maggs et al. builds on an unpublished manuscript by Vaidya [24]. Vaidya's ideas were further developed by Gremban and Miller, who first proposed the framework of using a tree decomposition-based preconditioner and gave explicit constructions for 2D meshes. This work is described in Gremban's thesis [14]. Recently, the ideas have further been extended in [9] and explained in [7].

As noted above, the results are parameterized on the max-flow min-cut gap η of the class of graphs in question. In terms of η, our algorithm gives an $O(\eta \log n \log\log n)$ approximation algorithm. For example, on planar graphs, η is a constant. On graphs that exclude the minor K_r, $\eta = O(r^2)$ [12] (their results improves the result in [17], which showed that $\eta = O(r^3)$).

A polynomial lower bound on oblivious, *deterministic* routing has been shown for hypercubes in [10], and subsequently improved by [16]. The situation is substantially different for oblivious, *randomized* algorithms on the hypercube, where the lower bounds are logarithmic. See [28, 1, 11] for details.

1.2 Techniques

The core of Räcke's proof is a hierarchical decomposition of the underlying graph. The decomposition is constructed by choosing a series of nested edge cuts of subgraphs into smaller and smaller pieces, until all pieces contain only one vertex. This is also sometimes called a *laminar* decomposition, since the subgraphs resulting from the cuts form a laminar set.

There is a natural tree associated with any such hierarchical decomposition, in which each node represents a subgraph, and the children of a node represent the different connected components after the edge cut of the parent. The weight of an edge from child to parent is the size of the cut between the child subgraph and the rest of the graph (not just to the parent).

Now consider routing a particular multicommodity flow between the leaves of the tree, using only the edges of the tree. It is not hard to see that routing a multicommodity flow on the associated decomposition tree of a graph has lower maximum edge congestion than the optimal flow routing on the original graph. The reason for this is that the edges in the tree represent upper-bounds on the capacity between a subgraph and the rest of the graph (recall that each edge's weight is equal to the sum of the capacities of the edges between these two components).

We would like to use the tree decomposition to give us "hints" as to how to route obliviously in the original graph. The path between two leaves in the tree induces a nested series of edge cuts in the graph; the "hint" Räcke uses is to route across each of these cuts sequentially. Each such cut is a potential bottleneck in the graph for the flow.

Somewhat counter-intuitively, we would like to find small cuts since these cuts are bottlenecks in that they represent the true capacity of the graph. Of course, the cuts in a particular decomposition tree may or may not accurately reflect the true bottlenecks in the graph. The challenge is to compute a decomposition in which each edge cut is a "true" bottleneck. Approximate max-flow min-cut theorems like Leighton-Rao [18] give a flow and a corresponding bottleneck, at least approximately.[2]

In summary, Räcke's technique is to find a good tree decomposition, then show that he can route in the original graph almost as well as in the tree by simulating the tree with a hierarchical set of multicommodity flows.

This summary is an injustice to the clever arguments that Räcke develops. For example, his algorithm alternatively cuts subgraphs whose exit edges are not well connected and merges pieces whose exit edges cannot communicate. The tradeoffs are quite subtle. Moreover, he pays only a logarithmic factor for his trouble. He pays another logarithmic factor for the recursion depth, and a third for the approximate relationship between multicommodity flows and the cut lower bounds. This leads to his final bound of $O(\log^3 n)$.

We proceed differently; where Räcke divided subpieces by searching exhaustively, we use the algorithm implicit in the

[2]In fact, the set of decomposition trees can be viewed as a family of routing congestion lower bounds on the original graph.

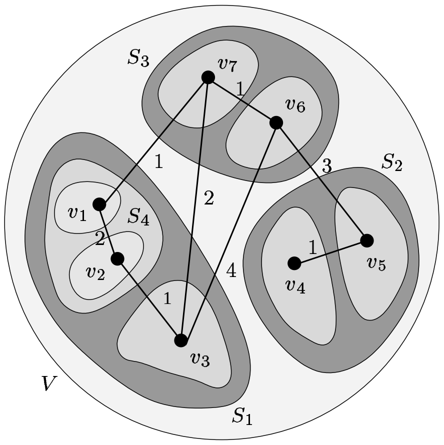

Figure 1: A graph on six vertices and its visual decomposition into laminar sets.

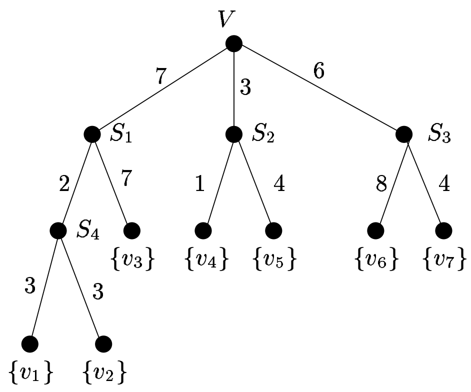

Figure 2: The resulting tree from the decomposition.

approximate max-flow min-cut theorem in [18] to either find a flow or find a good cut. That is, we recursively construct both the flows and the decomposition. This can be compared to Räcke who constructs the decomposition and then argues that no cut is bad, a postieri. We believe this approach is simpler in addition to being constructive.

We remark that the procedure appears worse than Räcke's, since he finds the optimal cuts at the time of the decomposition. Indeed, there are two ways in which the technique appears worse: first, the error in the approximation procedure is squared; second, we seem to need an extra logarithmic factor in order to ensure termination of the procedure.

Räcke, however, pays a logarithmic factor at the end when using the approximate max-flow min-cut theorems to show that the flow exists. While we don't find optimal cuts, we simultaneously construct cuts and flows and pay a logarithmic squared factor as we go, which is already contained in our "squared" error.

In addition, we further examine the cut/merge procedures of Räcke a bit more carefully. His analysis does not take credit for the number of cuts that happen in each call to these procedures, which may result in very small subpieces. By carefully taking credit for this, we can reduce the combined cost of each level and the depth of the tree from $\log^2 n$ to $\log n \log\log n$.

Finally, we are able to avoid paying the squared logarithmic factor for termination by allowing ourselves to recompute cuts from a higher level that now appear suboptimal.

Thus, we obtain a bound of $O(\log^2 n \log\log n)$ in terms of competitiveness with offline multicommodity flows.

We also note that, although this paper talks exclusively in terms of flows, it is natural to use these flows to randomly route with expected congestion equal to our bounds. To do this, simply treat the flows as probability distributions over paths, and route on a path according to its probability. If a sufficient number of packets are sent on each route, we can use Chernoff bounds to show that the actual congestion experienced is close to the expectation with high probability.[3]

[3]This assumption is not restrictive, however, since if the number of packets is small, then we trivially meet the competitive ratio requirements.

2. PRELIMINARIES

2.1 General notation

We will be working implicitly throughout with an undirected, weighted graph $G = (V, E)$. Let $B(u, v) \in \mathbb{Z}^{\geq 0}$ be the weight of the edge (u, v). We will usually refer to this as the *bandwidth* of the edge.

We will use capital letters like S, where $S \subseteq V$, to refer to both a subset S of the vertices of G, and the induced subgraph on them.

For any $u \in S$ and $S \subseteq V$, let $w_S(u) = \sum_{v \notin S} B(u, v)$; in words, $w_S(u)$ is the bandwidth u has to the outside of S. Generalizing, if $\mathcal{D} = \{U_i\}_i$ is a collection of disjoint subsets (a *partition*) of S, then for any i and any $u \in U_i$, let $w_{\mathcal{D}}(u) = \sum_{v \notin U_i} B(u, v)$.

Similarly, for $U \subseteq S$, let $w_S(U) = \sum_{v \in U} w_S(v)$ and for any partition $\mathcal{D} = \{U_i\}_i$, let $w_{\mathcal{D}}(U) = \sum_{v \in U} w_{\mathcal{D}}(v)$. This is the amount of bandwidth crossing the boundaries of the pieces of the partition $\mathcal{D}$, together with all bandwidth that leaves S.

Let $\operatorname{cap}(U_1, U_2) = \sum_{u \in U_1, v \in U_2} B(u, v)$.

Let $\delta_P(S) = \frac{1}{8\gamma\left(\log \frac{|P|}{|S|}\right) \log\log n}$, where γ is a constant which comes from a sparsest cut approximation algorithm (see subsection 2.2). In what follows, we discuss a decomposition tree, and will omit the subscript P; it should be understood from context that P is the parent of S in the decomposition tree.

2.2 Notation about flows

We will need to route many multicommodity flows in order to compute the decomposition tree; all of them, however, will have the same general form. Let $\textsc{UniformFlow}(S, \mathcal{D})$ denote a multicommodity flow over a subgraph S and a partition $\mathcal{D}$ of S. If $u \in U_1$ and $v \in U_2$, where $U_1, U_2 \in \mathcal{D}$, then the demand from u to v in the flow graph $\textsc{UniformFlow}(S, \mathcal{D})$ is defined as

$$w_{\mathcal{D}}(u) \frac{w_{\mathcal{D}}(v)}{w_{\mathcal{D}}(S)} \tag{1}$$

We will be especially interested in the *maximum congestion* under which these flows can be routed. This is the multiplicative inverse of what is called the *maximum concurrent* *multicommodity flow* under these demands. This is the maximum value q^* such that $q^* \cdot \text{demand}(u, v)$ units of flow can be routed simultaneously for all pairs u and v without exceeding the bandwidth of any edge. We note that q^* and the flows can be computed in polynomial time by linear programming. We will denote q^* by $\text{UniformFlowValue}(S, \mathcal{D})$.

We will also consider the corresponding sparsest cut problem, whose solution is a lower bound on this multicommodity flow problem. The sparsest cut of a graph under some set of demands is the edge cut $(U, S \setminus U)$ which minimizes $\frac{\text{cap}(U,S\setminus U)}{\text{demand}(U,S\setminus U)}$. We will use the notation $\text{UniformCut}(S, \mathcal{D})$ to represent a $\gamma \log n$-approximate sparsest cut on the same demand graph as $\text{UniformFlow}(S, \mathcal{D})$.

The multicommodity flow demands here are sometimes called "product" multicommodity flow [18]. A $\gamma \log n$-approximate sparsest cut can be found for demands of this type in polynomial time [18, 20, 3]. For this type of demand, a bound can be proven on the dilation of the flow in addition to its congestion (the dilation of a flow is the maximum length of any flow path) [18]; the bound is stated in terms of the expansion of the graph. This bound becomes important for the application of Maggs et al. We refer the reader to the relevant literature for details. In this paper will will simply make use of the fact that these flows can be computed in polynomial time.

2.3 Conventions and technical issues

We will assume throughout that:

- All edge weights in the graph are bounded to within a fixed polynomial factor of each other;
- log denotes the logarithm function to base two;
- and n is at least 4, which makes $\log \log n \geq 1$.

The bandwidth on any edge in the graph is a nonnegative integer. Imagine splitting up the bandwidth on an edge (u, v) with bandwidth $B(u, v)$ into $B(u, v)$ pieces with bandwidth one. When we refer to a "unit of bandwidth" from that edge, we will refer to one of these pieces. When we refer to the "border bandwidth" of a subgraph S, we mean the bandwidth that contributes to $w_S(S)$. Similarly, when we refer to a "unit of border bandwidth" of S, we refer to a unit-sized piece of $w_S(S)$. We speak in terms of units of bandwidth in our discussions and proofs.

3. THE ALGORITHM

Our algorithm builds a tree T one node at a time. Each node in the tree represents a subgraph of the graph, and the children of a node are a disjoint partition of the parent's subgraph. In order to create T, we will start with V at the root and repeatedly create children for each of the (current) leaves of the tree (by partitioning the set) until the leaves are all sets of size one. In what follows, we refer to a node in the tree as either a node or a subset of the graph.

The algorithm produces a tree T of the graph such that, for each node S in T, and S's set $\mathcal{D}$ of children:

- We route $\text{UniformFlow}(S, \mathcal{D})$ with low congestion. (In fact, we show that the congestion is $O(\frac{\log n}{\delta(S)})$.)
- Every $U \in \mathcal{D}$ has size at most $|S|/2$.

Consider some node S during the creation of the tree. In this section we show how to partition S into a set of children, $\mathcal{D}$, with the required properties. Our strategy is conceptually simple: start with some partition $\mathcal{D}$ of S and try to route $\text{UniformFlow}(S, \mathcal{D})$. If we can, then we we are done with S and can recurse on the elements of $\mathcal{D}$. Otherwise, we can either modify $\mathcal{D}$ and try to route again, or tell S's parent that it cut incorrectly.

More explicitly, we will think of the partition as having two stages. The first phase, the *merge* phase, will consist of a partition into *merge* sets, and the second will be a further partition of each merge set into what we refer to as *bandwidth* sets. The second phase is happening at the same time as the first phase for the children of the current node. At the end of the merge phase, we will do one of the following:

- Output a partition $\mathcal{D}$ of S, on which we can route $\text{UniformFlow}(S, \mathcal{D})$.
- Reject S and return a cut of it to S's parent. We will refer to such a cut as a *bad child event.*

The *bandwidth* phase, will consist of handling bad child events from S's children in $\mathcal{D}$. Each time such an event happens, we will need to revise $\mathcal{D}$ and recurse again on the new pieces.

For extra clarity, the algorithm appears below in pseudocode. Calling $\text{MakeTree}(V)$ will output a nested set of sets which represents the tree decomposition.

```
Algorithm $\text{MakeTree}(S)$:

$\mathcal{D} := \{\{v\} : v \in S\}$
// The merge phase:

$q := \text{UniformFlowValue}(S, \mathcal{D})$
if $q \leq \delta(S)/\log n$ then
  $(U, S \setminus U) := \text{UniformCut}(S, \mathcal{D})$
  // assume that $|U| \leq |S \setminus U|$
  if $w_{\mathcal{D}}(U) - w_S(U) \geq \frac{1}{2\gamma\delta(S)}\text{cap}(U, S \setminus U)$ then
    $\mathcal{D} := U \cup (\bigcup_{X \in \mathcal{D}} X \cap (S \setminus U))$
  else return failwith $(U, S \setminus U)$

// The merge phase has ended, and we can route the
// desired merge flow.

// The bandwidth phase:

Mark all $U \in \mathcal{D}$ as incorrect
while some $U \in \mathcal{D}$ is incorrect do
  if $\text{MakeTree}(U)$ = failwith $(A, U \setminus A)$ then
    $(U', U \setminus U') := \text{MaxFlow}(A, U \setminus A)$
    $\mathcal{D} := (\mathcal{D} \setminus \{U\}) \cup \{U', U \setminus U'\}$
    Mark $U'$ and $U \setminus U'$ as incorrect
  else Mark $U$ as correct; let $U_T$ be the subtree returned
    by $\text{MakeTree}(U)$
return $\{U_T : U \in \mathcal{D}\}$
```

3.1 The merge phase

At all times, we will be working with a current partition $\mathcal{D}$, which will be updated as the algorithm proceeds. At the start, $\mathcal{D} = \{\{v\} : v \in S\}$.

Given the current partition $\mathcal{D}$, we will repeatedly *merge* a part U of S which is too heavy relative to its capacity to the rest of S (where heavy is relative to $w_{\mathcal{D}}$).

In particular, we proceed as follows:

1. Find UNIFORMFLOWVALUE$(S, \mathcal{D})$. If this value is at least $\delta(S)/\log n$, then by definition we can route the flow UNIFORMFLOW$(S, \mathcal{D})$ with congestion at most $\frac{\log n}{\delta(S)}$, and we have achieved the requirement of the merge phase. Now we can recurse on the subgraphs in $\mathcal{D}$.

2. Otherwise, we find a cut which is quite sparse. Let $(U, S \setminus U) =$ UNIFORMCUT$(S, \mathcal{D})$. Since it is a $\gamma \log n$-approximate sparsest cut, and

$$\text{UNIFORMFLOWVALUE}(S, \mathcal{D}) < \delta(S)/\log n,$$

we know that

$$\frac{\text{cap}(U, S \setminus U)}{w_{\mathcal{D}}(U)} \leq \gamma\delta(S) \qquad (2)$$

by Lemma 9.

3. Now test whether

$$w_{\mathcal{D}}(U) - w_S(U) \geq \frac{1}{2\gamma\delta(S)}\text{cap}(U, S \setminus U). \qquad (3)$$

This inequality is true when there is too much bandwidth across $\mathcal{D}$ inside S for the cut $(U, S\setminus U)$ to support it. We now have two cases:

   (a) If condition (3) holds, then we know we can make progress (as explained in Lemma 1), and so add U to $\mathcal{D}$, remove vertices in U from other sets of $\mathcal{D}$,[4] and repeat from step 1.

   (b) Otherwise, we have a *bad child event*, and we return $(U, S \setminus U)$ to the parent of S for processing. Lemma 2 shows that the cut is within a factor of two of meeting equation 2 for the parent of S.

We next prove that the process halts, *i.e.*, we get a good decomposition. This lets us route UNIFORMFLOW$(S, \mathcal{D})$ with low congestion, or divide S in its parent's decomposition into two bandwidth sets.

LEMMA 1. *The process above halts in polynomial time.*

PROOF. The process will halt if either step 1 succeeds or we get to step 3(b). Hence assume that 3(b) never happens. We will show that step 1 succeeds after a polynomial number of steps.

Consider the potential function $w_{\mathcal{D}}(S) - w_S(S)$. Each time a non-halting (case 3(a)) cut is found, the potential function increases by at most twice the size of the cut, or $2\text{cap}(U, S \setminus U)$, and it decreases by at least $w_{\mathcal{D}}(U) - w_S(U)$. Since $\frac{1}{2\gamma\delta(S)} \geq 2\log\log n > 2$ for $n \geq 4$, we know by equation 3 that the increase is less than the decrease. (The first inequality holds because $\delta(S) \geq 1$ for all S.)

This means $w_{\mathcal{D}}(S) - w_S(S)$ gets smaller with every merge step, and since it starts with a polynomial size and cannot be negative, the algorithm stops after a polynomial number of steps. □

Now we just need to prove the following Lemma about step 3(b).

[4]This may create empty sets in $\mathcal{D}$, so remove them.

LEMMA 2. *If there is a bad child event with cut* $(U, S\setminus U)$, *then* $\frac{\text{cap}(U,S\setminus U)}{w_S(U)} \leq 2\gamma\delta(S)$.

PROOF. Let $(U, S \setminus U)$ be the cut found in step 1 which causes the bad child event. We know that $\frac{\text{cap}(U,S\setminus U)}{w_{\mathcal{D}}(U)} \leq \gamma\delta(S)$ because (2) is true; otherwise, we cannot have a bad child event. We also know that

$$w_{\mathcal{D}}(U) - w_S(U) < \frac{1}{2\gamma\delta(S)}\text{cap}(U, S \setminus U),$$

or else we would be in step 3(a). These two conditions imply that $\frac{\text{cap}(U,S\setminus U)}{w_S(U)} \leq 2\gamma\delta(S)$. □

3.2 Creating flow for the bandwidth sets

Suppose that the merge phase succeeds for S, *i.e.*, we can route UNIFORMFLOW$(S, \mathcal{D})$ with low congestion. However, $\mathcal{D}$ could be further refined by bad child events into a new partition $\mathcal{C}$. This subsection shows how to route UNIFORMFLOW$(S, \mathcal{C})$, while only increasing the congestion by a constant factor.

Consider some parent whose child disappoints it by causing a bad child event. Let S_c be the child node, S_p be the parent node, and $\mathcal{C}$ be the partition of S_p just before the bad child event. Note that $\mathcal{C}$ is not necessarily the partition at the end of the merge phase. In this case, the parent receives a cut of S_c into U and $S_c \setminus U$, and needs to update $\mathcal{C}$ accordingly.

Suppose that we can already route UNIFORMFLOW$(S_p, \mathcal{C})$ with low congestion. This is equivalent to saying that we can route between all of the units of bandwidth in S that contribute to $w_{\mathcal{C}}(S)$. Now we need to cut one of the sets $S_c \in \mathcal{C}$ into U and $S_c \setminus U$, but this creates more units of bandwidth that need to be routed. In order to handle these new units of bandwidth, we first route them to the old units of bandwidth, then use the flow on the old units of bandwidth to route to the destination. We will argue that:

- This does not congest any edge too much.

- No old unit of bandwidth receives too many units of new bandwidth.

Without loss of generality, assume that $|U| \leq |S_c \setminus U|$. We would like to establish a flow from the $2 \cdot \text{cap}(U, S_c \setminus U)$ units of bandwidth created across the cut to the units of border bandwidth of S_c in U (*i.e.*, $w_{S_c}(U)$).

Though we may not be able to route such a flow, we are able to find a $U' \subseteq U$ such that the cut $(U', S\setminus U')$ is at least as good and from which we *are* able to route the flow. This set U' is computed by the procedure MAXFLOW$(U, S_c \setminus U)$, which is defined below.

In particular, for an arbitrary S and a cut $(U, S \setminus U)$, define $\mathcal{N}(U, S \setminus U)$, where $|U| \leq |S \setminus U|$ as the following single-commodity maximum flow network, augmenting the subgraph U as follows:

- Add special source and sink vertices s and t, respectively.

- Add an edge from s to each vertex $u \in U$ with capacity $\text{cap}(u, S \setminus U)$.

- Add an edge from each vertex $v \in U$ to t with capacity $w_S(v)\lambda$, where $\lambda = \frac{\text{cap}(U,S\setminus U)}{w_S(U)}$.

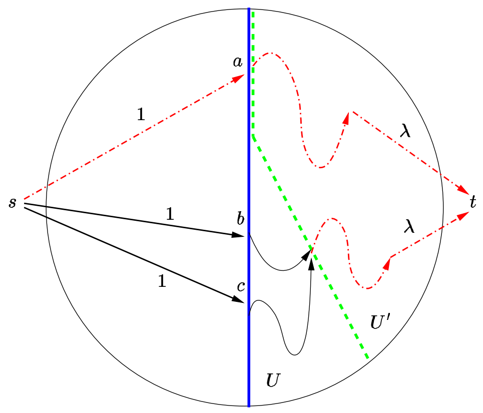

Figure 3: The max-flow adjustment. U' is to the right of the green, dotted line, and U is the right-hand semicircle. There are three edges across the cut $(U, S \setminus U$, with right-endpoints a, b and c, each with one unit of bandwidth. The red, dash-dot edges are saturated, and the black edges are not. Other edges are not shown and have no flow.

DEFINITION 1. *(Max-flow condition) Say that the cut $(U, S \setminus U)$ satisfies the max-flow condition if the minimum cut of $\mathcal{N}(U, S \setminus U)$ has capacity $\lambda w_S(v)$.*

Note that if the max-flow condition is satisfied, then with congestion 1 we can route flow from the cut units of bandwidth to the border bandwidth of U.

LEMMA 3. *Let S be a subgraph and $(U, S \setminus U)$ be a cut. Then there exists some cut $(U', S \setminus U')$ such that:*

- $\frac{\mathrm{cap}(U', S\setminus U')}{w_S(U')} \le \frac{\mathrm{cap}(U, S\setminus U)}{w_S(U)}$;
- $U' \subseteq U$.
- *and the max-flow condition holds for $(U', S \setminus U')$;*

PROOF. We argue that if U does not meet the max-flow condition, we can find a smaller set $U' \subsetneq U$ such that $\frac{\mathrm{cap}(U', S\setminus U')}{w_S(U')} \le \frac{\mathrm{cap}(U, S\setminus U)}{w_S(U)}$. If this set also does not meet the max-flow condition, we will repeat the process until we do find a set U' such that $\mathcal{N}(U', S \setminus U')$ is satisfies the max-flow condition.

Consider the maximum flow from s to t in $\mathcal{N}(U, S \setminus U)$. There is a corresponding minimum cut of this network which has capacity equal to the maximum flow.

If the minimum cut puts s and the vertices of U on one side, and t on the other, then $U' = U$ satisfies the conditions of the lemma, because the capacity of this cut is exactly $\lambda w_S(U)$.

Now suppose not. Then let U' be the set of vertices on the t side of the minimum cut. Note that U' is strictly smaller than U, since the cut separating U and s has capacity $\lambda w_S(U)$. The capacity of this cut in the flow network is $\mathrm{cap}(U \setminus U', U') + \mathrm{cap}(U', s) + \mathrm{cap}(U \setminus U', t) = \mathrm{cap}(U', S \setminus U') + w_S(U \setminus U')\lambda$ (the first two terms sum to $\mathrm{cap}(U', S \setminus U')$). It must be that

$$\begin{aligned} &\mathrm{cap}(U', S \setminus U') + w_S(U \setminus U')\lambda < w_S(U)\lambda \\ \Rightarrow\quad &\frac{\mathrm{cap}(U', S \setminus U')}{w_S(U')} < \lambda \end{aligned}$$

But recall that $\lambda = \frac{\mathrm{cap}(U, S\setminus U)}{w_S(U)}$; hence $\frac{\mathrm{cap}(U', S\setminus U')}{w_S(U')}$ is at most $\frac{\mathrm{cap}(U, S\setminus U)}{w_S(U)}$. So U' meets the first two conditions of the Lemma. Note that if max-flow condition does not hold for $(U', S \setminus U')$ we can repeat the argument above to find a set $U'' \subsetneq U'$ meeting the first two conditions.

Since the set in question gets smaller at each repetition, we either find a set meeting the max-flow condition, or we get to a set of one vertex which trivially meets the max-flow condition. Since each set has at least one fewer vertex than the previous one, we terminate after no more than $|U|$ iterations. $\square$

Note that the flow only uses edges in U', and creates congestion at most 1 on them. This flow routes the units of bandwidth created by the cut the to units of bandwidth that already existed, just as we wanted.

One technical point is that we actually create not one unit of bandwidth with each cut, but two: one on each endpoint of an edge in the cut. We will get around this by first routing directly across the cut, then using the max flow to route to the final destination. For clarity, we ignore this factor of two in the next subsection.

3.3 Bandwidth routing analysis

In this subsection, we consider the congestion created by routing through all of the bandwidth sets described above.

Let U be a set in the final *merge* phase partitioning $\mathcal{D}$ of S; consider the final *bandwidth* phase decomposition $\mathcal{C}$ of U (this is after the last cut has been sent from the children of S). We route one unit of flow from each unit of border bandwidth of sets in $\mathcal{C}$ to the border bandwidth of U such that each unit of border bandwidth of U receives at most a constant amount of flow as described above. Now we show that the total additional congestion in this process is small.

The partition $\mathcal{C}$ of U is produced by a series of bandwidth cuts. These cuts can be visualized as a binary tree, where subpieces are children of their containing pieces. Each such tree of cuts is rooted at a particular set in $\mathcal{D}$. Note that this tree is not the same as the decomposition tree of the graph.

We will route the flow by routing up the tree, one cut at a time. Each cut will be routed according to the flow described and proven to exist in subsection 3.2. We now bound the congestion incurred in routing this flow.

LEMMA 4. *Let U be a set in the final merge phase partition of S, and let $\mathcal{C}$ be the final bandwidth phase partition of U. The units of bandwidth contributing to $w_{\mathcal{C}}(U)$ can be routed inside U to the units of border bandwidth of U such that each unit receives at most 2 units of flow, and the maximum congestion on any edge in U is at most $2 \log n$.*

PROOF. The core idea of the proof is to notice that the capacity of bandwidth cuts is small relative to the weight on the smaller side. In particular, we know that cuts returned by bad child events are small by Lemma 2. Further, we know by Lemma 3 that the final bandwidth cuts made are also small.

The bandwidth sets formed by cutting a merge set U form a binary tree. Each node in this tree is a subset of U. We will use the convention that the left child of a node is the smaller side of the cut. Now define the *left depth* of a unit of bandwidth to be the number of left branches on the path from the root (the merge set) to the tree node in which the unit first appears as border bandwidth, plus one. For

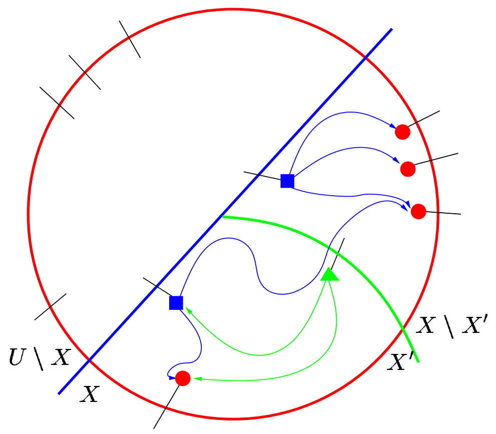

Figure 4: A two-level example of bandwidth flows. The large red circle is the boundary of U, a merge set, and the small red circles are the units of border bandwidth of U. The blue cut is a top-level bandwidth cut, and the blue squares are the units of border bandwidth of X. The green cut is a second-level bandwidth cut, and the green triangle is a unit of border bandwidth of X'. The green and blue paths are the routes used in the max flow to send flow up the bandwidth tree.

example, the left depth of the merge-level units of bandwidth in U is one.

Let $\Phi(j)$ denote the maximum *overload factor* of a unit of bandwidth at left depth at most j in the tree. This is the total amount of flow that is routed to that unit of bandwidth before being sent further up the tree.

We will now prove by (backward) induction that

$$\Phi(j) \leq \prod_{l=j}^{\log n} \left(1 + \frac{1}{l \log\log n}\right). \qquad (4)$$

For the base case, note that since the left depth can be no more than $(-1+\log n)+1$, $\Phi(\log n) = 1$ (the -1 is because $|U| \leq n/2$). Now assume equation 4 is true for all $l \geq j+1$. We will prove it for j.

Consider a unit of bandwidth at left-depth j in the tree. It can only have flow routed to it when it first appears further down in the tree on the left side of a cut. Let $(X', X \setminus X')$ be this cut of a node X. We know that the left-depth of the units of bandwidth in this cut are at least j.

Assume without loss of generality that X' is the left node, *i.e.*, $|X'| \leq |X \setminus X'|$. Now the units of bandwidth in X' across the cut are at left-depth $j+1$, and our unit of bandwidth is in X'. Then the following inequality holds:

$$\Phi(j) \leq \Phi(j+1) + 2\gamma\delta(X)\Phi(j) + 2\gamma\delta(X')\Phi(j+1). \qquad (5)$$

The first term comes from the fact that a unit of weight at left depth at most j is also a unit of weight at left depth at most $j+1$. Hence before flow has been routed at this level, the overload factor of this unit of bandwidth is at most $\Phi(j+1)$.

The second two terms account for the units of bandwidth created when X is split into X' and $X \setminus X'$. By Lemma 2 the capacity of the cut is at most $2\gamma\delta(X)w_X(X')$, and Lemma 3 preserves this property, so there are at most $2\gamma\delta(X)w_X(X')$ units of weight created in X', and each such unit of weight is at left depth at least $\Phi(j+1)$. There are also at most $2\gamma j(X)w_X(X')$ units of bandwidth created in $X \setminus X'$, and each such unit has left depth at least $\Phi(j)$. Since all of this is true for any unit of bandwidth at left depth j, equation 5 holds.

Noting that $\Phi(j+1) \leq \Phi(j)$ and solving for $\Phi(j)$, equation 5) becomes

$$\Phi(j) \leq \Phi(j+1)\frac{1}{1-4\gamma\delta(X)} \leq \Phi(j+1)(1+8\gamma\delta(X)),$$

as long as $4\gamma\delta(X) < 1/2$.

Since the left child has size at most half its parent, and it takes $j-1$ left child steps to get to X, it must be that $|X| \leq |U|/2^{j-1}$. Since $|U| \leq |S|/2$, $|X| \leq |S|/2^j$, and hence $\log(|S|/|X|) \geq j$. This makes $\delta(X) \leq \frac{1}{8\gamma j \log\log n}$.

Hence

$$\begin{aligned}
\Phi(j) &\leq \Phi(j+1)\left(1 + \frac{8\gamma}{8\gamma j \log\log n}\right) \\
&\leq \left(\prod_{l=j+1}^{\log n}\left(1+\frac{1}{l\log\log n}\right)\right)\left(1+\frac{1}{j\log\log n}\right) \\
&= \prod_{l=j}^{\log n}\left(1+\frac{1}{l\log\log n}\right)
\end{aligned}$$

The second inequality is the inductive hypothesis. This concludes the sub-Lemma.

By the sub-Lemma, we know know that

$$\Phi(1) \leq \prod_{l=1}^{\log n}\left(1+\frac{1}{l\log\log n}\right).$$

By Lemma 10, the product on the right-hand side is at most 2. Hence any unit of bandwidth receives at most 2 units of flow.

Now, by Lemma 3 each maximum flow can be routed with no congestion if only one unit of flow is routed from each unit of new bandwidth to each unit of old bandwidth. Recall that we are sending at most two units of flow. Hence the congestion on an edge involved in any particular max flow is at most two.

An edge is only congested by max flows when it is on the small side of the cut (the left child side), or is contained in the cut. In the first case, since both endpoints of the edge must be in the small side, the edge can only be congested at most $-1+\log n$ times (the -1 is since $|U| \leq n/2$). The second case only happens at most once. Hence the maximum congestion over any edge is at most $2\log n$. □

3.4 Routing Uniform Flow

We now show how to use the max flows described in section 3.2 to route among the units of bandwidth of the bandwidth sets. Let $\mathcal{D}_m$ be the merge sets of S, and let $\mathcal{D}_b$ the refinement into bandwidth sets. By construction, we know that we can route UNIFORMFLOW$(S, \mathcal{D}_m)$ with low congestion. Our aim is to route UNIFORMFLOW$(S, \mathcal{D}_b)$.

Here is the way in which we will route flow:

1. Using the maximum flows, send the units of bandwidth of $\mathcal{D}_b$ to their corresponding merge set (the root of their trees). By Lemma 4, (along with the technicality described at the end of Section 3.2), this puts at most

a factor four more weight on the units of bandwidth of $\mathcal{D}_m$. That is, each unit of bandwidth of $\mathcal{D}_m$ is now responsible for as many as 4 units of bandwidth.

2. Use UNIFORMFLOW$(S, \mathcal{D}_m)$ to route the flow uniformly between the merge sets. The demand between u and v has increased by a factor of at most 4, so the congestion increases by a factor of at most 4. Since we can already route UNIFORMFLOW$(S, \mathcal{D}_m)$, this causes no problems.

3. Use the maximum flows to send flow from the units of $\mathcal{D}_m$ to their corresponding bandwidth children. Note that this is step one backward, and therefore doubles the amount of congestion from step 1.

3.5 Total congestion of an edge over all flows

Now we turn our attention to the congestion experienced by an edge (u, v) in the graph. The edge can be congested in two ways: by max flows, and by the multicommodity flows found at the end of the merge phase (recall that the multicommodity flow for each node in the decomposition tree is the combination of these two flows).

Max-flow congestion The edge can be congested by a max flow each time both u and v appear in a bandwidth routing tree, in the sense of Lemma 4. By the lemma, each time the edge adds congestion of at most $2 \cdot 2 = 4$, where the extra factor of two comes from fact we must support both vertices of the newly cut edges. (See the last paragraph of section 3.2.)

Each time u and v appear in the smaller half corresponds to a given subgraph S's partition in the tree. A given max flow is used exactly once to be able to route each flow UNIFORMFLOW$(S, \mathcal{D})$, as described in subsection 3.4. Each of these flows is used in multicommodity flows UNIFORMFLOW$(S, \mathcal{D})$ for at most two nodes of the decomposition tree (S, and S's parent), by Lemma 8. Hence each max flow in which (u, v) participates is used at most twice.

The edge participates in at most $\log n$ flows, since the sets for those flows all lie on a single root-leaf path in the decomposition tree. Hence (u, v) incurs at most $2 \log n \cdot 4 = 8 \log n$ congestion over all of the flows in the decomposition tree.

Multicommodity flow congestion The edge can be congested, for various sets S with *merge*-level partitions $\mathcal{D}_m$, by the flow UNIFORMFLOW$(S, \mathcal{D}_m)$. As in the max flow case, by Lemma 8, each of these flows is used twice to route: once for S's parent, and once for S.

The total congestion due to the multicommodity flows is at most

$$\sum_{\text{tree nodes } S \ni \{u,v\}} \frac{\log n}{\delta(S)},$$

by condition 2. All of these sets S lie on a single path in the decomposition tree which starts at the root and stops at the decomposition which separates u from v. We can therefore index the set which is j steps from the root in this path as S_j; let m be the number of sets in the path. In this notation, the above sum can be written as

$$\begin{aligned} \sum_{k=1}^{m-1} \frac{\log n}{\delta(S_k)} &= (\log n) \sum_{k=1}^{m-1} 8\gamma \log \frac{|S_{i-1}|}{|S_i|} \log \log n \\ &= 8\gamma \log n \log \log n \sum_{k=1}^{m-1} \log \frac{|S_{i-1}|}{|S_i|} \\ &\leq 8\gamma \log^2 n \log \log n \qquad (6) \end{aligned}$$

The inequality (6) is proven as follows: consider the product $\prod_{k=1}^{m-1} \frac{|S_{i-1}|}{|S_i|}$. The product telescopes, and is hence equal to

$$|S_0|/|S_{m-1}| \leq n.$$

Taking logs of both sides yields the inequality.

Putting it all together, we see that any edge incurs at most $8 \log n + 8\gamma \log^2 n \log \log n$ total congestion over the entire tree. This is summarized in the Lemma below.

LEMMA 5. *The congestion of any edge over all of the flows in the entire tree is at most* $8 \log n + 8\gamma \log^2 n \log \log n = O(\log^2 n \log \log n)$.

3.6 Running Time

LEMMA 6. *The algorithm runs in polynomial time.*

PROOF. The depth of the decomposition tree is at most $\log n$ since each merge set is at most half the size of its parent (see Lemma 9), and any bandwidth sets produced are smaller than the merge set that spawned them.

The running time of the merge procedure without any bad child events is polynomial, since we have assumed that the total bandwidth in the graph is polynomially bounded. Processing a bad child event also takes polynomial time. Hence we next need to bound the number of bad child events. The number of bad child events is $O(n \log n)$ since each produces a new bandwidth set. The total number of bandwidth sets in the entire decomposition is $O(n \log n)$ since any level of the decomposition is a partition of the nodes and there at most $O(\log n)$ levels of the decomposition. □

The running time could be more shrewdly bounded, but this is not in the scope of this paper.

4. COMPETITIVE ROUTING

Now we turn to using the tree T computed in the previous section to route in the graph. In order to route from a to b, we will route through a sequence of multicommodity flows. There will be one flow for each edge in the path in T from the leaf $\{a\}$ to the leaf $\{b\}$.

We will first define a particular flow with respect to each tree node S, and then show how to use those flows to give flows for the path from a to b. Let the decomposition of S into children be $\mathcal{D}_1$, and its decomposition into grandchildren be $\mathcal{D}_2$ (note that $\mathcal{D}_2$ is in some sense a refinement of $\mathcal{D}_1$). Define the demand from $u \in S$ to $v \in S$ be

$$w_{X_u}(X_u) \frac{w_{\mathcal{D}_2}(u)}{w_{\mathcal{D}_2}(X_u)} \frac{w_{\mathcal{D}_1}(v)}{w_{\mathcal{D}_1}(S)}, \qquad (7)$$

where $X_u \in \mathcal{D}_1$ is the subpiece containing u.

Here are three observations about the flow we need to route:

- Suppose that, for some $X_u \in \mathcal{D}_1$, $a \in X_u$ and $b \notin X_u$, so that any flow from a to b in the tree needs to route across the tree edge from X_u to S. Further suppose that (using induction) we have already spread our flow from a to b evenly across the bandwidth of the vertices of X_u with respect to $\mathcal{D}_2$, that is, with a $\frac{w_{\mathcal{D}_2}(v)}{w_{\mathcal{D}_2}(X_u)}$ fraction of the flow at each vertex $v \in X_u$. We would like to send the flow out of X_u and spread it over all of the vertices of S with respect to $\mathcal{D}_1$ (each vertex $v \in S$ receives a $\frac{w_{\mathcal{D}_1}(v)}{w_{\mathcal{D}_1}(S)}$ fraction). This will let us maintain our inductive condition, and it will also give us a way to route out of X_u.

- *Any* flow out of X_u is limited by $w_{X_u}(X_u)$; if we would like to send $c \cdot w_{X_u}(X_u)$ units of flow out of X_u, then by averaging there must be some edge between X_u and the rest of the graph that has congestion at least c. On the other hand, if we can find a way to route $w_{X_u}(X_u)$ units of flow out of X_u with low congestion L, we can also route $cw_{X_u}(X_u)$ units of flow with congestion cL by simply multiplying the flow on each edge by a factor of c. Putting these facts together, we see that the ratio of our congestion to the optimal congestion will be at most L, regardless of the amount of flow that needs to be routed. For this reason, we will concentrate on trying to route only $w_{X_u}(X_u)$ units of flow.

- Since it is only harder to route more flow, in the worst case we will need to simultaneously route $w_X(X)$ units of flow out of all of the sets $X \in \mathcal{D}_1$.

These observations put together motivate the definition of the demands in (7) (reading from left to right in the equation): they say we need to route $w_{X_u}(X_u)$ units of flow, originally distributed according to the $\mathcal{D}_2$ distribution in X_2, to the $\mathcal{D}_1$ distribution over all of S, and that we need to do this simultaneously for all sets X_u.

4.1 Congestion analysis

This subsection will make the previous subsection's argument formal. We will be working throughout with a fixed but arbitrary set of demands that need to be routed on the graph. Our goal is to find out how much worse our congestion is than the best possible.

DEFINITION 2. *For any $S \subseteq V$, let dem(S) be the demand originating inside S which needs to leave it. Let $\beta(S) = \frac{\mathsf{dem}(S)}{\mathsf{cap}(S, V\setminus S)}$.*

LEMMA 7. *(Lower bound on OPT) Any routing of these demands must incur congestion at least $\beta = \max_S \beta(S)$ on some edge, where the maximum is taken over all sets S in the decomposition tree.*

Consider a set S in the decomposition tree. Let $\mathcal{D}_1$ be the partition of S into children, and $\mathcal{D}_2$ be the partition into grandchildren.

Suppose that, for any $X \in \mathcal{D}_1$, the demand dem(X) is spread uniformly over the border bandwidth of the sets of $\mathcal{D}_2$ inside U (*i.e.*, each vertex $v \in X$ has $\frac{w_{\mathcal{D}_2}(v)}{w_{\mathcal{D}_2}(X)}\mathrm{dem}(X) = w_{\mathcal{D}_2}(v)\beta(X)$ units of the demand).

Then we want to route that demand so that it is uniformly spread over the border bandwidth of the sets in $\mathcal{D}_1$ (*i.e.*, each vertex $v \in S$ gets $\frac{w_{\mathcal{D}_1}(v)}{w_{\mathcal{D}_1}(S)} \sum_{X \in \mathcal{D}_1} \mathrm{dem}(X)$ units of demand). Furthermore, this routing will be accomplished using UNIFORMFLOW$(S, \mathcal{D}_1)$, and UNIFORMFLOW$(X, \mathcal{D}_2)$, for each X, with only one use of each flow, and only overloading each of them by a multiplicative factor of β.

LEMMA 8. *We can use the flows* UNIFORMFLOW$(S, \mathcal{D}_1)$, *and* UNIFORMFLOW$(X, \mathcal{D}_2)$, *for each $X \in \mathcal{D}_1$, to route the demands from equation 7 with congestion at most β. Further, each of the above flows is used once.*

PROOF. Notice that for any $X \in \mathcal{D}_1$, we know from the assumptions of the Lemma that any vertex $v \in X$ starts out with $w_{\mathcal{D}_2}(v)\beta(U)$ units of demand. We will assume for simplicity that they all have $w_{\mathcal{D}_2}(v)\beta$ units of demand, since this is only larger.

We will now show how to route assuming that each vertex has only $w_{\mathcal{D}_2}(v)$, and then multiply by β at the end to give the result of the Lemma.

Recall that the $\mathcal{D}_2$ distribution in X is the final result after the merge and bandwidth decomposition phases of the subgraph X, and that the $\mathcal{D}_1$ distribution in S is the final result after the phases on S.

Our routing scheme is as follows:

1. For each X, inside X, route from the border bandwidth with respect to $w_{\mathcal{D}_2}$ to the border bandwidth with respect to $w_{\mathcal{D}_1}$. In equations, the demand from u to v will be
$$\begin{aligned} w_{\mathcal{D}_1}(X)\frac{w_{\mathcal{D}_2}(u)}{w_{\mathcal{D}_2}(X)}\frac{w_{\mathcal{D}_1}(v)}{w_{\mathcal{D}_1}(X)} &= \frac{w_{\mathcal{D}_2}(u)w_{\mathcal{D}_1}(v)}{w_{\mathcal{D}_2}(X)} \\ &\leq \frac{w_{\mathcal{D}_2}(u)w_{\mathcal{D}_2}(v)}{w_{\mathcal{D}_2}(X)} \quad (8) \end{aligned}$$
To do this, notice that inequality (8) shows that the flow UNIFORMFLOW$(X, \mathcal{D}_2)$ is sufficient to route this flow.

2. Route between the border bandwidth with respect to $w_{\mathcal{D}_1}$ on all of S, *i.e.*the flow demands UNIFORMFLOW$(S, \mathcal{D}_1)$.

Note that each flow is used only once, as required. □

THEOREM 1. *The competitive ratio of our algorithm is* $O(\log^2 n \log\log n)$.

PROOF. This is a consequence of Lemmas 5, 7, 8. □

5. MISCELLANEOUS LEMMAS

LEMMA 9. *Fix a subgraph S and a decomposition $\mathcal{D}$ of S. Suppose that* UNIFORMFLOWVALUE$(S, \mathcal{D}) < q$, *for some q. Let $(U, S \setminus U)$ =* UNIFORMCUT$(S, \mathcal{D})$. *Assume that $|U| \leq |S \setminus U|$. Then $\frac{\mathsf{cap}(U, S\setminus U)}{w_{\mathcal{D}}(U)} < q\gamma \log n$.*

PROOF. Since the cut $(U, S \setminus U)$ is a $\gamma \log n$-approximate sparsest cut, it must be that
$$\frac{\mathsf{cap}(U, S \setminus U)}{\frac{w_{\mathcal{D}}(U)w_{\mathcal{D}}(S\setminus U)}{w_{\mathcal{D}}(S)}} < q\gamma \log n$$
This implies that $\frac{\mathsf{cap}(U, S\setminus U)}{w_{\mathcal{D}}(U)} < q\gamma \log n$ as desired, since since $w_{\mathcal{D}}(S \setminus U) \leq w_{\mathcal{D}}(S)$. □

LEMMA 10. *If $c \leq 1/(\log k)$, then $\prod_{i=1}^{k}(1 + c/i) \leq 2$.*

PROOF. Let $y = \prod_{i=1}^{k}(1 + c/i)$. Take natural logs of both sides of the equations to get that $\ln y = \sum_{i=1}^{k} \ln(1 + c/i)$. Using the fact that $\ln(1 + x) \leq x$, we can rewrite the equations as $\ln y \leq \sum_i (c/i) \leq c \ln k$. But $\ln y \leq c \ln k \leq \frac{\ln k}{\log k} = \ln 2$, so $y \leq e^{\ln 2} = 2$. □

6. REFERENCES

[1] B. Aiello, T. Leighton, B. Maggs, and M. Newman. Fast algorithms for bit-serial routing on a hypercube. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 55–64, 1990.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, May 1997.

[3] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, Feb. 1998.

[4] B. Awerbuch and Y. Azar. Local optimization of global objectives: competitive distributed deadlock resolution and resource allocation. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 240–249, 1994.

[5] B. Awerbuch and T. Leighton. 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*, pages 487–496, 1994.

[6] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke. Optimal oblivious routing in polynomial time. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, June 2003.

[7] M. Bern, J. R. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo. Support graph preconditioners. Unpublished manuscript. Available at `ftp://ftp.cs.sandia.gov/pub/papers/bahendr/support-graph.ps.gz`.

[8] M. Bienkowski, M. Korzeniowski, and H. Räcke. A practical algorithm for constructing oblivious routing schemes. In *Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures*, June 2003.

[9] E. Boman and B. Hendrickson. Support theory for preconditioning. Unpublished manuscript. Available at `ftp://ftp.cs.sandia.gov/pub/papers/bahendr/support-theory.ps.gz`.

[10] A. Borodin and J. Hopcroft. Routing, merging and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 10(1):130–145, 1985.

[11] A. Borodin, P. Raghavan, B. Schieber, and E. Upfal. How much can hardware help routing? *Journal of the ACM*, 44(5):726–741, Sept. 1997.

[12] J. Fakcharoenphol and K. Talwar. Improved decompositions for graphs with excluded minors. Unpublished manuscript.

[13] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. In *Proceedings of the 26th Annual Symposium on the Foundations of Computer Science*, pages 241–249, 1985.

[14] K. Gremban. *Combinatorial Peconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems.* PhD thesis, Carnegie Mellon University, Oct. 1996.

[15] K. D. Gremban, G. L. Miller, and M. Zagha. Performance Evaluation of a New Parallel Preconditioner. Technical Report CMU-CS-94-205, Carnegie Mellon University, 1994.

[16] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Fast algorithms for bit-serial routing on a hypercube. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 31–36, 1990.

[17] P. Klein, S. A. Plotkin, and S. Rao. Excluded minors, network decomposition, and multicommodity flow. In ACM, editor, *Proceedings of the twenty-fifth annual ACM Symposium on the Theory of Computing, San Diego, California, May 16–18, 1993*, pages 682–690, New York, NY, USA, 1993. ACM Press.

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

[19] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, Oct. 1985.

[20] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. In *IEEE Symposium on Foundations of Computer Science*, pages 577–591, 1994.

[21] B. M. Maggs, G. L.Miller, O. Parekh, R. Ravi, and S. L. M. Wu. Solving symmetric diagonally-dominant systems by preconditioning. Unpublished manuscript, 2002.

[22] B. M. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Proceedings. 38th Annual Symposium on Foundations of Computer Science*, pages 284–293, 1997.

[23] H. Räcke. Minimizing congestion in general networks. In *Proceedings of the 43rd Annual Symposium on the Foundations of Comuter Science*, pages 43–52, Nov. 2002.

[24] P. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript, 1991.

[25] L. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th ACM Symposium on Theory of Computing*, pages 263–277, 1981.

[26] B. Vöcking. *Static and Dynamic Data Management in Networks.* PhD thesis, Universität Paderborn, 1998.

[27] B. Vöcking. Almost optimal permutation routing on hypercubes. In ACM, editor, *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing: Hersonissos, Crete, Greece, July 6–8, 2001*, pages 530–539, New York, NY, USA, 2001. ACM Press.

[28] T. Yeh and C. Lei. Competitive analysis of minimal oblivious routing algorithms on hypercubes. *IEICE Transactions on Information and Systems*, E84-D(1):65–75, Jan. 2001.