Optimal Hierarchical Decompositions for Congestion Minimization in Networks

Harald Räcke*
Department of Computer Science
University of Warwick
H.Raecke@warwick.ac.uk

ABSTRACT

Hierarchical graph decompositions play an important role in the design of approximation and online algorithms for graph problems. This is mainly due to the fact that the results concerning the approximation of metric spaces by tree metrics (e.g. [10, 11, 14, 16]) depend on hierarchical graph decompositions. In this line of work a probability distribution over tree graphs is constructed from a given input graph, in such a way that the tree distances closely resemble the distances in the original graph. This allows it, to solve many problems with a distance-based cost function on trees, and then transfer the tree solution to general undirected graphs with only a logarithmic loss in the performance guarantee.

The results about oblivious routing [30, 22] in general undirected graphs are based on hierarchical decompositions of a different type in the sense that they are aiming to approximate the bottlenecks in the network (instead of the point-to-point distances). We call such decompositions *cut-based decompositions*. It has been shown that they also can be used to design approximation and online algorithms for a wide variety of different problems, but at the current state of the art the performance guarantee goes down by an $O(\log^2 n \log\log n)$-factor when making the transition from tree networks to general graphs.

In this paper we show how to construct cut-based decompositions that only result in a logarithmic loss in performance, which is asymptotically optimal. Remarkably, one major ingredient of our proof is a distance-based decomposition scheme due to Fakcharoenphol, Rao and Talwar [16]. This shows an interesting relationship between these seemingly different decomposition techniques.

The main applications of the new decomposition are an optimal $O(\log n)$-competitive algorithm for oblivious routing in general undirected graphs, and an $O(\log n)$-approximation for Minimum Bisection, which improves the $O(\log^{1.5} n)$ approximation by Feige and Krauthgamer [17].

*The author acknowledges the support of DIMAP (the Centre for Discrete Mathematics and its Applications)

STOC'08, May 17–20, 2008, Victoria, British Columbia, Canada.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Routing and layout*

General Terms

Theory, Algorithms

Keywords

Oblivious Routing, Approximating Metrics by Tree Metrics, Hierarchical Decompositions

1. INTRODUCTION

A hierarchical decomposition of a graph is a recursive partitioning of the node set into smaller and smaller pieces until at the lowest level of the hierarchy individual pieces only contain single vertices. Such a decomposition is naturally associated with a tree network (*the decomposition tree*) that reflects this partitioning process and in which the leaf nodes correspond to nodes of the original graph.

In recent years, these types of decompositions have become a major tool for the development of approximation and online algorithms for graph problems, as they allow to approximate arbitrary undirected graphs by tree networks. The most prominent examples for this are the hierarchical decompositions that lie at the heart of the results concerning the probabilistic approximation of metric spaces by tree metrics.

In this line of work [10, 11, 14, 16] which was initiated by Bartal [10] the node set of a graph equipped with the shortest path metric is given, and it is shown how to compute a probabilistic distribution over hierarchical decompositions such that the leaf-to-leaf distances in the corresponding decomposition trees closely resemble the distances in the original graph. More formally, the distance between two leaf nodes in a decomposition tree is always larger than the distance between the corresponding vertices in the graph; and, conversely, the distance between two graph nodes is only a factor $1/f$ smaller (for some $f \geq 1$) than the *expected distance* between the corresponding leaf nodes when choosing a decomposition tree at random.

This result is very powerful as it allows for many problems to first solve the tree instance of the problem and then to transfer this solution to the original graph while paying only a factor f in the performance guarantee. This paradigm has been successfully used for problems like e.g. Group Steiner

Tree, Metric Labeling, Buy-at-Bulk Network Design, and Vehicle Routing. The original result by Bartal [10] provided a factor $f = O(\log^2 n)$ which was subsequently improved by him in [11] to $f = O(\log n \log\log n)$, and finally, Fakcharoenphol, Rao and Talwar showed a factor of $f = O(\log n)$ which is asymptotically tight.

Naturally, these techniques only work if the cost-function that has to be optimized is distance-related or, more precisely, is linear in the set of point-to-point distances. We will refer to these types of decompositions as distance-based decompositions in the following.

Räcke [30] introduced a decomposition that aims at constructing a tree that does not approximate point-to-point distances in the input graph (like the Bartal-technique described above), but instead approximates the cut structure of the graph in the following sense: Given a concurrent multicommodity flow problem (CMCF-problem[1]) in G, the optimum solution of the corresponding flow problem in a decomposition tree has a lower congestion; and, conversely, given a CMCF-problem in a decomposition tree the corresponding problem in G can be solved with a congestion that is only a factor f larger.[2] Following the same paradigm as above, i.e., solving problems on trees and then transferring the solution to general networks gives algorithms for e.g. Oblivious Routing [30], Simultaneous Source Location [2], Online Multicut [1] and k-multicut [21]. The original factor for this decomposition technique was $f = O(\log^3 n)$, which was later improved by Harrelson, Hildrum and Rao [22] to $f = O(\log^2 n \log\log n)$.

In this paper we show how to improve the guarantee for the latter type of hierarchical decompositions, which we call *cut-based* decompositions, from a factor $O(\log^2 n \log\log n)$ to $O(\log n)$. Then we show, among other applications, that this gives an oblivious routing algorithm with competitive ratio $O(\log n)$, and an approximation algorithm for Minimum Bisection with approximation guarantee $O(\log n)$. One main difference to the previous work by Harrelson et al. [22] is that instead of one decomposition tree, we compute a convex combination of decomposition trees, as it is the case for the distance-based decomposition schemes (FRT, Bartal). In fact, the FRT-decomposition is a major ingredient used in our proof, which shows that both types of decompositions, distance-based decompositions and cut-based decompositions, are closely related.

1.1 Related Work

A main motivation for finding good cut-based decompositions for general graphs is the problem of obliviously routing traffic in a network while minimizing the congestion, i.e., the maximum load of a network link. The problem is to set up a unit flow for each source-target pair $(s,t) \in V \times V$ that determines how demand between s and t is routed in the network. This unit flow is pre-specified without knowing the actual demands. When a demand vector $\vec{d}$ is given that specifies for each pair of nodes the amount of traffic to be sent, the demand-vector is routed by simply scaling the unit flow between a pair (s,t) by the corresponding demand d_{st} between the two nodes. This basically means that traffic is routed along pre-computed path and that no path-selection is done dynamically. This can be implemented very efficiently even in a distributed environment, and is the main motivation for considering such routing schemes.

The congestion $C_{\text{obl}}(G,\vec{d})$ that is obtained by the *oblivious routing*, is then compared to the optimal possible congestion $C_{\text{opt}}(G,\vec{d})$ that can be obtained for demand vector $\vec{d}$ in G. The competitive ratio of the oblivious routing scheme is defined as $\max_{\vec{d}}\{{}^{C_{\text{obl}}(G,\vec{d})}/_{C_{\text{opt}}(G,\vec{d})}\}$.

Räcke [30] gave the first oblivious routing scheme with a polylogarithmic competitive ratio ($O(\log^3 n)$) in general networks using a hierarchical decomposition technique. However, the result was non-constructive in the sense that only an exponential time algorithm was given to construct the hierarchy. This problem was addressed by Azar et al. [9] (see also [3]) who gave a polynomial time algorithm to construct an optimal oblivious routing scheme for a given graph without using a hierarchical decomposition. Polynomial time algorithms for constructing the hierarchical decomposition were independently given by Bienkowski et al. [13] and Harrelson et al. [22]. Whereas the first result shows a slightly weaker competitive ratio for the constructed hierarchy than the result in [30], the result by Harrelson et al. even improved the competitive ratio to $O(\log^2 n \log\log n)$, which is currently the best known bound. In this paper we improve the bound to $O(\log n)$ which is asymptotically optimal due to a lower bound of $O(\log n)$ on the grid ([27, 12]).

The other main motivation for our result is the Minimum Bisection problem that asks to partition the vertex set of a given graph into equal sized parts such that the number of edges between both parts is minimized. Feige and Krauthgamer [17] gave the first polylogarithmic approximation algorithm[3] for this problem with an approximation guarantee of $O(\log^2 n)$, which later improved to $O(\log^{1.5} n)$ because of the improved sparsest cut algorithm of Arora et al. [5]. Our results imply an approximation guarantee of $O(\log n)$ for this problem.

1.2 Notations and Definitions

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 $u, v \in V$.

Decomposition Trees.

A *decomposition tree* for the graph G is a rooted tree $T = (V_t, E_t)$ whose leaf nodes correspond to nodes in G, i.e., there is a one-to-one relation between nodes in G and leaf nodes in T. Whenever we use the concept of a decomposition tree for

[1] A CMCF-problem in a graph $G = (V,E)$ is defined by a demand vector $\vec{d}$, where for a node pair $(s,t) \in V \times V$, d_{st} denotes the demand that has to be shipped between s and t. The goal is to route all demand in the network G via a multicommodity flow while minimizing the congestion, which is the maximum over all edges of the flow along the edge divided by the bandwidth of the edge.

[2] Additionally, it is shown how to easily transfer a solution from the decomposition tree to the graph G.

[3] Interestingly, their algorithm is also based on a hierarchical decomposition of the graph but it does not seem to fit into a general framework which would make it applicable to a wider variety of problems.

a graph G we implicitely assume that we are also given an embedding of T into G. This means there is a *node mapping function* $m_V : V_t \to V$ that maps tree nodes to nodes in the graph, and because of the property of a decomposition tree this mapping function induces a bijection between leaf nodes of T and nodes in G. We are also given a function $m_E : E_t \to E^*$ that maps an edge $e_t = (u_t, v_t)$ of T to a path P_{uv} between the corresponding end-points $u = m_V(u_t)$ and $v = m_V(v_t)$ in G. For a decomposition tree T we also introduce functions $m'_V : V \to V_t$ and $m'_E : E \to E^*_t$ responsible for mapping from G to T. The function m'_V outputs for a node $v \in V$ the leaf node in T corresponding to v, and the function m'_E gives for an edge $e = (u, v) \in E$ the (unique) shortest path in T between $m'_V(u)$ and $m'_V(v)$.

Multicommodity Flows.

A multicommodity flow in a graph with n nodes is given by $\binom{n}{2}$ flows — one flow for each unordered pair $\{u, v\}$ of nodes in G (a commodity), where for each node-pair $\{u, v\}$ we arbitrarily choose one of the nodes as the source and the other as the target. When we consider a multicommodity flow on a decomposition tree we assume that the commodities are only formed by pairs of leaf nodes. For a multicommodity flow f_T on a decomposition tree we use $m(f_T)$ to denote the multicommodity flow that is obtained by mapping f_T to G via the edge-mapping function m_E. Formally, if an edge carries flow $f^i_T(e_t)$ for commodity i on a tree edge e_t, then the graph edge e carries flow $\sum_{e_t \in E_t : e \in m_E(e_t)} f^i_T(e_t)$ for commodity i. Similarly, we define for a flow f in G, $m'(f)$ as the flow in T obtained by mapping f to T.

Note that we can add multicommodity flows and scale them, and that in particular we have $m(\alpha \cdot f_T + \alpha' \cdot f'_T) = \alpha \cdot m(f_T) + \alpha' \cdot m(f'_T)$.

The Minimum Communication Cost Tree Problem.

In the Minimum Communication Cost Tree Problem we are given an undirected graph $G = (V, E)$. Every edge $e \in E$ has an associated *length* $\ell(e)$, and we use d_{uv} to denote the resulting shortest path distance between two nodes $u, v \in V$. Furthermore, we are given a requirement function $r : V \times V \to \mathbb{R}^+_0$ that specifies an amount of traffic that has to be sent between u and v. The goal is to route the requirements in a *tree-like fashion* while minimizing the total cost. Formally, the task is to construct a *decomposition tree* $T = (V_t, E_t)$, that minimizes

$$\text{cost}(T) = \sum_{(u,v)} d_T(u, v) \cdot r(u, v) \ ,$$

where $d_T(u, v)$ denotes the distance when connecting u and v via the tree. This is defined as follows. We define the length of a tree edge $e_t = (u_t, v_t)$ as the length of the corresponding path $m_E(e_t)$ in G. Then, the tree-distance $d_T(u, v)$ between graph nodes u and v is given by the shortest path distance between the corresponding leaf-nodes in T.

We can also write the above cost in a different way. A tree edge $e_t = (u_t, v_t)$ partitions the leaf nodes of the tree and, hence, the nodes of the graph, into two disjoint sets V_{u_t} and V_{v_t}. Let $r(e_t) := \sum_{u \in V_{u_t}, v \in V_{v_t}} r(u, v)$ denote the total requirement that has to cross the corresponding cut. All this traffic has to be forwarded via the path $m_E(e_t)$.

We define the *load* $\text{load}_T(e)$ that is induced on an edge $e \in E$ by tree T (and its embedding) as

$$\text{load}_T(e) := \sum_{e_t \in E_t : e \in m_E(e_t)} r(e_t) \ ,$$

which is the total traffic that goes over e if the requirement is routed via T along the chosen path system. With these definitions we can write the cost of a decomposition tree for a Minimum Communication Cost Tree instance as

$$\text{cost}(T) = \sum_{e \in E} \text{load}_T(e) \cdot \ell(e) \ .$$

We will use the following result about the Minimum Communication Cost Tree Problem which is due to Fakcharoenphol, Rao and Talwar [16].

THEOREM 1. *Given an instance for the Minimum Communication Cost Tree Problem, a solution with cost $O(\log n) \cdot \sum_e r(e) \cdot \ell(e)$ can be computed in polynomial time.*

1.3 Overview of Techniques

The proof technique used in this paper is very similar to the technique used by Charikar et al. [14] for finding a probabilistic embedding of a metric into a small number of dominating tree metrics. In the following we give a rough overview of both proofs.

Informally, the approach by Charikar et al. can be stated as follows. You are given a graph with edge-lengths, and the goal is to find an embedding into a probability distribution over dominating trees such that for each pair of nodes the expected stretch is small. They set up an instance of the MCCT-problem where the length of the edges are the same as the edge-lengths in the graph, and initially all requirements are one. For this instance solve the MCCT-problem and compute a tree T_1 for which the *average stretch* is low. Inevitably, some node-pairs will be stretched a lot by this tree. For the next iteration you re-weight the requirements by increasing it for pairs that experienced a large stretch in the first round, and decreasing it for others. Repeating this process it can be shown that you end up with a probability distribution of trees such that for every node-pair the expected stretch is small.

Our goal is to *re-route* the edges in the graph (map them to paths that connect the corresponding end-points) so that the collection of paths (one path for every edge) forms a tree, and the *load* induced by this re-routing onto a graph edge is not too large. For this we set up an MCCT-problem where the requirement $r(u, v) = \text{c}(u, v)$, and initially all edge-lengths are one. We can use Theorem 1 to find a tree T_1 (plus embedding) such that the *average load* of a graph edge is small. Inevitably, some graph edges will experience a large load. For the next iteration we re-weight the edge-lengths for graph edges that experienced a large load in the first round, and decrease it for others. Repeating this process it can be shown that you end up with a probability distribution of trees such that for every graph edge G the expected load is small.

Glossing over a lot of details this means that the difference between the result by Charikar et al. [14] and ours is mainly that we re-weight the edge-lengths instead of the requirements. This gives cut-based decompositions instead of the distance-based decompositions by Charikar et al. In the following sections we make these ideas precise.

2. APPROXIMATING THE BOTTLENECKS OF A GRAPH BY A TREE

In this problem we are given a graph $G = (V, E)$ together with a bandwidth function $\mathrm{c} : V \times V \to \mathbb{R}^+$ that describes the bandwidth of the edges in E with the convention that $\mathrm{c}(u, v) = 0$ iff $(u, v) \notin E$. Given a decomposition tree T for G we define the capacity $\mathrm{c}(u_t, v_t)$ of a tree edge $e_t = (u_t, v_t)$ as

$$\mathrm{c}(u_t, v_t) := \sum_{u \in V_{u_t}, v \in V_{v_t}} \mathrm{c}(u, v) \ ,$$

where V_{u_t} and V_{v_t} denote the two partitions of V induced by the cut corresponding to edge e_t. Given a multicommodity flow in G we want to compare its congestion in G, to its congestion in T.

THEOREM 2. *Suppose you are given a multicommodity flow f in G with congestion C_G. Then the flow $m'(f)$ obtained by mapping f to some decomposition tree T results in a flow in T that has congestion $C_T \leq C_G$.*

PROOF. Suppose an edge $e_t = (u_t, v_t)$ in the tree has congestion C_T. All traffic that traverses this edge in T has to traverse the cut in G between V_{u_t} and V_{v_t}. The total capacity of all edges over this cut is exactly equal to $\mathrm{c}(e_t)$. Hence, one of these edges must have relative load at least C_T. This gives $C_T \leq C_G$. □

Given a decomposition tree together with an embedding of this tree into the network G we can ask for the load that is induced on a graph edge e by this embedding. We define the load $\mathrm{load}_T(e)$ of an edge e as

$$\mathrm{load}_T(e) := \sum_{e_t \in E_t : e \in m_E(e_t)} \mathrm{c}(e_t) \ .$$

Note that this is the same as the load induced on an edge e in the solution of the MCCT-problem when the requirements are $r(u, v) = \mathrm{c}(u, v)$.

Let for an edge e, $\mathrm{rload}_T(e) := {}^{\mathrm{load}_T(e)}/_{\mathrm{c}(e)}$ denote the *relative load* of e induced by decomposition tree T. We are looking for a convex combination of decomposition trees such that for every edge the expected relative load is small, i.e.,

$$\textit{minimize } \beta := \max_{e \in E} \left\{ \sum_i \lambda_i \, \mathrm{rload}_{T_i}(e) \right\} \ .$$

CLAIM 3. *Suppose we are given a convex combination of decomposition trees with maximum expected relative load β, and suppose that we are given for each tree T_i a multicommodity flow f_i that has congestion 1 in T_i. Then, the multicommodity flow $\sum_i \lambda_i m_{T_i}(f_i)$ has congestion at most β when mapped to G.*

PROOF. Fix a tree T_i. Routing the flow f_i in the tree generates congestion at most 1, which means that the amount of traffic that is send along a tree edge $e_t = (u_t, v_t)$ is at most $\mathrm{c}(e_t)$. Hence, the total traffic that is induced on a graph-edge e when mapping $\lambda_i f_i$ to G is at most $\lambda_i \, \mathrm{load}_{T_i}(e)$. Therefore, the *relative* load induced on e when mapping all flows $\lambda_i f_i$ is at most $\sum_i {}^{\mathrm{load}_{T_i}(e)}/_{\mathrm{c}(e)} = \sum_i \lambda_i \, \mathrm{rload}_{T_i}(e) \leq \beta$. □

The above two claims give, e.g., a routing algorithm that solves a multicommodity flow problem with congestion at most β times the optimum. Given a multicommodity flow problem that can be routed with congestion $C_{\mathrm{opt}}(G)$ in G, simply compute the optimum solution for the corresponding flow problem in each tree T_i (this is straightforward as routing paths are unique in the tree). Then map the solution from each tree T_i to the graph G and scale it by a factor λ_i. This gives a solution in G with congestion at most $\beta \cdot \max_i \{C_{\mathrm{opt}}(T_i)\} \leq \beta \cdot C_{\mathrm{opt}}(G)$.

In the following section we show how to obtain a convex combination of decomposition trees for which $\beta = O(\log n)$. This gives the following main theorem.

THEOREM 4. *For a graph G there exists a convex combination of decomposition trees T_i defined by multipliers λ_i with $\sum_i \lambda_i = 1$ such that the following holds. Suppose we are given for each tree T_i a multi-commodity flow f_i that has congestion at most C. Then mapping the flows f_i to G while scaling flow f_i by λ_i results in a multicommodity flow $f := \sum_i \lambda_i m_{T_i}(f_i)$ in G that has congestion at most $O(\log n)$.*

2.1 Finding a Convex Combination of Trees

We can write the problem of deciding whether for a given value β there exists a convex combination of decomposition trees such that every edge has expected relative load at most β as the problem of finding a point in the following Polyhedron $P(\beta)$:

$$\begin{aligned} \forall e \in E \quad \sum\nolimits_i \lambda_i \, \mathrm{rload}_{T_i}(e) &\leq \beta \\ \sum\nolimits_i \lambda_i &\geq 1 \qquad (P(\beta)) \\ \forall i \qquad \lambda_i &\geq 0 \end{aligned}$$

The above is a mixed packing and covering problem (all entries are positive). There exists methods ([31], [26], [29], [18], [23]) for efficiently computing approximate solutions to such problems. The general idea behind most of these methods is to iteratively compute a maximum violated constraint in the dual and to increase the primal variable corresponding to this constraint. In the following we adapt the algorithm by Young [31] to our problem.

First, we write the edge-constraints in matrix form. Let $\mathcal{T}$ denote the set of all decomposition trees, and let M denote an $|E| \times |\mathcal{T}|$ matrix with $M_{eT} = \mathrm{rload}_T(e)$. Then we can write the edge-constraints from above as $M \cdot \vec{\lambda} \leq \beta \cdot \vec{1}$. Define for an $|E|$-dimensional vector $\vec{x}$

$$\mathrm{lmax}(\vec{x}) := \ln \left(\sum\nolimits_e \mathrm{e}^{x_e} \right) \geq \max_e \{x_e\} \ .$$

We can view $\mathrm{lmax}(\cdot)$ as a "smooth" function that approximates the maximum function. Instead of showing the existence of a point in $P(\beta)$ for some $\beta = O(\log n)$, we show something stronger, namely we show that there are values λ_i that fulfill

$$\begin{aligned} \mathrm{lmax}(M\vec{\lambda}) &\leq \beta \\ \sum\nolimits_i \lambda_i &\geq 1 \\ \forall i \qquad \lambda_i &\geq 0 \end{aligned}$$

The key to this analysis is to understand by how much $\mathrm{lmax}(M\lambda)$ changes when we increase the multiplier λ_i for a tree T_i. We follow the analysis of Young [31]. We first define

```
find_convex_combination()
    for all i: λ_i := 0
    while Σ_i λ_i < 1 do
        find a tree T_i with partial_i(λ⃗) ≤ O(log n)
        ℓ_i := max_e{rload_{T_i}(e)}
        δ_i := min{ℓ_i, 1 − Σ_i λ_i}
        λ_i := λ_i + δ_i
    return λ⃗
```

Figure 1: The algorithm for finding a good convex combination of decomposition trees.

a function $\text{partial}'_e(\vec{x})$ that gives the sensitivity of lmax when changing the load on an edge e:

$$\text{partial}'_e(\vec{x}) := \frac{\partial \,\text{lmax}(\vec{x})}{\partial x_e} = \frac{\text{e}^{x_e}}{\sum_e \text{e}^{x_e}} \ .$$

Now, we define functions $\text{partial}_i(\vec{\lambda})$ that measure the change in $\text{lmax}(M\vec{\lambda})$ when changing multiplier λ_i:

$$\begin{aligned}\text{partial}_i(\vec{\lambda}) &:= \frac{\partial \,\text{lmax}(M\vec{\lambda})}{\partial \lambda_i} \\ &= \sum_e \text{rload}_{T_i}(e) \cdot \text{partial}'_e(M\vec{\lambda}) \ .\end{aligned}$$

When changing a vector $\vec{x}$ by adding a vector $\vec{\epsilon}$ we can estimate the change $\text{lmax}(\vec{x}+\vec{\epsilon}) - \text{lmax}(\vec{x})$ via the partial derivatives, i.e., we would expect that $\text{lmax}(\vec{x}+\vec{\epsilon}) - \text{lmax}(\vec{x}) \approx \sum_e \epsilon_e \cdot \text{partial}'_e(\vec{x})$ for small enough $\vec{\epsilon}$. The following lemma by [31] makes this precise.

LEMMA 5. *For all $\vec{x}, \vec{\epsilon} \geq 0$ with $0 \leq \epsilon_e \leq 1$,*

$$\text{lmax}(\vec{x}+\vec{\epsilon}) \leq \text{lmax}(\vec{x}) + 2\sum_e \epsilon_e \,\text{partial}'_e(\vec{x}) \ .$$

LEMMA 6. *For some decomposition tree T_i we use $\ell_i := \max_e\{\text{rload}_{T_i}(e)\}$ to denote the maximum load induced on a link in G. Let $\vec{\delta} = \delta_i \vec{e}_i$ with $\delta_i \leq \frac{1}{\ell_i}$. Then,*

$$\begin{aligned}&\text{lmax}(M(\vec{\lambda}+\vec{\delta})) \\ &\quad\leq \text{lmax}(M\vec{\lambda}) + 2\sum_e (M\vec{\delta})_e \,\text{partial}'_e(M\vec{\lambda}) \\ &\quad= \text{lmax}(M\vec{\lambda}) + 2\delta_i \,\text{partial}_i(\vec{\lambda}) \ ,\end{aligned}$$

since $(M\vec{\delta})_e = \delta_i \,\text{rload}_{T_i}(e) \leq 1$.

This means that if we have a tree T_i that has $\text{partial}_i(\vec{\lambda}) \leq \beta$, then increasing the variable λ_i by $\delta_i \leq 1/\ell_i$, causes $\sum_i \lambda_i$ to increase by δ_i, while $\text{lmax}(M\vec{\lambda})$ only increases by $2\delta_i\beta$. Repeating this until $\sum_i \lambda_i$ is larger than 1, gives a set of variables λ_i that fulfill the constraints in Polyhedron $P(\beta)$. The following lemma shows that we can always find a tree with $\text{partial}_i(\vec{\lambda}) \leq \beta$ for $\beta = O(\log n)$.

LEMMA 7. *For a vector $\vec{\lambda}$ of multipliers with a polynomial number of non-zero entries we can efficiently compute a tree T_i such that $\text{partial}_i(\vec{\lambda}) = O(\log n)$.*

PROOF. For a tree T_i we have

$$\begin{aligned}\text{partial}_i(\vec{\lambda}) &= \sum_e \text{rload}_{T_i}(e) \frac{\text{e}^{(M\vec{\lambda})_e}}{\sum_e \text{e}^{(M\vec{\lambda})_e}} \\ &= \sum_e \text{load}_{T_i}(e) \frac{\text{e}^{(M\vec{\lambda})_e}}{\text{c}(e)\sum_e \text{e}^{(M\vec{\lambda})_e}}\end{aligned}$$

If we define $\ell(e) := \text{e}^{(M\vec{\lambda})_e}/\text{c}(e)\sum_e \text{e}^{(M\vec{\lambda})_e}$ to be the length of an edge, then finding the tree T_i such that the above is minimized is the Minimum Communication Cost Tree Problem where the requirements are $r(u,v) = \text{c}(u,v)$. Applying Theorem 1 gives a solution with total cost at most

$$\begin{aligned}&O(\log n) \cdot \sum_e r(e) \cdot \ell(e) \\ &\qquad\leq O(\log n) \cdot \sum_e \text{c}(e) \cdot \frac{\text{e}^{(M\vec{\lambda})_e}}{\text{c}(e)\ \sum_e \text{e}^{(M\vec{\lambda})_e}} \\ &\qquad= O(\log n) \ ,\end{aligned}$$

as desired. □

Hence, the algorithm given in Figure 1 finds a convex combination of decomposition trees for which the maximum average load of an edge is at most $O(\log n)$. It remains to bound the number of iterations of this algorithm.

LEMMA 8. *The number of iterations is $O(|E| \log n)$.*

PROOF. Define a potential function $\sum_e \sum_i \lambda_i \,\text{rload}_{T_i}(e)$ that describes the total relative load that has been placed on the edges so far. The potential function is bounded by $O(\log n) \cdot |E|$ as the relative load induced on an edge does not exceed $O(\log n)$ in the end.

For an iteration of the algorithm let e' denote the edge that has the largest relative load for the chosen tree T_i. We increase λ_i by $1/\ell_i$ (except maybe for the last iteration). Hence, $\sum_i \lambda_i \,\text{rload}_{T_i}(e')$ increases by 1, which in turn means that the potential function increases by 1. This gives a bound of $O(|E| \log n)$ on the number of iterations. □

3. APPLICATIONS

In this section we apply our main theorem (Theorem 4) to various network problems. All these problems either aim at minimizing the congestion for a communication problem or aim at finding a small cut in the graph. On the one hand we improve the guarantees for various problems for which the decomposition technique of Harrelson et al. [22]

currently provides the best bounds. This leads usually to an improvement by a $\log n \log\log n$ factor. Let us note however that simply replacing the decomposition of [22] with our new decomposition is not possible for all problems. The Simultaneous Source Location problem [2] e.g. has a bicriteria solution that depends on the decomposition of Harrelson et al., and the results in this paper don't seem to give improved bounds.

We also give improved guarantees for Minimum Bisection and Online Multicast Routing which are problems where the currently best known algorithms ([17] and [8]) do not depend on the hierarchical decomposition of Harrelson et al.

Min Bisection.

The Minimum Bisection problem gets an input graph G with an even number of vertices and asks for a partitioning of the vertex set into two equal size sets B and W such that the capacity of edges between these two sets is minimized. In the following we call vertices in B *black vertices* and vertices in W *white vertices*. Further, we use $\text{cost}_G(B, W)$ to denote the cost of the bisection (B, W).

In order to apply Theorem 4 we need to extend the definition of a bisection to decomposition trees. For a decomposition tree T_i we define a leaf bisection to be a partition of the leaf nodes of T_i into equal size sets W_i and B_i. We define the *cost* $\text{cost}_{T_i}(B_i, W_i)$ of a leaf bisection of T_i as the minimum capacity of edges that have to be removed in order to disconnect W_i from B_i. The following lemma follows from a straightforward dynamic programming algorithm.

LEMMA 9. *On a tree, a leaf bisection with minimum cost can be computed in polynomial time.*

Our algorithm for Minimum Bisection is as follows. First, compute the convex combination of decomposition trees defined in Theorem 4. Then compute a minimum leaf bisection for each decomposition tree in the support. Each of these leaf bisections also defines a bisection in G. Output the best of these bisections.

In the following we argue that this algorithm gives an $O(\log n)$-approximation. Let (B, W) denote a bisection in the graph G. Since each decomposition tree is a better communication network (due to Theorem 2) it follows that $\text{cost}_{T_i}(B, W) \geq \text{cost}_G(B, W)$. To see this consider attaching a super-source s to all nodes in B and a super-sink t to nodes in W via infinite-capacity edges. In G we can then route a single-commodity flow of value $\text{cost}_G(B, W)$ between s and t with congestion 1. This gives rise to a multi-commodity flow where all commodities with non-zero flow have one end-point in B and the other in W. Transferring this multicommodity flow to T_i gives gives congestion at most 1 in T_i according to Theorem 2. This however would not be possible if the sets B and W could be separated by removing a set of edges of total capacity strictly less than $\text{cost}_G(B, W)$. Hence, $\text{cost}_{T_i}(B, W) \geq \text{cost}_G(B, W)$.

Let (B^*, W^*) denote the optimum bisection in graph G and let (B_i^*, W_i^*) denote the optimum bisection in tree T_i. Assume for contradiction that none of the tree bisections (B_i^*, W_i^*) is a β-approximation, where $\beta = O(\log n)$ is the factor in Theorem 4. This means that $\forall i : \text{cost}_{T_i}(B^*, W^*) \geq \text{cost}_{T_i}(B_i^*, W_i^*) \geq \text{cost}_G(B_i^*, W_i^*) > \beta \cdot \text{cost}_G(B^*, W^*)$.

Hence, we can set up a demand-vector $\vec{d_i}$ for every tree that routes demand between nodes in B^* and W^* such that the total routed demand is larger than $\beta \cdot \text{cost}_G(B^*, W^*)$. We can route the convex combination of these demands in G with congestion at most β. However, this means that the capacity of edges between B^* and W^* must be larger than $\text{cost}_G(B^*, W^*)$. This is a contradiction.

Oblivious Routing.

An oblivious routing algorithm sets up a unit flow for each source-target pair $(s, t) \in V \times V$ that determines how demand between s and t is routed in the network. This unit flow is pre-specified without knowing the actual demands. When a demand vector $\vec{d}$ is given that specifies for each pair of nodes the amount of traffic to be sent, the demand-vector is routed by simply scaling the unit flow between a pair (s, t) by the corresponding demand d_{st} between the two nodes.

The convex combination of decomposition trees given in Theorem 4 defines a unit flow for every source target pair, by combining for a pair (u, v) the paths between u and v in trees T_i where the path from T_i is weighted with λ_i. Given a demand-vector that can be routed with congestion C in G, routing it in a decomposition tree creates congestion less than C in the tree. Mapping the flows from all decomposition trees back, gives a flow in G that routes the demand-vector and has congestion at most $O(\log n)$ due to Theorem 4. Hence, the oblivious routing scheme has competitive ratio $O(\log n)$.

This improves on the result of Harrelson, Hildrum and Rao [22] and gives a tight bound for oblivious routing in general undirected graphs (see [27, 12] for the lower bound) .

It also matches the bound given by Aspnes et al. [6] for adaptive routing algorithms. As the lower bound of $\Omega(\log n)$ in [27, 12] holds for adaptive algorithms this shows that on worst-case networks adaptive algorithms cannot asymptotically outperform oblivious routing algorithms. It would be interesting to analyze whether there exist networks where adaptive algorithms are in fact asymptotically better than oblivious routing schemes.

Sparsest Cut.

Given a cut $C \subset V$ in the graph G and a demand vector $\vec{d}$ the sparsity $\phi(C, \vec{d})$ of C is defined by

$$\phi(C, \vec{d}) := \frac{\text{cap}(C)}{\sum_{(x,y):\{x,y\}\cap C=1} d_{xy}} \ .$$

The sparsest cut problem is the problem of finding a cut with minimum sparsity. Let e denote an edge of a decomposition tree T_i. Then the load on e when routing d in T_i is $1/\phi(C_e, \vec{d})$, where C_e denotes the cut in G that corresponds to edge e. Let $e_{\max}$ denote the edge with highest load in all decomposition trees with non-zero support ($\lambda_i > 0$), and let its load be $L_{\max}$. Then Theorem 4 says that we can route d with congestion $O(\log n)L_{\max}$ in G. This implies that the sparsest cut in G has a sparsity larger than $1/(O(\log n)L_{\max})$, as the inverse of the sparsity is a lower bound on the congestion that can be obtained for a flow problem. However, $C_{e_{\max}}$ is a cut of sparsity $1/L_{\max}$, which means that we have an $O(\log n)$-approximation to the sparsest cut problem. Of course, this does not improve the approximation guarantee of sparsest cut as there are better algorithms based on semidefinite programming [5, 4, 15].

Max concurrent flow-sparsest cut ratio.

The above discussion also establishes a ratio of at most $O(\log n)$ between the throughput that can be obtained for a maximum concurrent flow problem, and the sparsest cut. Again this is not new (see [24, 7, 25]) but it seems interesting that this ratio can be obtained while only looking at a restricted number of cuts. Only looking at the sparsest among all edges/cuts of decomposition trees that are in the support gives an upper bound on the throughput of the flow problem that is within $O(\log n)$ of what can be achieved.

Multicast Routing.

In the multicast-routing problem a routing request consists of a set of terminals instead of only two terminals as in standard routing. The above discussion about oblivious routing, sparsest cut, and maxflow mincut ratio also transfers to this setting. We obtain an $O(\log n)$ competitive routing algorithm for the online multicast routing problem. To the best of our knowledge the currently best algorithm for this problem is due to Awerbuch and Azar [8], and obtains a competitive ratio of $O(\log^2 n)$.

The existence of a maxflow mincut ratio of $O(\log n)$ for this problem was shown in [28].

Multicut.

In [1] the authors show how to apply [22] to multicut problems. In the minimum multicut problem we are given a set of source target pairs and the task is to remove an edge-set with minimum capacity such that all source-target pairs are separated. Let $C_{\text{opt}}(G)$ and $C_{\text{opt}}(T_i)$ denote the cost of an optimum multicut in G and T_i, respectively. On trees there is a maxflow-mincut relation for the multicut problem. Let for a multicut instance $\text{MCF}(G)$ and $\text{MCF}(T_i)$ the maximum total flow that can be send between source target pairs in G and T_i, respectively. Garg et al. [20] have shown that $C_{\text{opt}}(T_i) \leq 2\,\text{MCF}(T_i)$ holds for tree networks ($\text{MCF}(G) \leq C_{\text{opt}}(G)$ holds for any network). Further, they show how to obtain a 2-approximation for the problem on trees. We can approximate the Minimum Multicut problem by the following randomized algorithm:

- select a random tree T_i with $\Pr[T_i \text{ is chosen}] = \lambda_i$.
- compute an approximation to the minimum multicut in T_i.

Let $C_{\text{appr}}(T_i)$ denote the cost of the approximate solution in tree T_i. The expected cost of the above algorithm is

$$\begin{aligned}\sum\nolimits_i \lambda_i \,\text{cost}_{\text{appr}}(T_i) &\leq 2 \sum\nolimits_i \lambda_i \,\text{cost}_{\text{opt}}(T_i) \\ &\leq 4 \sum\nolimits_i \lambda_i \,\text{MCF}(T_i)\ .\end{aligned}$$

Let $\vec{d_i}$ denote a demand vector for T_i that can be routed with congestion one and ships $\text{MCF}(T_i)$ of flow between sources and targets. According to Theorem 4 the demand vector $\frac{1}{\delta}\sum_i \lambda_i \vec{d_i}$ can be routed in G with congestion one, where $\delta = O(\log n)$ denotes the factor in Theorem 4. This means that $\delta\,\text{MCF}(G) \geq \sum_i \lambda_i\,\text{MCF}(T_i)$. Plugging this into the above equation gives that the expected cost is bounded by $O(\log n) \cdot \text{cost}_{\text{opt}}(G)$. This means we get an $O(\log n)$-approximation to minimum multicut. This matches the best approximation guarantees for minimum multicut ([19]). The advantage of the tree technique is that it also works for the following extensions of the multicut problem.

Minimum k-multicut.

In the k-multicut problem we are given a set of source target pairs, and the task is to remove an edge-set of minimum capacity that separates at least k of these pairs. Golovin et al. [21] give an $\frac{8}{3}+\epsilon$ approximation on trees and use [22] to obtain an $O(\log^2 n \log\log n)$-approximation for general networks. Using the above randomized approach gives a randomized algorithm that has expected cost only $O(\log n)$ times the optimum.

Online Multicut.

Alon et al. [1] introduce an online version of the multicut problem. In this version the source-target pairs arrive one by one, and an online-algorithm has to remove edges separating the pairs without knowing future pairs that will appear. They demonstrate an $O(h)$-competitive algorithm for this problem for trees of height h. Since, all decomposition trees in the original decomposition [30] are of height $O(\log n)$ they can use this to to obtain a randomized algorithm for the Online Multicut Problem in general undirected graphs with competitive ratio $O(\log^3 n \log\log n)$.

If we want to improve on this result using the new decomposition technique we need to show an upper bound on the height of the decomposition tree. However, this is not true in general as sometimes the solution to the MCCT problem requires trees of large height. Albeit, it can be shown that if the total capacity of edges is polynomially bounded (where we scale the capacities such that the lowest capacity is one), then the MCCT-problems we generate allow a tree-solution with logarithmic height. This holds because of the following: Consider an MCCT-instance where the total requirement is bounded by a polynomial $p(n)$ and the lowest requirement is 1. Without loss of generality we can re-scale the distances such that $\sum_e r(e)d(e) = 1$. Then we know that edges of length larger than 1 can be removed as they will not be used. We can contract edges of length smaller than $1/(n^2 p(n))$, since adding them afterwards only increases the cost of a given solution by $1/n$ (which in relative terms makes for a very small factor). This is a problem instance for which all distances are within a polynomial factor of each other. Applying FRT to such an instance gives a decomposition tree with logarithmic height.

This gives the following lemma.

Claim 10. *There is an algorithm for the Online Multicut Problem with competitive ratio $O(\log^2 n)$ if edge-capacities are polynomially bounded.*

For the case that edge-capacities are not polynomially bounded one can use the following standard technique. First, partition all edges into classes such that the k-th class contains $E_k = \{e \in E \mid n^k \leq c(e) \leq n^{k+1}\}$, $k = 0, 1, 2 \ldots$. Then we generate for each E_k a graph G_k, as follows. We take the original graph $G = (V, E)$; delete all edges for which $c(e) \leq n^{k-2}$ and contract all edges for which $c(e) \geq n^{k+1}$. For a demand pair (u, v) we define its *threshold-capacity* as the maximum value c such that any cut between u and v contains an edge of at least c. We then *assign* (u, v) to the graph G_k such that $c \in [n^k, n^{k+1}]$. The following holds

- Every graph G_k has a polynomial ratio between minimum and maximum capacity.
- If a pair assigned to graph G_k is separated in G_k, then it is separated in the original graph if we additionally

remove all edges e with $c(e) \leq n^{k-2}$ from the graph. Note that this only increases the cost for a cut by a factor of 2.

This gives rise to the following algorithm for the Online Multicut Problem. Given an input-pair, first determine the graph G_k it is assigned to. Delete all edges with $c(e) \leq n^{k-2}$ and use the algorithm from Claim 10 to separate the pair in the graph G_k. The algorithm is $O(\log n)$-competitive since at any point there are only a constant number of graphs G_k that contain undeleted edges and have a node-pair assigned to them. Furthermore, by Claim 10 we are $O(\log n)$-competitive in each G_k. Altogether we get the following lemma

LEMMA 11. *There is an algorithm for the Online Multicut Problem with competitive ratio $O(\log^2 n)$.*

Dynamic Data Management.

In [27] Maggs et al. introduce a dynamic data management problem where the goal is to minimize the congestion in the network. They develop tree algorithms for this problem and show how to use decomposition trees to obtain algorithm for the mesh. Using the new decomposition technique provided by Theorem 4 gives $O(\log n)$-competitive algorithms for this problem in general undirected graphs.

Acknowledgement

The author would like to thank Anupam Gupta and Chandra Chekuri for useful discussions and suggestions for improving a preliminary version of this paper.

4. REFERENCES

[1] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. S. Naor. A general approach to online network optimization problems. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 577–586, 2004.

[2] K. Andreev, C. Garrod, B. M. Maggs, and A. Meyerson. Simultaneous source location. In *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 13–26, 2004.

[3] D. Applegate and E. Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs. In *Proceedings of the ACM Symposium on Communications Architectures & Protocols (SIGCOMM)*, pages 313–324, 2003.

[4] S. Arora, J. Lee, and A. Naor. Euclidean distortion and the sparsest cut. In *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC)*, pages 553–562, 2005.

[5] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings, and graph partitionings. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*, pages 222–231, 2004.

[6] J. Aspnes, Y. Azar, A. Fiat, S. A. 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, 1997. Also in *Proc. 25th STOC*, 1993, pp. 623–631.

[7] 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.

[8] B. Awerbuch and Y. Azar. Competitive multicast routing. *Wireless Networks*, 1(1):107–114, 1995.

[9] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke. Optimal oblivious routing in polynomial time. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC)*, pages 383–388, 2003.

[10] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

[11] Y. Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of the 30th ACM Symposium on Theory of Computing (STOC)*, pages 161–168, 1998.

[12] Y. Bartal and S. Leonardi. On-line routing in all-optical networks. *Theoretical Computer Science*, 221(1-2):19–39, 1999. Also in *Proc. 24th ICALP*, 1997, pp. 516–526.

[13] M. Bienkowski, M. Korzeniowski, and H. Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 24–33, 2003.

[14] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. A. Plotkin. Approximating a finite metric by a small number of tree metrics. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 379–388, 1998.

[15] S. Chawla, A. Gupta, and H. Räcke. An improved approximation to sparsest cut. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 102–111, 2005.

[16] J. Fakcharoenphol, S. B. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC)*, pages 448–455, 2003.

[17] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Review*, 48(1):99–130, 2006. Also in *Proc. 41st FOCS*, 2000, pp. 105–115 and in SICOMP 31:(4):1090–1118, 2002.

[18] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 300–309, 1998.

[19] N. Garg, V. V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.

[20] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

[21] D. Golovin, V. Nagarajan, and M. Singh. Approximating the k-multicut problem. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 621–630, 2006.

[22] C. Harrelson, K. Hildrum, and S. B. Rao. A polynomial-time tree decomposition to minimize congestion. In *Proceedings of the 15th ACM Symposium*

on Parallelism in Algorithms and Architectures (SPAA), pages 34–43, 2003.

[23] R. Khandekar. *Lagrangian Relaxation based Algorithms for Convex Programming Problems*. PhD thesis, Indian Institute of Technology Dehli, 2004.

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

[25] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. Also in *Proc. 35th FOCS*, 1994, pp. 577–591.

[26] M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*, pages 448–457, 1993.

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

[28] V. Nagarajan and R. Ravi. Approximation algorithms for requirement cut on graphs. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 209–220, 2005.

[29] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 495–504, 1991.

[30] H. Räcke. Minimizing congestion in general networks. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 43–52, 2002. Co-Winner of Best Paper Award.

[31] N. E. Young. Sequential and parallel algorithms for mixed packing and covering. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 538–546, 2001.