Abstract

In this work, we discuss the results and consequences of Harald Räcke's paper *Optimal Hierarchical Decompositions for Congestion Minimization in Networks.* We present how decomposition trees can be used to facilitate different kinds of graph problems, and give a short overview over previous developments in this area and in the area of Oblivious Routing, which was identified by the author as a key problem and improved in a series of publications involving decomposition trees.

1 Decomposition Trees for Oblivious Routing

Decomposition trees are a special form a graph transformations and used to develop approximation algorithms. The idea is to generate a decomposition tree for the graph, to run the algorithm on the decomposition tree (faster than on the original graph), and to convert the result back to the original graph, instead of running an algorithm on the original graph.

1.1 Oblivious Routing

Routing network traffic in flow networks is a crucial problem in computer science. A great variety of algorithms exist that focus on different aspects, such as reducing the length of the path or reducing the latency, or reducing the congestion of the flow network. The author of the paper claims that his results are useful in particular in the latter case [7].

The congestion of a network flow graph is defined as the maximum congestion of any edge, where the congestion of an edge is defined as the flow imposed on an edge, divided by the capacity of the edge. I.e., the congestion of an edge is the occupancy rate r of the edge. For a routing network, $0 \leq r \leq 1$ is enforced and an $r > 1$ implies that an edge is overloaded.

The key idea of oblivious routing is that the route does not depend on the current traffic in the network, i.e. every routing decision is made independently. This simiplifies the involved algorithms significantly and allows making decisions locally, without having knowledge about the complete system. A centralized $\mathcal{O}(\log n)$-approximation was already presented by Aspnes in 1993 [1].

1.2 Oblivious Routing with Decomposition Trees

The author presents an algorithm that approximates the optimal congestion inside a flow network, i.e. a multicommodity flow that minimizes the congestion of the graph, by a factor of $\mathcal{O}(\log n)$. The following list gives an outline of the steps involved in this procedure. We will discuss these in more detail in the remainder of this work.

1. Generate a number of decomposition graphs T_i of G, together with scaling factors λ_i (convex combinations).

2. Compute an optimal flow in every decomposition tree T_i with one commodity per T_i (easy).

3. Bound the quality of the convex combination of T_is induced by λ_i.

4. Bound the quality of the solution when it is converted from the decomposition tree back to the original graph.

1.3 Previous Work

The author pursued a lot of reserach in the area of routing problems. In 2002, he published a paper in which he proved non-constructively that an $\mathcal{O}(\log^3 n)$-approximation for minimizing congestion in flow networks (therefore resulting in an oblibious routing algorithm) exists using a decomposition tree [8]. This result was remarkable in a sense that, even though better non-oblivious routing algorithms existed at that time[1], he showed that no information about the distribution of the current load inside the network is necessary and a slightly worse approximaton can still be achieved. In 2003, Bienkowski et al. presented an $\mathcal{O}(\log^4 n)$-approximation, together with a simpler proof for the existence of an $\mathcal{O}(\log^3 n)$ algorithm [2]. In the same year, Azar et al. presented an efficent $\mathcal{O}(\log^3 n)$-approximation whose existence Räcke proved. Also in 2003, Harrelson et al. presented an $\mathcal{O}(\log^2 n \log\log n)$-approximation, which was the best known approximation until Räcke published the paper discussed in this work in 2008 [6].

Motivated by results for a better oblivious routing algorithm in directed graphs, Hajiaghayi et al. analyzed possible improvements for the undirected case. However, it turned out that these results could not advance the undirected case any further, and they proved a tighter lower bound of $\Omega(\frac{\log n}{\log\log n})$ instead [4].

2 Decomposition Trees

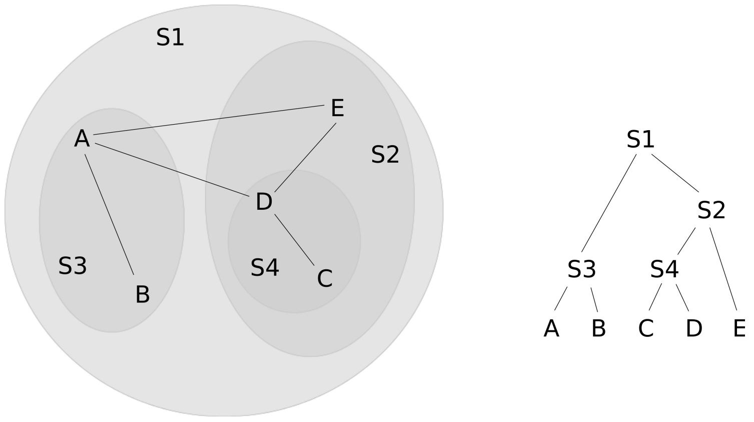

Figure 1: An example for a decomposition tree.

2.1 Notation and Definitions

We are given a graph $G = (V, E)$ that we want to transform into a tree graph T that is *similar* to G. Different ways for decomposing graphs have been proposed, among them distance-based decompositions and cut-based decompositions. We will focus on the latter in this work. In a decomposition tree, adjacent vertices are grouped in trees. These trees (or remaining single vertices) can again be grouped into trees, until, eventually, a single root is left over. For a decomposition tree $T = (V_t, E_t)$ for the graph G, we introduce the following notation and definitions.

- $m_V : V_t \rightarrow V$ is a function that maps nodes from T to vertices in G, especially leaf nodes in T to graph vertices V.
- $m'_V : V \rightarrow V_t$ is the m_V's inverse function.
- $m_E : E_t \rightarrow E^*$ maps an edge $e_t = (u_t, v_t)$ in T to a path P_{uv} (a set of edges) between $u = m_V(u_t)$ and $v = m_V(v_t)$ in G.

[1]In 1993, Aspnes et al. [1] presented a non-oblivious online routing algorithm which is an $\mathcal{O}(\log n)$-approximation.

- $m'_E : E \to E_t^*$ maps an edge in G to the shortest path connecting the corresponding nodes in T, i.e. the shortest path between $m'_V(u)$ and $m'_V(u)$ for an edge $e = (u, v) \in E$.

2.2 The Multicommodity Network Flow Problem

A multicommodity flow problem is a network flow problem with multiple flows in the same network. Formally, we are given an undirected graph $G = (V, E)$ and a capacity function $c : E \to \mathbb{R}_0^+$ that assigns every edge a maximum flow. For every unordered pair $\{u, v\}$ (commodity), we are looking for a flow in the graph network starting at the source node u and flowing into the sink node v, subject to flow conservation and capacity constraints. We are also given a demand value d per commodity, i.e. the amount of flow we wish to flow from u To v.

We can map a flow f_T in T back to a flow $m(f_T)$ in G using the edge mapping function m_E. If an edge e_t has flow $f_T^i(e_t)$ for commodity i, then an edge $e \in E$ has flow $\sum_{e_t \in E_t : e \in m_E(e_t)} f_T^i(e_t)$. For mapping a flow from G to T, we use the function $m'(f)$.

2.3 Approximating bottlenecks in a graph

In a graph, a bottleneck is a set of edges that limits the flow, i.e. if we increase the capacity of all these edges, we can increase the network flow. By approximating the bottlenecks efficiently, we can approximate multicommodity flows efficently, and therefore solve problems such as oblivious routing. In this section, we show how to approximate bottlenecks with decomposition graphs and prove the quality of this method. In a later section, we will discuss how to generate the decomposition graphs that we use in this section.

Flow Network We are given a graph $G = (V, E)$ with a capacity function $c(u, v)$ that limits the flow on the edge (u, v). We are also given a decomposition tree T for G with a capacity function $c_T(u_t, v_t)$ that is defined as follows.

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

We can think of the edge (u_t, v_t) as a cut. T is a tree, so removing an edge partitions the graph into two sets V_{u_t} and V_{v_t}.

Congestion in G and T The congestion of a graph is defined as the maximum congestion of an edge. The congestion of an edge is flow divided by capacity.

Lemma 1 *For a multicommodity flow f in G with congestion C_G, T has a congestion of $C_T \leq C_G$ when mapping f to T.*

Let $e_t = (u_t, v_t)$ be an edge in T that reaches the maximum congestion C_T. Note, that e_t defines a cut, i.e. it partitions T into V_{u_t} and V_{v_t} and all flow between these two vertex sets has to go through e_t. Now consider a cut between the edges $m_V(V_{u_t})$, $m_V(V_{v_t})$ in G. The flow crossing the cut in G is greater or equal to the flow crossing the edge e_t. According to the previous definition, the cut capacity in G equals $c(e_t)$. Note, that congestion is defined as flow divided by capacity. If f_T is the flow in T and f_G is the flow in G, then the following holds true since $\sum_{p \in cut_G} f_G(p) \geq f_T(e_t)$.

$$\exists e_p \in cut_G : \frac{f_T(e_t)}{c_T(e_t)} \leq \frac{f_G(e_p)}{c(e_p)}$$

Therefore, one of the cut edges in G must be above the average and $C_G \geq C_T$.

Embedding of T in G We define the load of an edge e induced by T as follows.

$$load_T(e) = \sum_{e_t \in E_T : e \in m_E(e_t)} c(e_t)$$

The relative load of an edge e induced by T is defined as follows.

$$rload_T(e) = \frac{load_T(e)}{c(e)}$$

Convex Combination of Decomposition Trees This step in the analysis is crucial. We are now given a convex combination of decomposition trees T_i. The following lemma bounds the congestion of the flow in the convex combination of T_is when it is converted back to G.

Lemma 2 *Let $\{(T_i, \lambda_i)\}$ be a convex combination of decomposition trees, i.e. $\forall i : \lambda_i \geq 0$, $\sum_i \lambda_i = 1$, and the maximum expected relative load be β, i.e. $\forall e \in E : \sum_i \lambda_i \cdot rload_{T_i}(e) \leq \beta$. Furthermore, in every T_i, there is a multicommodity flow f_i with congestion 1 in T_i. Then, the combined and scaled multicommodity flow $\sum_i \lambda_i \cdot m_{T_i}(f_i)$ has congestion less or equal to β when mapped back to G.*

For a specific T_i, we never send more flow across an edge than its capacity, since the congestion is 1. Therefore, for an edge $e \in E$, $f_G(e) \leq \lambda_i \cdot load_{T_i}(e)$. If we consider all T_i, then $f_G(e) \leq \sum_i \lambda_i \cdot load_{T_i}(e)$, resulting in a congestion of $\frac{\sum_i \lambda_i \cdot load_{T_i}(e)}{c(e)} = \sum_i \lambda_i \cdot rload_{T_i}(e)$. Since, by definition, this value is less or equal to β for every edge, the congestion in G is less or equal to β.

Approximating the optimal congestion Let G be a flow network graph with optimal congestion $C_{opt}(G)$. Given a *good* convex combination of decomposition trees that satisfies the requirements in Lemma 2, we can compute the flow in every T_i and map the flow back to G. According to Lemma 2, the congestion in G is less or equal to $\beta \max_i C_{opt}(T_i)$, which is, according to Lemma 1, less or equal to $\beta \cdot C_{opt}(G)$.

2.4 Generating a Convex Combination of Decomposition Graphs

In this section, we give a brief summary of how to generate good convex combination of decomposition graphs that satisfy the requirements of Lemma 2 with $\beta = \mathcal{O}(\log n)$. We can formulate this problem as follows. Find λ_i and T_i (a point in a polyhedron), such that the following inequalities are satisfied.

$$\forall e \in E : \sum_i \lambda_i \cdot rload_{T_i}(e) \leq \beta$$

$$\sum_i \lambda_i \geq 1$$

$$\forall i : \lambda_i \geq 0$$

This is equivalent to showing that for $\beta = \mathcal{O}(\log n)$ the polyhedron is non-empty. We can reformulate the problem as follows. Let us define a matrix M.

$$M = \begin{pmatrix} rload_{T_1}(e_1) & rload_{T_2}(e_1) & \cdots \\ rload_{T_1}(e_2) & rload_{T_2}(e_2) & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

Then, the first inequality can be written as follows.

$$M \cdot \vec{\lambda} \leq \beta \cdot \vec{1}$$

Instead of comparing every element of the vector, we can use the max function.

$$\max(M \cdot \vec{\lambda}) \leq \beta$$

The author now replaces this inequality by a stronger one that implies the original one, by defining $lmax(\vec{x}) = \ln(\sum_e e^{x_e}) \geq \max_e x_e$.

$$lmax(M \cdot \vec{\lambda}) \leq \beta$$

We now examine the partial derivates of *lmax* by an element of the vector x_e and by λ_i.

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

$$partial_i(\vec{\lambda}) = \frac{\partial lmax(M \cdot \vec{\lambda})}{\partial \lambda_i} = \sum_e rload_{T_i}(e) \cdot partial'_e(M \cdot \vec{\lambda})$$

These partial derivates are the basis for an iterative algorithm. We can estimate the change of $lmax(\vec{x})$ when adding $\vec{\epsilon}$, using the partial derivates, according to the following lemma that was shown by Young.

Lemma 3 *For all $\vec{x}, \vec{e} \geq 0$, $0 \leq \epsilon_e \leq 1$, the following holds true. $lmax(\vec{x} + \vec{\epsilon}) \leq 2 \cdot \sum_e \epsilon_e \cdot partial'_e(\vec{x})$.*

Computing a Convex Combination The author shows that for a T_i with $partial_i(\vec{\lambda}) \leq \beta$, increasing a variable λ_i by $\delta_i = \frac{1}{l_i}$, where $l_i = \max_e rload_{T_i}(e)$, increases $\sum_i \lambda_i$ by δ_i, but $lmax(M \cdot \vec{\lambda})$ only by $2 \cdot \delta_i \cdot \beta$. Note, that we want $lmax(M \cdot \vec{\lambda})$ to be as small as possible, while $\sum_i \lambda_i = 1$, since it is a convex combination. This gives rise to the following iterative algorithm for computing a convex combination.

```
$\forall i : \lambda_i = 0$
while $\sum_i \lambda_i < 1$ do
    $T_i =$ tree with $partial_i(\vec{\lambda}) \leq \beta$
    $l_i = \max_e rload_{T_i}(e)$
    $\delta_i = \min\{l_i, 1 - \sum_i \lambda_i\}$
    $\lambda_i = \lambda_i + \delta_i$
end
```

Figure 2: Algorithm for generating a convex combination of decomposition trees.

The author shows, that a T_i with $partial_i(\vec{\lambda}) = \mathcal{O}(\log n)$ can be computed efficiently by solving the Minimum Communication Cost Tree Problem. He also shows that the algorithm terminates in $\mathcal{O}(|E| \cdot \log n)$ steps.

2.5 Applications

The author shows that the idea presented in his paper can be applied to various graph problems and improves the best solution that was known at that point for some problems.

Oblivious Routing Harrelson et al. presented an algorithm for Oblivious Routing with an approximation of $\mathcal{O}(\log^2 n \log\log n)$ [5]. With the method shown in the author's paper, an $\mathcal{O}(\log n)$-approximation can be achieved.

Minimum Bisection In the Minimum Bisection Problem, we are given a graph $G = (V, E)$ and want to partition the vertex set into two equally-sized sets, such that cut capacity is minimized. An $\mathcal{O}(\log n)$-approximation can be achieved, which improves the best previously known $\mathcal{O}(\log^{1.5} n)$-approximation by Feige et al. [3].

3 Future Work

The $\mathcal{O}(\log n)$-approximation is asymptotically optimal, so we cannot get a better approximation algorithm.

References

[1] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 623–631, New York, NY, USA, 1993. ACM.

[2] Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, pages 24–33, New York, NY, USA, 2003. ACM.

[3] Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput.*, 31(4):1090–1118, April 2002.

[4] Mohammad T. Hajiaghayi, Robert D. Kleinberg, Tom Leighton, and Harald Räcke. New lower bounds for oblivious routing in undirected graphs. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 918–927, New York, NY, USA, 2006. ACM.

[5] Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, pages 34–43, New York, NY, USA, 2003. ACM.

[6] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 255–264, New York, NY, USA, 2008. ACM.

[7] Harald Räcke. Survey on oblivious routing strategies. In *Proceedings of the 5th Conference on Computability in Europe: Mathematical Theory and Computational Practice*, CiE '09, pages 419–429, Berlin, Heidelberg, 2009. Springer-Verlag.

[8] Harald Rcke. Minimizing congestion in general networks. In *IN PROCEEDINGS OF THE 43RD IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE (FOCS)*, pages 43–52, 2002.