A Practical Algorithm for Constructing Oblivious Routing Schemes*

Marcin Bienkowski† Miroslaw Korzeniowski† Harald Räcke‡

Abstract

In a (randomized) oblivious routing scheme the path chosen for a request between a source s and a target t is independent from the current traffic in the network. Hence, such a scheme consists of probability distributions over $s-t$ paths for every source-target pair s,t in the network.

In a recent result [11] it was shown that for any undirected network there is an oblivious routing scheme that achieves a polylogarithmic competitive ratio with respect to congestion. Subsequently, Azar et al. [4] gave a polynomial time algorithm that for a given network constructs the best oblivious routing scheme, i.e. the scheme that guarantees the best possible competitive ratio. Unfortunately, the latter result is based on the Ellipsoid algorithm; hence it is unpractical for large networks.

In this paper we present a combinatorial algorithm for constructing an oblivious routing scheme that guarantees a competitive ratio of $\mathcal{O}(\log^4 n)$ for undirected networks. Furthermore, our approach yields a proof for the existence of an oblivious routing scheme with competitive ratio $\mathcal{O}(\log^3 n)$, which is much simpler than the original proof from [11].

1 Introduction

Efficient routing protocols for unstructured network topologies have become more and more important in recent years, because of the dramatic growth of the Internet, and the increasing popularity of e.g. ad-hoc networks and networks of workstations. A routing algorithm for such networks should be simple in order to enable quick routing decisions; it should be distributed in order to work efficiently in large networks, and it should be online in order to deal with various traffic patterns.

In this paper we focus on the problem of online virtual circuit routing in which routing requests, consisting of a source and a target node, arrive online and a routing algorithm has to select a path in the network that connects the source and the target node for each request. The goal is to minimize the congestion, i.e. the maximum load of a network link, where the load of a link is the amount of data transmitted by the link divided by the link-capacity.

One approach towards online routing in networks is to route obliviously, i.e. without any knowledge of the current state of the network. For an oblivious algorithm the path chosen for

*This work was partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and by DFG-Sonderforschungsbereich 376 “Massive Parallelität: Algorithmen Entwurfsmethoden Anwendungen”

†International Graduate School of Dynamic Intelligent Systems, Computer Science Department, University of Paderborn, D-33102 Paderborn, Germany. Email: {young,rudy}@upb.de

‡Heinz Nixdorf Institute and Computer Science Department, University of Paderborn, D-33102 Paderborn, Germany. Email: harry@upb.de

a request may only depend on the source node, the target node and on some random input if randomization is allowed. Therefore, an oblivious algorithm meets all the criteria described above. It is simple, because routing paths can be realized via a lookup in a routing table; it is distributed, since all routing decisions can be made locally and it is online, since it does not require preprocessing.

In a recent result [11] it was shown that for any undirected network, there exists an oblivious routing algorithm that achieves a competitive ratio of $\mathcal{O}(\log^3 n)$ with respect to congestion, where n denotes the number of nodes in the graph. This result is non-constructive, i.e. the question whether such an oblivious algorithm can be found in polynomial time remained open.

This question was subsequently addressed by Azar et. al [4] who have shown that the optimal oblivious routing scheme, i.e., the scheme that guarantees the best possible competitive ratio, can be constructed in polynomial time.

In this paper we present a constructive version of the results in [11] that guarantees a competitive ratio of $\mathcal{O}(\log^4 n)$. While this algorithm guarantees a weaker bound than the polytime algorithm of [4] it has the following advantages.

The algorithm in [4] is based on linear programming with an infinite number of constraints. Therefore, it uses the Ellipsoid algorithm with a separation oracle to compute the solution. This approach is unpractical for large networks.

A second important difference to the work in [4] lies in the structure of the resulting oblivious routing scheme. Our algorithm follows the approach of [11] and constructs a hierarchical decomposition of the network that then can be used to define the oblivious routing scheme. As shown in [9] and [11] this hierarchical decomposition can be used to solve other important problems in the area of distributed computing, as e.g. multicast routing and data management problems. Furthermore, Maggs et al. [10] have shown most recently that the decomposition can be used as a preconditioner for solving sparse linear systems. Hence, our work does not only give an efficient construction of an oblivious routing scheme but also gives a constructive vesion for all the above problems, that depend on the hierarchical decomposition.

Finally, an important contribution of this work is that it enormously simplifies the proofs in [11].

1.1 Related work

Rhagavan and Thompson [12] have shown that the offline version of the virtual circuit routing problem can be solved via a concurrent multicommodity flow problem. By applying randomized rounding to the solution of the CMCF-problem they get a virtual circuit routing algorithm that well approximates the lowest possible congestion.

In the online setting Aspnes et al. [1] presented an algorithm that achieves a competitive ratio of $\mathcal{O}(\log n)$ w.r.t. congestion. This algorithm is based on the use of an exponential cost function. Each edge e is assigned a length that is exponential in the current load of e. If a routing request occurs the algorithm chooses a shortest path between source and destination with respect to the length assigned to the edges. The competitive ratio of this algorithm is optimal due to a lower bound provided in the same paper. The drawback of this algorithm is that it is centralized and it serializes the routing requests.

Awerbuch and Azar [3] gave a distributed algorithm that repeatedly scans the network so as to choose the routes. Unfortunately, this algorithm requires shared variables on the edges of the network and hence is hard to implement.

All above algorithms are adaptive. In [13] Valiant and Brebner considered oblivious routing on specific network topologies and designed an efficient randomized oblivious routing algorithm for the hypercube. Later, Borodin and Hopcroft [5] and subsequently Kaklamanis et al. [7] have shown that randomization is required for efficient oblivious algorithms, since deterministic algorithms cannot well approximate the minimal possible congestion on any non-trivial network.

Independently from our work Harrelson et al. [6] gave a construction of a hierarchical decomposition that guarantees better bound on the competitive ratio than our result.

2 Preliminaries

We model the network as a complete weighted undirected graph G with node set V. We use n to denote the cardinality of V, i.e. $|V| = n$. Network links are represented via a weight function $\mathrm{c} : V \times V \to \mathbb{R}_0^+$ that for a pair of nodes describes the link-capacity between these nodes. If $\mathrm{c}(u,v) = 0$ for two nodes u and v, then there is no link between these nodes in the physical network. Note that the graph G is undirected which means that we assume $\mathrm{c}(u,v) = \mathrm{c}(v,u)$ for any two nodes $u, v \in V$. Furthermore, we assume that the weight function c is normalized, i.e. the minimum nonzero capacity of a link is 1. We denote the maximum capacity of a network link with $c_{\max}$.

We define a function $\mathrm{cap} : 2^V \times 2^V \to \mathbb{R}_0^+$ which for two subsets $X, Y \subseteq V$ describes the total link-capacity that is available between nodes of X and nodes of Y. It is defined as follows:

$$\mathrm{cap}(X,Y) := \sum_{x \in X, y \in Y} \mathrm{c}(x,y) \ .$$

For a set $X \subseteq V$ we denote the total capacity of edges leaving set X in G with $\mathrm{out}(X) = \mathrm{cap}(X, \overline{X})$, where $\overline{X} := V \setminus X$.

A randomized oblivious routing scheme consists of a probability distribution over s-t paths for each source-target pair s, t. Equivalently, such a probability distribution can be viewed as a unit flow between s and t.

We assume that our oblivious algorithm may route fractionally, i.e. a routing request of demand d between s and t may be fulfilled via a flow of value d between s and t, and is not restricted to use only a single path. The results of Rhagavan and Thompson [12] show that fractional routing and the method of probabilistically choosing a fixed path, where the probability that a path is chosen corresponds to the flow in the fractional routing are nearly equivalent.

Since we use fractional routing we can neglect individual routing requests and only need the total demand between every source-target pair to specify the communication load induced by a routing algorithm. This is done via a *demand matrix* D which is an $n \times n$ nonnegative matrix where the diagonal entries are 0.

For a given routing algorithm and a demand matrix we define the (absolute) load of a link as the total amount of data transmitted by the link. The relative load of a link is defined to be its load divided by its capacity. Finally, we define the *congestion* to be the maximum over the relative loads of all links in the network.

Suppose that $d_{s,t}$ denotes the total demand of all routing requests between s and t. The load induced by an oblivious algorithm on an edge e is given by $\sum_{s,t} d_{s,t} \cdot \mathrm{flow}_{s,t}(e)$, where $\mathrm{flow}_{s,t} : V \times V \to \mathbb{R}_0^+$ denotes the unit flow between s and t, used in the oblivious routing

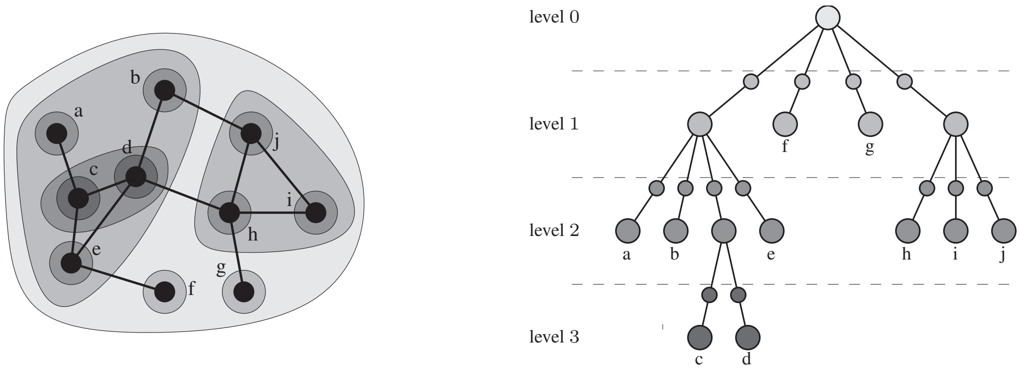

Figure 1: A hierarchical decomposition of a graph and the associated decomposition tree. Small circles in the right figure correspond to blue vertices and large circles correspond to red vertices.

scheme. This fixes the congestion, as well. The optimal congestion that can be achieved for a given demand matrix D can be simply computed via a concurrent multicommodity flow problem.

Let $\mathrm{opt}(D)$ and $\mathrm{obl}(D)$ denote the congestion achieved for demand matrix D, by an optimal algorithm and by a given oblivious routing scheme, respectively. The competitive ratio of an oblivious algorithm is defined as $\sup_D\{\frac{\mathrm{obl}(D)}{\mathrm{opt}(D)}\}$. We will show how to construct an oblivious routing scheme with a competitive ratio of $\mathcal{O}(\log^4 n)$.

2.1 The hierarchical decomposition

Our oblivious routing scheme depends on a hierarchical decomposition of the network which is defined as follows. A *hierarchical decomposition* $\mathcal{H}$ of the graph G is a set system over the universe V that has the following properties

- $\mathcal{H}$ is *laminar*, i.e. for two subsets $X, Y \in \mathcal{H}$ either $X \setminus Y$, $Y \setminus X$ or $X \cap Y$ is empty.
- $\mathcal{H}$ contains V and all sets $\{v\}$, $v \in V$.

Given a hierarchical decomposition $\mathcal{H}$ of G we construct a decomposition tree $T_{\mathcal{H}} = (V_t, E_t)$. The node set $V_t = V_t^B \uplus V_t^R$ of the tree consists of a set V_t^B of *blue nodes* and a set V_t^R of *red nodes* that are defined as follows. For each set $H \neq V$ from the laminar system $\mathcal{H}$ the tree contains two nodes $r_t \in V_t^R$ and $b_t \in V_t^B$. We call H the set or cluster corresponding to r_t and b_t. Further, r_t and b_t are called the *red node* and *blue node*, respectively, corresponding to cluster H. For $H = V$ the tree contains only a red node but no blue one.[1]

In the following the cluster corresponding to a node $v_t \in V_t$ will be denoted with H_{v_t}. A red node r_t and a blue node b_t in $T_{\mathcal{H}}$ are connected if $H_{b_t} \subseteq H_{r_t}$ and if there is no $H \in \mathcal{H}$ such that $H_{b_t} \subsetneq H \subsetneq H_{r_t}$. Note that by this definition $T_{\mathcal{H}}$ is indeed a tree, since $\mathcal{H}$ is a laminar system. We assume $T_{\mathcal{H}}$ to be rooted at the node corresponding to the cluster V, that contains all nodes in the network. By this definition the root and the leaves of $T_{\mathcal{H}}$ are red

[1] Note that this definition of a decomposition tree substantially differs from the definition used in [11], since a cluster of the set system may correspond to several tree nodes and not only to one. The new definition may seem unnatural but it will turn out that it will help us to simplify the proof of the competitive ratio of the oblivious routing scheme.

nodes and the leaves correspond to sets $\{v\}, v \in V$, i.e. there is a one-to-one relation between the nodes of G and the leaf nodes of $T_{\mathcal{H}}$.

We define levels for nodes and edges in $T_{\mathcal{H}}$, as follows. The level of a node v_t of $T_{\mathcal{H}}$ is defined as the number of red nodes on the path from v_t to the root, not counting v_t. The level of an edge $(r_t, b_t) \in E_t$ is defined as the level of the red node r_t of the edge. Further, we define the level of a cluster H of the laminar system as the level of a corresponding node in $T_{\mathcal{H}}$. (Note that both nodes corresponding to H are on the same level.) Finally, we say that an edge e of G is *cut* on level $\ell \geq 1$ if both endpoints of e are contained in the same level $\ell - 1$ cluster but in different level ℓ clusters. We use $\text{level}(e)$ for an edge $e \in E$ to denote the level on which e is cut. Figure 1 gives an example of a complete laminar system and the corresponding decomposition tree.

3 The routing scheme

The oblivious routing scheme that for each pair u, v of nodes in V defines a unit flow between u and v is based on the solution of a certain concurrent multicommodity flow problem (CMCF-problem) for each cluster of the hierarchical decomposition $\mathcal{H}$. In order to specify these CMCF-problems we first define a weight function $\mathrm{w}_\ell : 2^V \to \mathbb{R}_0^+$ for each level $\ell \in \{0, \ldots, \text{height}(T_{\mathcal{H}})\}$ as follows:

$$\mathrm{w}_\ell(X) := \sum_{\substack{e \in X \times V \\ \text{level}(e) \leq \ell}} c(e) \quad .$$

Informally speaking, the weight function $\mathrm{w}_\ell(X)$ counts for a subset X, the capacity of all edges that are adjacent to nodes in X and are cut before, or at level ℓ in the hierarchical decomposition. The following properties of w_ℓ will be used intensively throughout the paper. First of all w_ℓ is *additive*, i.e. for a set $X = X_1 \uplus X_2$, $\mathrm{w}_\ell(X) = \mathrm{w}_\ell(X_1) + \mathrm{w}_\ell(X_2)$. Furthermore, for a level ℓ cluster H_{v_t} we have $\mathrm{w}_\ell(S_{v_t}) = \text{out}(S_{v_t})$. Finally, $\mathrm{w}_{\ell-1}(X) \leq \mathrm{w}_\ell(X)$ holds for any $\ell \in \{1, \ldots, \text{height}(T_G)\}$.

The CMCF-problem for a level ℓ cluster H_{v_t} of the decomposition tree is defined as follows. There are $|H_{v_t}|^2$ commodities $d_{u,v}$ for $u, v \in H_{v_t}$. The source for commodity $d_{u,v}$ is u, its sink is v and its demand is

$$\text{dem}(u, v) := \frac{\mathrm{w}_{\ell+1}(u) \cdot \mathrm{w}_{\ell+1}(v)}{\mathrm{w}_{\ell+1}(H_{v_t})} \quad .$$

We solve the CMCF-problem *in the* cluster H_{v_t}, i.e. the flow is restricted to use only links inside H_{v_t} and has to respect the link-capacities, i.e. the flow that traverses an edge must be smaller than the capacity of that edge. The throughput fraction of a solution to a CMCF-problem is the minimum, over all commodities, of the fraction of the commodity's demand that is actually met by the solution.

The following theorem shows that there is a good oblivious routing scheme if all CMCF-problems can be solved with a large throughput fraction.

Theorem 1 *Let $q_{\min}$ denote the minimum throughput fraction that is achieved for the CMCF-problem of a cluster of the hierarchical decomposition $\mathcal{H}$. Further, let h denote the height of $T_{\mathcal{H}}$. Then there is an oblivious routing scheme with competitive ratio $\mathcal{O}(h/q_{\min})$. This scheme can be constructed in polynomial time.*

Proof. We first describe the oblivious routing scheme. Let $s, t \in V$ be a source-target pair.

We construct a unit flow from s to t, as follows. The flow paths are chosen according to the path in $T_{\mathcal{H}}$ between the nodes that correspond to $\{s\}$ and $\{t\}$. Let $v_1, \ldots, v_r$ denote the tree nodes on this path and let $\ell(i)$ denote the level of node v_i.

Initially all the flow starts in s, i.e. in the only node contained in H_{v_1}. Then the flow is first distributed among the nodes in H_{v_2}; then it is distributed among nodes in H_{v_3} and so on, until it is distributed among nodes in $H_{v_r} = \{t\}$, so that all flow reaches the target t. The distribution of the flow among nodes in cluster H_{v_i} is not uniform but depends on the level of v_i and on its color. If v_i is a blue node, then a node $u \in H_{v_i}$ receives a fraction of $\mathrm{w}_{\ell(i)}(u)/\mathrm{w}_{\ell(i)}(H_{v_i})$ of the flow and if v_i is a red node, u receives a fraction of $\mathrm{w}_{\ell(i)+1}(u)/\mathrm{w}_{\ell(i)+1}(H_{v_i})$.

The intuition behind these values is as follows. If v_i is a blue node then the flow is distributed according to the weight of edges that leave or enter cluster H_{v_i}. (Recall that $\mathrm{w}_{\ell(i)}(H_{v_i})$ counts the capacity of all edges that leave H_{v_i}.) This is reasonable because the flow sent to v_i has either just entered cluster H_{v_i} or is going to leave this cluster in the next step. (For a flow that does not have to leave or enter H_{v_i}, either $s, t \notin H_{v_i}$ or $s, t \in H_{v_i}$ must hold. In the first case the flow would not be routed to v_i. In the second case v_i would be the node with the lowest level on the path from v_1 to v_r. This is a red node.) In both cases it seems a good idea to store the flow somehow close to the edges that connect H_{v_i} to the rest of the graph.

If v_i is a red node the flow distribution is done according to the weight of edges that leave or enter the sub-clusters of H_{v_i}. In the case that the flow is at a red node v_i it either has to enter a sub-cluster of H_{v_i} in the next step, or it has to leave H_{v_i}. It will turn out that the first case is more critical for deriving a good bound on the competitive ratio. Therefore the flow is distributed according to the edges leaving sub-clusters of H_{v_i}.

Let $f_i(u)$ denote the fraction of flow received by node u in cluster H_{v_i}. The transition from the distribution for cluster H_{v_i} to the distribution for cluster $H_{v_{i+1}}$ is done as follows. Let r_t denote the red node from $\{v_i, v_{i+1}\}$. A node $u \in H_{v_i}$ sends a fraction of $f_i(u) \cdot f_{i+1}(v)$ to node $v \in H_{v_{i+1}}$ using the flow paths of commodity $d_{u,v}$ from the definition of the multicommodity flow problem of cluster H_{r_t}. Obviously this transforms the flow distribution of H_{v_i} into the flow distribution of $H_{v_{i+1}}$ since the total flow sent to v will be $f_{i+1}(v)$.

Now, we argue that this oblivious routing scheme has a competitive ratio of $\mathcal{O}(h/q_{\min})$. Suppose that we are given routing demands between source-target pairs such that the congestion when routing these demands optimally, is 1. We show that the congestion when using the oblivious routing scheme described above is only $\mathcal{O}(h/q_{\min})$.

Fix an edge $e \in E$ and a level ℓ. Let r_t denote a red level ℓ node of $T_{\mathcal{H}}$ and let d denote the degree of r_t. We denote the children of r_t in $T_{\mathcal{H}}$ with b_i, $i \in \{1, \ldots, d\}$ and the father with b_t. We are interested in the load $L_{r_t}(e)$ that is created by the oblivious routing scheme on edge e for transforming distributions between H_{r_t} and clusters H_{b_t} and H_{b_i}, $i \in \{1, \ldots, d\}$. Obviously this load is 0 if e is not contained in cluster H_{r_t}, since in this case the corresponding CMCF does not use edge e. Therefore, let in the following r_t denote the level ℓ node such that e is contained in H_{r_t} if such a node exists. We will bound $L_{r_t}(e)$ by the following claim.

Claim 2 *For each edge $e \in E$, $L_{r_t}(e) = \mathcal{O}(1/q_{\min})$.*

Proof. First consider load created between clusters H_{r_t} and H_{b_t}. The flow that is sent between $u \in H_{r_t}$ and $v \in H_{b_t}$ is at most

$$\mathrm{flow}(u,v) := \left(\frac{w_{\ell+1}(u)}{w_{\ell+1}(H_{r_t})}\right) \cdot \left(\frac{w_\ell(v)}{w_\ell(H_{b_t})}\right) \cdot \mathrm{out}(H_{r_t}) \ . \tag{1}$$

This holds since $(\frac{w_{\ell+1}(u)}{w_{\ell+1}(H_{r_t})})$ is the fraction of the total flow that resides in u according to the distribution on cluster H_{r_t} and $(\frac{w_\ell(u)}{w_\ell(H_{b_t})})$ is the corresponding term for v in cluster H_{b_t}. Furthermore all flow that is sent between clusters H_{r_t} and H_{b_t} corresponds to demands between source-target pairs s, t for which exactly one node of s, t is contained in H_{r_t}. Therefore the value of this flow is at most $\mathrm{out}(H_{r_t})$ since an optimal algorithm can route the demands with congestion 1. (This would not be possible if the flow that needs to leave or enter H_{r_t} was be larger than $\mathrm{out}(H_{r_t})$.)

We can utilize $w_\ell(H_{b_t}) = \mathrm{out}(H_{r_t})$ and $w_\ell(v) \le \mathrm{w}_{\ell+1}(v)$ in Equation 1 and we get that $\mathrm{flow}(u,v) \le \frac{\mathrm{w}_{\ell+1}(u) \cdot \mathrm{w}_{\ell+1}(v)}{\mathrm{w}_{\ell+1}(H_{r_t})} \le \mathrm{dem}(u,v)$, where $\mathrm{dem}(u,v)$ is the demand between u and v in the CMCF-problem for H_{r_t}. Since the flow is sent according to the CMCF, the load for an edge e will be at most $1/q_{\min}$.

Now, we consider load created for sending flow between cluster H_{r_t} and clusters H_{b_i}, $i \in \{1, \ldots, d\}$. The flow that is sent between $u \in H_{r_t}$ and $v \in H_{b_i}$ is at most

$$\mathrm{flow}(u,v) := \left(\frac{w_{\ell+1}(u)}{w_{\ell+1}(H_{r_t})}\right) \cdot \left(\frac{w_{\ell+1}(v_i)}{w_{\ell+1}(H_{b_i})}\right) \cdot \mathrm{out}(H_{b_i}) \ .$$

Since $w_{\ell+1}(H_{b_i}) = \mathrm{out}(H_{b_i})$ (b_i is on level $\ell + 1$) we get that also in this case $\mathrm{flow}(u,v) \le \mathrm{dem}(u,v)$. Therefore the load on edge e due to this flow will be at most $1/q_{\min}$. Altogether this yields the claim. ■

Since each edge is contained in at most h different clusters, the theorem follows from the above claim. ■

4 Construction of the decomposition

In this section we show that for any graph $G = (V, E)$ we are able to construct a hierarchical decomposition $\mathcal{H}$, such that the height of the corresponding decomposition tree $T_{\mathcal{H}}$ is $\mathcal{O}(\log n)$ and in each cluster $H \in \mathcal{H}$ the CMCF-problem has a throughput fraction of at least $\Omega(1/\log^3 n)$. This yields an oblivious routing scheme for graph G which is $\mathcal{O}(\log^4 n)$-competitive with respect to congestion. Furthermore we show that this construction can be done in polynomial time with respect to c_{max} and the number of nodes in the graph G.

In order to formally define our construction algorithm we need some notation. Consider any set of nodes $X \subseteq V$ and a concurrent multicommodity flow problem on X. A cut in the subgraph induced by X is a partition of X into two subsets A and $B = X \setminus A$. The sparsity of a cut (A, B) is defined as $\frac{\mathrm{cap}(A,B)}{\mathrm{dem}(A,B)}$, where $\mathrm{dem}(A, B)$ is the demand of the CMCF-problem that is separated by the cut, i.e. the sum over all demands of commodities for which sources and destinations lie in different parts of the cut.

Clearly, the sparsity of a cut in X places an upper bound on the throughput fraction of the corresponding multicommodity flow problem. Let σ denote the maximum possible ratio between the throughput fraction of a CMCF-problem and the sparsity of an approximate sparsest cut on G, computed by a suitable algorithm. For general graphs there exist approximation algorithms such that $\sigma = \mathcal{O}(\log n)$ and for planar graphs there are algorithms with $\sigma = \mathcal{O}(1)$ (see [2] and [8]).

Denoting the throughput fraction of the CMCF-problem with q, we obtain that there exists a cut (A, B) such that

$$\frac{\mathrm{cap}(A,B)}{\mathrm{dem}(A,B)} \leq \sigma \cdot q \ .$$

Furthermore, such a cut can be constructed in polynomial time with respect to $|X|$. Let ϕ denote the minimum sparsity of a cut. Then $\mathrm{cap}(A,B)/\,\mathrm{dem}(A,B) \leq \sigma \cdot \phi$. Therefore we call (A, B) an *approximate sparsest cut*.

For the remainder of the section we define $\lambda = 64 \cdot \sigma \cdot \log n$ and $q_{\min} = 1/(24 \cdot \sigma \cdot \lambda)$. We say that a cluster H fulfills the *throughput property* if the solution to the CMCF-problem in H has a throughput fraction of at least $q_{\min}$. Notice that $q_{\min} = \Omega(1/(\log n \cdot \sigma^2)) = \Omega(1/\log^3 n)$.

In the following we describe an algorithm for partitioning a single level ℓ cluster H. If we partition H into subclusters then the function $\mathrm{w}_{\ell+1}(\cdot)$ is well defined over the subsets of H and so is the CMCF-problem in the cluster H. By appropriate partitioning our algorithm tries to ensure that H fulfills the throughput property. If this is possible we can apply our algorithm to the set V and then recursively to the computed subclusters. This yields a hierarchical decomposition $\mathcal{H}$ of graph G that consists of clusters in which the solution to the CMCF-problem has a sufficiently large throughput fraction. Additionally the algorithm ensures that the size of each subcluster of an input cluster H is at most a constant fraction of the size of H. Thus, the height of the decomposition tree $T_{\mathcal{H}}$ is at most logarithmic. Furthermore, the algorithm runs in polynomial time.

An important difficulty of this approach is that it turns out that not every subset H of V can be partitioned into subclusters such that the corresponding CMCF-problem on H fulfills the throughput property. To ensure that such a partitioning is possible, we need an additional precondition that has to be fulfilled by an input set for the algorithm. This precondition is as follows.

Definition 3 *A level ℓ cluster H fulfills the precondition if for all sets U, such that $|U| \leq \frac{3}{4}|S|$ the following condition holds:*

$$\lambda \cdot \mathrm{cap}(U, H \setminus U) \geq \mathrm{w}_\ell(U) \ .$$

Now we can formally describe the properties of the algorithm.

Lemma 4 *Let H be a level ℓ cluster, which contains at least two vertices and fulfills the precondition. Then it is possible to partition H into disjoint subclusters H_i with the following characteristics:*

1. *H fulfills the throughput property.*
2. *For each subcluster H_i we have $|H_i| \leq \frac{2}{3} \cdot |H|$.*
3. *Each subcluster H_i fulfills the precondition.*

Moreover this partitioning can be done in polynomial time with respect to $|H|$ and c_{max}, where c_{max} denotes the maximum capacity of a network link.

```
AssurePrecondition (R)
    P_R := {R}
    do
        for each R_i ∈ P_R do
            compute (A, B) - an approximate sparsest cut for G(R_i)
            ψ := sparsity of the cut (A, B)
            if ψ ≤ 4σ/λ then
                P_R := P_R \ {R_i}
                P_R := P_R ∪ {A, B}
    until we made no changes to P_R in this iteration
    return P_R
```

Figure 2: The algorithm AssurePrecondition

Now, we first argue that the algorithm characterized by the above lemma yields the construction of the hierarchical decomposition $\mathcal{H}$. First we apply the algorithm to the set V which is the only cluster on level 0 of the decomposition tree. V fulfills the precondition, because $\mathrm{w}_0(V) = 0$. The algorithm returns a partitioning of V that defines the function $w_1(\cdot)$ and yields the level 1 clusters which fulfill the precondition. We apply the algorithm recursively to all these clusters until we get singleton sets $\{v\}, v \in V$. By Property 1 our algorithm ensures that for each cluster the corresponding CMCF-problem has a good throughput fraction. Further, the height of the decomposition tree $T_{\mathcal{H}}$ is logarithmic because of Property 2 of the lemma. Since the number of all clusters on a single level of the decomposition tree is at most n, the number of clusters in the hierarchy $\mathcal{H}$ is at most $\mathcal{O}(n \cdot \log n)$. Therefore the total construction time is also polynomial with respect to c_{max} and n.
In the rest of this section we present our construction algorithm.

Proof of Lemma 4. The algorithm for partitioning a set H according to the requirements of Lemma 4 uses a subroutine that is described in the proof of the following lemma.

Lemma 5 *It is possible to partition any set $R \subseteq V$ into disjoint sets R_i, such that each R_i fulfills the precondition and $\sum_i \mathrm{out}(R_i) \leq 2\,\mathrm{out}(R)$. Moreover, this partitioning can be computed in polynomial time with respect to $|R|$.*

Proof. We use the algorithm AssurePrecondition described in Figure 2. The algorithm works as follows. We start with a partition that contains only R. In each iteration we consider each set R_i of the current partitioning $\mathcal{P}_R$. We define a concurrent multicommodity flow problem $\mathcal{G}$ with demands $\mathrm{dem}(u, v) = \mathrm{w}_\ell(u)/|R_i|$ for each ordered pair $u, v \in R_i$. Then we compute (A, B) – an approximate sparsest cut of R_i. Let ψ denote the sparsity of this cut, i.e., $\psi = \mathrm{cap}(A, B)/\left(\frac{|B|}{|R_i|} \cdot \mathrm{w}_\ell(A) + \frac{|A|}{|R_i|} \cdot \mathrm{w}_\ell(B)\right)$. If $\psi \leq \frac{4\sigma}{\lambda}$, then R_i is replaced by A and B in the current partitioning $\mathcal{P}_R$. We proceed until the sparsity of the computed approximate cut for each R_i is greater than $\frac{4\sigma}{\lambda}$. For simpler notation we denote the term $\frac{4\sigma}{\lambda}$ with Λ further on.

The algorithm runs in polynomial time because the number of iterations is bounded by $|R|$ and each iteration runs in polynomial time.

First we prove that after this algorithm has finished, each set R_i from the partitioning of R fulfills the precondition. Assume for contradiction that there exists a set R_i and $U \subseteq R_i$ such that $|U| \leq \frac{3}{4}|R_i|$ and $\lambda \cdot \operatorname{cap}(U, R_i \setminus U) < w_\ell(U)$. Let ϕ denote the sparsity of the sparsest cut for $\mathcal{G}(R_i)$. We derive a bound on ϕ and thus also on the sparsity of the approximate sparsest cut ψ computed by the algorithm.

$$\lambda \cdot \operatorname{cap}(U, R_i \setminus U) < w_\ell(U) \leq 4\frac{|R_i \setminus U|}{|R_i|} \cdot w_\ell(U) \leq 4\left(\frac{|R_i \setminus U|}{|R_i|} w_\ell(U) + \frac{|U|}{|R_i|} w_\ell(R_i \setminus U)\right)$$

This gives that the sparsity of the cut $(U, R \setminus U)$ is at most $4/\lambda$. Therefore we get $\psi \leq \sigma \cdot \phi \leq 4\sigma/\lambda$ which is a contradiction, since in this case the algorithm would have divided R_i. Hence, each set $R_i \in \mathcal{P}_R$ fulfills the precondition.

To prove that $\sum_i \operatorname{out}(R_i) \leq 2\operatorname{out}(R)$ we consider a directed weighted graph H with node set V_H whose vertices correspond to edges of G leaving a partition R_i in the current partitioning $\mathcal{P}_R$. For simpler notation, let $R_H \subseteq V_H$ denote the set of nodes of H that represent edges which have exactly one endpoint in R, i.e. edges that contribute to $\operatorname{out}(R)$.

The edges of H will model the fact that newly introduced capacity is amortized against already existing capacity. In the following, we define the set of edges of H more precisely. Consider a step of the algorithm in which a set R_i is divided into sets A and B. Such a step increases the capacity of edges that leave partitions of $\mathcal{P}_R$ by the capacity of edges between A and B, i.e., $2\operatorname{cap}(A, B)$. (Each edge is counted twice since it leaves two partitions of $\mathcal{P}_R$.) For each such edge we introduce a new vertex in H. We want to derive a bound on the total capcity that is added to H. Therefore we amortize the newly created capacity $2\operatorname{cap}(A, B)$ against the capacity $\operatorname{out}(R_i)$.

Let $E_A = A \times \overline{R_i}$ and $E_B = B \times \overline{R_i}$ denote the set of edges that have one endpoint outside R_i and the other in A and B, respectively. ($E_A \cup E_B$ contains all edges that leave the set R_i.) In order do describe our amortization scheme we introduce the following notion. We say that the edges from $E_A \cup E_B$ *pay* for the new edges from $A \times B$. We define for each pair of edges $e \in A \times B$ and $e' \in E_A \cup E_B$ a price $\operatorname{pay}(e', e)$ that describes the amount that is paid by edge e' for edge e. We require that for each edge $e \in A \times B$ it holds that

$$\sum_{e' \in E_A \cup E_B} \operatorname{pay}(e', e) \geq 2c(e) \ ,$$

i.e., we pay enough for edge e.

We model this payment in the graph H via a directed edge from $v_{e'}$ to v_e that is weighted with $\operatorname{pay}(e', e)$. Then the above requirement simply states that for each node $v_e \in V_H \setminus R_H$ (a node that is added to H during the running time of AssurePrecondition), the weight of incoming edges must be at least as large as two times the weight of v_e, i.e., $c(e)$.

The exact definition of the function $\operatorname{pay}(\cdot, \cdot)$ is as follows. For an edge $e_a \in E_A$ we define

$$\operatorname{pay}(e_a, e) := 2\Lambda \cdot \frac{c(e)}{\operatorname{cap}(A, B)} \cdot \frac{|B|}{|R_i|} \cdot c(e_a),$$

and for an edge $e_b \in E_B$ we define

$$\operatorname{pay}(e_b, e) := 2\Lambda \cdot \frac{c(e)}{\operatorname{cap}(A, B)} \cdot \frac{|A|}{|R_i|} \cdot c(e_b) \ .$$

In order to simplify our notation we extend the function $\text{pay}(\cdot,\cdot)$ to vertices of H, i.e. for two vertices $v_e, v_{e'} \in V_H$ that correspond to edge $e, e' \in V \times V$ we define $\text{pay}(v_{e'}, v_e) = \text{pay}(e', e)$ which describes the weight of edge $(v_{e'}, v_e) \in E_H$.

The following claim shows that by the above definition we pay enough for new edges.

Claim 6 $\forall v_e \in V_H \setminus R_H :\ \sum_{v \in V_H} \text{pay}(v, v_e) \geq 2c(e)$.

Proof. Let e denote an edge that is created when partitioning a set R_i into A and B. We can estimate the incoming edges of v_e by

$$\begin{aligned}
\sum_{v \in V_H} \text{pay}(v, v_e) &= \sum_{e' \in E_A \cup E_B} \text{pay}(v_{e'}, v_e) \\
&= \sum_{e_a \in E_A} \text{pay}(v_{e_a}, v_e) + \sum_{e_b \in E_B} \text{pay}(v_{e_b}, v_e) \\
&= \sum_{e_a \in E_A} 2\Lambda \cdot \frac{|B|}{|R_i|} \cdot \frac{\mathrm{c}(e)}{\text{cap}(A,B)} \cdot \mathrm{c}(e_a) + \sum_{e_b \in E_B} 2\Lambda \cdot \frac{|A|}{|R_i|} \cdot \frac{\mathrm{c}(e)}{\text{cap}(A,B)} \cdot \mathrm{c}(e_b) \\
&= 2\Lambda \cdot \frac{\mathrm{c}(e)}{\text{cap}(A,B)} \cdot \left(\frac{|B|}{|R_i|} \cdot \mathrm{w}_\ell(A) + \frac{|A|}{|R_i|} \cdot \mathrm{w}_\ell(B) \right) \\
&\geq 2\,\mathrm{c}(e),
\end{aligned}$$

where the last step follows from the fact that (A, B) is a cut with sparsity at most Λ which implies that $\Lambda \cdot \left(\frac{|B|}{|R_i|} \cdot \mathrm{w}_\ell(A) + \frac{|A|}{|R_i|} \cdot \mathrm{w}_\ell(B) \right) \geq \text{cap}(A, B)$. ■

The following claim relates the total weight of edges leaving node $v_e \in V_H$ to $c(e)$.

Claim 7 *The total payment of an edge e during the whole algorithm is at most* $\mathrm{c}(e)/2$*, i.e.*

$$\sum_{v \in V_H} \text{pay}(v_e, v) \leq \mathrm{c}(e)/2 \ .$$

Proof. Let $e = (v, u)$. We consider sequences of different sets in which v and u lie during the run of the algorithm. We denote these sequences of sets as $V_0, V_1, \ldots, V_l$ and $U_0, U_1, \ldots, U_k$ for v and u, respectively. For those sequences, let $n_i = |V_i|$ and $m_j = |U_j|$.

Each edge (v_e, v) of H corresponds to cutting some V_i into V_{i+1} and $V_i \setminus V_{i+1}$ or U_j into U_{j+1} and $U_j \setminus U_{j+1}$. First we consider the case in which V_i is cut into $A := V_{i+1}$ and $B := V_i \setminus V_{i+1}$. We can estimate the total weight of edges between v_e and nodes of H that represent edges from $A \times B$ as follows

$$\begin{aligned}
\sum_{e' \in A \times B} \text{pay}(v_e, v_{e'}) &= \sum_{e' \in A \times B} 2\Lambda \cdot \frac{|B|}{|V_i|} \cdot \frac{\mathrm{c}(e')}{\text{cap}(A,B)} \cdot \mathrm{c}(e) \\
&= 2\Lambda \cdot \mathrm{c}(e) \cdot \frac{|B|}{|V_i|} \cdot \frac{\sum_{e' \in A \times B} \mathrm{c}(e')}{\text{cap}(A,B)} \\
&= 2\Lambda \cdot \mathrm{c}(e) \cdot \frac{|B|}{|V_i|} \\
&= 2\Lambda \cdot \mathrm{c}(e) \cdot \frac{n_i - n_{i+1}}{n_i}
\end{aligned}$$

An analogous bound can be proved for the case when we divide U_j into U_{j+1} and $U_j \setminus U_{j+1}$, namely $\sum_{e' \in A \times B} \mathrm{pay}(v_e, v_{e'}) = 2\Lambda \cdot \mathrm{c}(e) \cdot \frac{m_j - m_{j+1}}{m_j}$.

Claim 8 *For any sequence of numbers $n = n_0 > \cdots > n_k \geq 1$ it holds that $\frac{n_0-n_1}{n_0} + \frac{n_1-n_2}{n_1} + \cdots + \frac{n_{k-1}-n_k}{n_{k-1}} \leq 2\log n$.*

Thus, the total weight outgoing from v_e can be estimated as

$$\begin{aligned}
\sum_{v \in V_H} \mathrm{pay}(v_e, v) &= 2\Lambda \cdot \mathrm{c}(e) \cdot \left[\left(\tfrac{n_0-n_1}{n_0} + \cdots + \tfrac{n_{k-1}-n_k}{n_{k-1}} \right) + \left(\tfrac{m_0-m_1}{m_0} + \cdots + \tfrac{m_{l-1}-m_l}{m_{l-1}} \right) \right] \\
&\leq 2\Lambda \cdot \mathrm{c}(e) \cdot (2\log n + 2\log n) \\
&\leq 8 \cdot \tfrac{4\sigma}{\lambda} \cdot \log n \cdot \mathrm{c}(e) \\
&\leq \mathrm{c}(e)/2 \ .
\end{aligned}$$

This gives the claim. ■

Now, we can use the above claims to show that $\sum_i \mathrm{out}(R_i) \leq 2\,\mathrm{out}(R)$. This is done by summing the weight of all incident edges for each node $v_e \in H$, where incoming edges are counted positively and outgoing edges are counted negatively. Recall that $R_H \subseteq H$ denotes the set of nodes of H that represent edges leaving set R in the graph G. We get

$$\begin{aligned}
0 &= \sum_{v_e \in V_H} \left(\sum_{v \in V_H} \mathrm{pay}(v, v_e) - \sum_{v \in V_H} \mathrm{pay}(v_e, v) \right) \\
&= \sum_{v_e \in V_H \setminus R_H} \left(\sum_{v \in V_H} \mathrm{pay}(v, v_e) - \sum_{v \in V_H} \mathrm{pay}(v_e, v) \right) - \sum_{v_e \in R_H} \left(\sum_{v \in V_H} \mathrm{pay}(v_e, v) \right) \\
&\geq \sum_{v_e \in V_H \setminus R_H} \left(2\,\mathrm{c}(e) - \tfrac{1}{2}\,\mathrm{c}(e) \right) - \sum_{v_e \in R_H} \left(\tfrac{1}{2}\,\mathrm{c}(e) \right) \ .
\end{aligned}$$

This gives $\mathrm{out}(R) \geq 3 \sum_{v_e \in V_H \setminus R_H} \mathrm{c}(e)$. Altogether we get

$$\begin{aligned}
2\,\mathrm{out}(R) &\geq \mathrm{out}(R) + 3 \sum_{v_e \in V_H \setminus R_H} c(e) \\
&\geq \mathrm{out}(R) + 2 \sum_{v_e \in V_H \setminus R_H} c(e) \\
&= \sum_i \mathrm{out}(R_i) \ ,
\end{aligned}$$

as desired. ■

Now, we describe the algorithm Partition for partitioning a cluster H according to the requirements of Lemma 4.

In each iteration of the algorithm we maintain a partitioning $\mathcal{P}_H$ that fulfills Requirements 2 and 3, i.e. each subcluster H_i has at most $\frac{2}{3}|H|$ nodes and fulfills the precondition. We begin with subclusters containing only one node. Clearly both requirements are fulfilled.

The general idea of the algorithm to guarantee that the solution for the CMCF-problem that corresponds to the final partitioning has a good throughput fraction, is as follows. In each iteration the algorithm checks whether the CMCF-problem that corresponds to the

PARTITION (S)
$\mathcal{P}_H := \{\{v\} \mid v \in H\}$
while H does not fulfill the throughput property
do
compute (A, B) - an approximate sparsest cut of H /* W.l.o.g. $|A| \leq |B|$ */
$U^* :=$ round(A)
for each $H_i \subseteq U^*$ **do** $\mathcal{P}_H := \mathcal{P}_H \setminus H_i$
$\mathcal{P}_H := \mathcal{P}_H \cup$ ASSUREPRECONDITION(U^*)
end
return $\mathcal{P}_H$

Figure 3: The algorithm Partition

current partitioning already has a throughput fraction larger than $q_{\min}$. If this is the case, the algorithm terminates, since all the requirements are fulfilled. Otherwise the algorithm tries to find a collection of sublusters H_i of the current partitioning that has a certain property, namely that $\text{out}(\biguplus_{i\in I} H_i) \ll \text{w}_{\ell+1}(\biguplus_{i\in I} H_i)$, where I is the index set of the collection of these subclusters. Then the algorithm removes all the subclusters that belong to this collection and adds a new single subcluster $U^* := \biguplus_{i\in I} H_i$ that simply contains all nodes from the collection. Then in a final step the algorithm partitions U^* with ASSUREPRECONDITION in order to ensure that every subcluster of the partitioning fulfills the precondition.

We call the replacement of all H_i by clusters that results from ASSUREPRECONDITION(U^*) a *local improvement* of the algorithm. A key result for the construction is that we show that in each local improvement step the total capacity of edges that connect different subclusters decreases at least by a constant. Since this capacity is clearly bounded from below by 0, the algorithm will terminate after at most $|H|^2 \cdot \text{c}_{max}$ iterations, i.e. in polynomial time. Furthermore, since we show that the algorithm always makes an improvement step if H does not fulfill the throughput property, we can conclude that after termination the CMCF-problem on H can be solved with throughput fraction of at least $q_{\min}$.

Now we describe how the algorithm finds a collection of subclusters such that $\text{out}(\biguplus_{i\in I} H_i) \ll \text{w}_{\ell+1}(\biguplus_{i\in I} H_i)$ if H does not fulfill the throughput property. First we compute an approximate sparsest cut (A, B) corresponding to the CMCF-problem in H. Without loss of generality we can assume that $|A| \leq |B|$.

For the sparsity of this cut we have $\text{cap}(A, B)/\text{dem}(A, B) \leq \sigma \cdot q_{\min} = 1/(24\lambda)$, where

$$\begin{aligned}
\text{dem}(A, B) &= \sum_{u\in A, v\in B} \text{dem}(u, v) + \sum_{u\in A, v\in B} \text{dem}(u, v) \\
&= \sum_{u\in A, v\in B} \frac{\text{w}_{\ell+1}(u) \cdot w_{\ell+1}(v)}{w_{\ell+1}(H)} + \sum_{u\in B, v\in A} \frac{\text{w}_{\ell+1}(u) \cdot w_{\ell+1}(v)}{w_{\ell+1}(H)} \\
&= 2 \cdot \frac{\text{w}_{\ell+1}(A) \cdot w_{\ell+1}(B)}{w_{\ell+1}(H)} \\
&\leq 2 \cdot \text{w}_{\ell+1}(A) \ .
\end{aligned}$$

Combining the sparsity of cut (A, B) with the above inequality we get

$$\frac{\mathrm{cap}(A,B)}{\mathrm{w}_{\ell+1}(A)} \leq 2 \cdot \frac{\mathrm{cap}(A,B)}{\mathrm{dem}(A,B)} \leq \frac{1}{12\lambda} \ . \tag{2}$$

We cannot directly use the set A to improve the current partitioning of H, because it does not have to consist of whole subclusters H_i. Therefore we define a set $U^* := \mathrm{round}(A)$, using a construction taken from [11]. U^* is a rounding of the set A using the current partitioning, i.e. U^* is a union of disjoint subclusters H_i.

More precisely, let $A_i := A \cap H_i$ and $B_i := B \cap H_i$. We partition all indices of subclusters into sets I_L and I_S. If $|A_i| \geq \frac{3}{4} \cdot |H_i|$ then we say that H_i has *large intersection* with A and $i \in I_L$. Otherwise $i \in I_S$. U^* is a union of all subclusters H_i which have large intersection with A, i.e. $U^* := \biguplus_{i \in I_L} H_i$. Note that this definition and the fact that $|A| \leq \frac{1}{2}|H|$ ensure that $|U^*| \leq \frac{2}{3}|H|$. The following technical claim is proven in [11]. (see appendix)

Claim 9 $\frac{\mathrm{cap}(A,B)}{w_{\ell+1}(A)} \geq \frac{1}{4\lambda} \cdot \frac{\mathrm{out}(U^*)}{w_{\ell+1}(U^*)}$.

Using this claim and Equation 2 we get $\mathrm{w}_{\ell+1}(U^*) \geq 3\ \mathrm{out}(U^*)$. We are now able to prove that the PARTITION algorithm terminates.

Lemma 10 *The algorithm* PARTITION *terminates and its running time is polynomially bounded with respect to* c_{max} *and* $|H|$*.*

Proof. By $W(H) := \mathrm{w}_{\ell+1}(H) - \mathrm{out}(H)$ we denote the total capacity of edges that connect different subclusters in H. For proving the lemma it suffices to show that in each iteration of the PARTITION algorithm, $W(H)$ decreases by at least 1.

In each round we remove all the subclusters contained in U^*. Therefore, $W(H)$ decreases by $\mathrm{w}_{\ell+1}(U^*)$, i.e., by at least $3\ \mathrm{out}(U^*)$. After that we add clusters R_i returned by ASSUREPRECONDITION(U^*) and $W(H)$ increases by $\sum_i \mathrm{w}_{\ell+1}(R_i)$. Using Lemma 5 we obtain

$$\sum_i w_{\ell+1}(R_i) = \sum_i \mathrm{out}(R_i) \leq 2\ \mathrm{out}(U^*) \ .$$

Thus in each iteration $W(H)$ decreases by at least $\mathrm{out}(U^*)$. Since the capacity function $\mathrm{c}(.)$ is normalized, $\mathrm{out}(U^*) \geq 1$, which yields the lemma. ∎

We have proved that the algorithm PARTITION terminates and since the algorithm does not terminate unless the throughput fraction of the CMCF-problem in H is greater than $q_{\min}$, we can conclude that finally H fulfills the throughput property. This finishes the proof of Lemma 4. ∎

Theorem 11 *Let σ denote the gap between an approximate sparsest cut and the throughput fraction of a CMCF-problem on a graph G. Then a hierarchical decomposition of G that guarantees a competitive ratio of $\mathcal{O}(\log^2 n \cdot \sigma^2)$ for the oblivious routing problem can be constructed in polynomial time.*

Proof. Combining Theorem 1 and Lemma 4 gives the theorem. ∎

Remark 12 *If in the algorithm* ASSUREPRECONDITION *the subroutine for approximating a sparsest cut is replaced by an exact algorithm, the competitive ratio of the resulting oblivious routing scheme is $\mathcal{O}(\log^2 n \cdot \sigma)$.*

Proof. If a cut in the AssurePrecondition algorithm is computed optimally the precondition holds with $\lambda = \mathcal{O}(\log n)$ for every cluster. Therefore, we could compute a partitioning for which the throughput fraction of the CMCF-problem would be at least $q_{\min} = 1/(24 \cdot \sigma \cdot \lambda)$. Since the height of the decomposition tree remains logarithmic, Theorem 1 implies that the oblivious routing scheme yields a competitive ratio of $\mathcal{O}(\log^2 n \cdot \sigma)$. ■

Note that this version of the construction algorithm is not polynomial since computing a sparsest cut is NP-hard. But this modification of our construction algorithm proves the existence of an oblivious routing scheme with competitive ratio $\mathcal{O}(\log^3 n)$. Hence, it achieves the same result as in [11] but with a much simpler proof.

For planar networks [11] proves the existence of an oblivious routing scheme with competitive ratio $\mathcal{O}(\log^2 n)$. Since the sparsest cut gap σ is constant for planar networks, Theorem 11 shows that for such graphs a hierarchical decomposition with competitve ratio $\mathcal{O}(\log^2 n)$ can be constructed in polynomial time. Hence, there is no asymptotical difference for the competitive ratio on planar networks between the nonconstructive result of [11] and our constructive result.

References

[1] J. Aspens, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proc. of the 25th ACM Symp. on Theory of Computing (STOC)*, pages 623–631, 1993.

[2] 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, 1998.

[3] B. Awerbuch and Y. Azar. Local optimization of global objectives: competitive distributed deadlock resolution and resource allocation. In *Proc. of the 35th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 240–249, 1994.

[4] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke. Optimal oblivious routing in polynomial time. In *Proc. of the 35th ACM Symp. on Theory of Computing (STOC)*, 2003. to appear.

[5] A. Borodin and J. Hopcroft. Routing, merging and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30(1):130–145, 1985.

[6] C. Harrelson, K. Hildrum, and S. Rao. A polynomial-time tree decomposition to minimize congestion. In *Proc. of the 15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 34–43, 2003.

[7] C. Kaklamanis, D. Krizanc, and A. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Proc. of the 2nd ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 31–36, 1990.

[8] P. Klein, S. A. Plotkin, and S. Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proc. of the 25th ACM Symp. on Theory of Computing (STOC)*, pages 682–690, 1993.

[9] B. M. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Exploiting locality for networks of limited bandwidth. In *Proc. of the 38th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 284–293, 1997.

[10] B. M. Maggs, G. L. Miller, O. Parekh, R. Ravi, and S. L. M. Woo. Solving symmetric diagonally-dominant systems by preconditioning. manuscript, 2003.

[11] H. Räcke. Minimizing congestion in general networks. In *Proc. of the 43th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 43–52, 2002.

[12] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1981.

[13] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proc. of the 13th ACM Symp. on Theory of Computing (STOC)*, pages 263–277, 1981.

A Appendix

A.1 Proof of Claim 9

Let ℓ be the level of a cluster H. Since H fulfills the precondition and $|A| \leq \frac{1}{2}|H|$, we have

$$\lambda \cdot \mathrm{cap}(A, B) \geq \mathrm{w}_\ell(A) \geq \mathrm{w}_\ell(U^* \cap A) \ . \tag{3}$$

Analogously the $\ell + 1$ level sets H_i fulfill the precondition and for all $i \in I_S$ $A_i \leq \frac{3}{4}|H_i|$ and for all $i \in I_L$ $B_i \leq \frac{3}{4}|H_i|$. Thus,

$$\begin{aligned} &\forall i \in I_S \quad (\lambda + 1) \cdot \mathrm{cap}(A_i, B_i) \geq w_{\ell+1}(A_i) + \mathrm{cap}(A_i, B_i) = \mathrm{out}(A_i) \\ &\forall i \in I_L \quad (\lambda + 1) \cdot \mathrm{cap}(A_i, B_i) \geq w_{\ell+1}(B_i) + \mathrm{cap}(A_i, B_i) = \mathrm{out}(B_i) \ . \end{aligned}$$

Similarly we have

$$\lambda \cdot \sum_{i \in I_L} \mathrm{cap}(A_i, B_i) \geq \sum_{i \in I_L} \mathrm{w}_{\ell+1}(B_i) \geq \sum_{i \in I_L} \mathrm{w}_\ell(B_i) = \mathrm{w}_\ell(\biguplus_{i \in I_L} B_i) = \mathrm{w}_\ell(U^* \setminus A) \ .$$

Thus,

$$\begin{aligned} (\lambda + 1) \cdot \mathrm{cap}(A, B) \geq &(\lambda + 1) \cdot \sum_{i \in I_L} \mathrm{cap}(A_i, B_i) + (\lambda + 1) \cdot \sum_{i \in I_S} \mathrm{cap}(A_i, B_i) \\ \geq &\mathrm{w}_\ell(U^* \setminus A) + \sum_{i \in I_S} \mathrm{out}(A_i) \ . \end{aligned} \tag{4}$$

We also relate $\mathrm{cap}(A,B)$ to $\mathrm{cap}(U^*, H \setminus U^*)$.

$$
\begin{aligned}
\mathrm{cap}(U^*, H \setminus U^*) &= \mathrm{cap}(\biguplus_{i\in I_L} H_i, \biguplus_{i\in I_S} H_i) \\
&= \mathrm{cap}(\biguplus_{i\in I_L} A_i, \biguplus_{i\in I_S} B_i) + \mathrm{cap}(\biguplus_{i\in I_L} A_i, \biguplus_{i\in I_S} A_i) + \mathrm{cap}(\biguplus_{i\in I_L} B_i, \biguplus_{i\in I_S} H_i) \\
&\leq \mathrm{cap}(A,B) + \sum_{i\in I_S} \mathrm{out}(A_i) + \sum_{i\in I_L} \mathrm{out}(B_i) \\
&\leq \mathrm{cap}(A,B) + (\lambda+1)\cdot \sum_{i\in I_S} \mathrm{cap}(A_i,B_i) + (\lambda+1)\cdot \sum_{i\in I_L} \mathrm{cap}(A_i,B_i) \\
&\leq \mathrm{cap}(A,B) + (\lambda+1)\cdot \mathrm{cap}(A,B) \\
&= (\lambda+2)\cdot \mathrm{cap}(A,B)
\end{aligned}
\tag{5}
$$

We use inequalities 3, 4 and 5 to bound the expression $\mathrm{out}(U^*) + \sum_{i\in I_S} \mathrm{out}(A_i)$ as follows

$$
\begin{aligned}
\mathrm{out}(U^*) + \sum_{i\in I_S} \mathrm{out}(A_i) &\leq \mathrm{cap}(U^*, H\setminus U^*) + \mathrm{w}_\ell(U^*) + \sum_{i\in I_S} \mathrm{out}(A_i) \\
&\leq \mathrm{cap}(U^*, H\setminus U^*) + \mathrm{w}_\ell(U^*\cap A) + \mathrm{w}_\ell(U^*\setminus A) + \sum_{i\in I_S} \mathrm{out}(A_i) \\
&\leq \mathrm{cap}(A,B)\,((\lambda+2)+\lambda+(\lambda+1)) \\
&\leq 4\lambda\cdot \mathrm{cap}(A,B) \enspace .
\end{aligned}
$$

For the last inequality we utilized $\lambda \geq 3$.
Now,

$$
\begin{aligned}
\frac{\mathrm{w}_{\ell+1}(A)}{\mathrm{cap}(A,B)} &= \frac{\sum_i \mathrm{cap}(A_i, \overline{H_i})}{\mathrm{cap}(A,B)} \\
&= \frac{\sum_{i\in I_L} \mathrm{cap}(A_i, \overline{H_i}) + \sum_{i\in I_S} \mathrm{cap}(A_i, \overline{H_i})}{\mathrm{cap}(A,B)} \\
&\leq 4\lambda\cdot \frac{\sum_{i\in I_L} \mathrm{cap}(A_i, \overline{H_i}) + \sum_{i\in I_S} \mathrm{cap}(A_i, \overline{H_i})}{\mathrm{out}(U^*) + \sum_{i\in I_S} \mathrm{out}(A_i)} \\
&\leq 4\lambda\cdot \frac{\sum_{i\in I_L} \mathrm{cap}(H_i, \overline{H_i}) + \sum_{i\in I_S} \mathrm{out}(A_i)}{\mathrm{out}(U^*) + \sum_{i\in I_S} \mathrm{out}(A_i)} \\
&= 4\lambda\cdot \frac{\mathrm{w}_{\ell+1}(U^*) + \sum_{i\in I_S} \mathrm{out}(A_i)}{\mathrm{out}(U^*) + \sum_{i\in I_S} \mathrm{out}(A_i)} \\
&\leq 4\lambda\cdot \frac{w_{\ell+1}(U^*)}{\mathrm{out}(U^*)},
\end{aligned}
$$

where the last inequality follows since $\mathrm{w}_{\ell+1}(U^*) \geq \mathrm{out}(U^*)$. This finishes the proof. ∎