Oblivious Routing and Minimum Bisection

Markus Kaiser

Technische Universität München
markus.kaiser@in.tum.de

Abstract. Oblivious routing is generalization of multi commodity flows where the actual demand function is unknown. This paper proofs the existence of an approximate solution using tree metrics which can easily be transformed into an $\mathcal{O}(\log n)$ approximation algorithm. This result is then applied to the minimum bisection problem asking for an vertex bisection with minimal cost in the edges between the sets, also resulting in an $\mathcal{O}(\log n)$ approximation.

1 Oblivious Routing

Maximum flow is a well understood algorithmic problem. Given a directed graph $G = (V, E)$, a mapping $c : E \to \mathbb{R}_0^+$ denoted by $c_e \coloneqq c(e)$ assigning a maximum non-negative capacity to every edge and a source and target node $s, t \in V$, we are asked to find a flow of maximum value from s to t. For simplicity, we assume that G is complete, which can be achieved by adding edges of capacity 0.

Definition 1 (Flow [CLRS01]). *A function $f : E \to \mathbb{R}_0^+$ assigning a throughput to every edge. It satisfies the following two properties:*

Capacity Constraint: *For all $e \in E$, we have $f_e \leq c_e$.*
Flow Conservation: *For all $v \in V \setminus \{s, t\}$, we have*

$$\sum_{(u,v)\in E} f_{uv} = \sum_{(v,w)\in E} f_{vw}.$$

We denote $f_e \coloneqq f(e)$ for $e \in E$ and $f_{uv} \coloneqq f(u, v)$ for $u, v \in V$.

The maximum flow problem can be solved in polynomial time using a number of algorithms, for example the Ford-Fulkerson method or Push-relabel algorithms, both of which are described in [CLRS01].

This paper will introduce a generalization of this problem known as oblivious routing and proof the existence of an $\mathcal{O}(\log n)$ approximation algorithm based on [Räc08] and described in [WS11]. After giving a definition of the problem, it is formulated as a linear problem and rewritten to yield the desired algorithm using a theorem giving an $\mathcal{O}(\log n)$ approximation of arbitrary metrics using tree metrics. Finally, the result is applied to give an $\mathcal{O}(\log n)$ approximation of the minimum bisection problem.

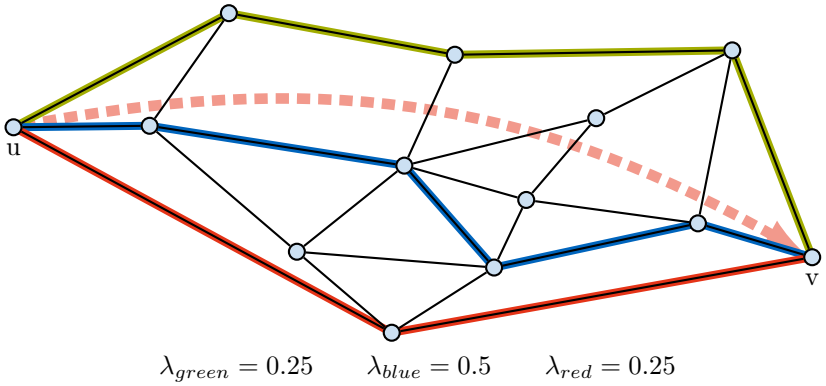

Fig. 1. A solution to the oblivious routing problem gives a set of paths and an associated convex combination for any two nodes $u, v \in V$. In this case, we are given the green, blue and red paths from u to v. The dashed demand d_{uv} gets routed along these three paths, split according to the factors λ_i.

1.1 Problem definition

To define oblivious routing, we first consider a simpler generalization of maximum flow, the multi-commodity flow problem. Instead of a single source and target node, we now allow an arbitrary number of nonnegative flow demands between two nodes given by a demand function $d : V^2 \to \mathbb{R}_0^+$ on an undirected graph. Every such demand d_{uv} must be routed from u to v using a set of u-v-paths and the total flow f_e on an edge $e \in E$ is the sum of all demands routed through it. The task of is now to find a flow satisfying all demands while exceeding the capacities of all edges as little as possible. This excess is called congestion.

Definition 2 (Congestion). *The congestion ρ attributed to a flow f denotes the smallest factor ρ such that for all edges $e \in E$ we have $f_e \leq \rho \cdot c_e$.*

$$\rho = \max_{e \in E} \frac{f_e}{c_e}$$

This problem is still solvable in polynomial time using linear programming [WS11]. One more generalization leads to the oblivious routing problem. We now want to find flow solutions without knowing the demands beforehand, i.e. we want to find a set of u-v-paths and associated fractions of demands for all $u, v \in V$ such that for any demand function this set of paths performs well.

Problem 1 (Oblivious Routing). Given an undirected Graph $G = (V, E)$ and an edge capacity function $c : E \to \mathbb{R}^+$. Calculate a convex combination of paths for each $(u, v) \in V^2$ such that for any demand function the congestion of the resulting flow will be as small as possible.

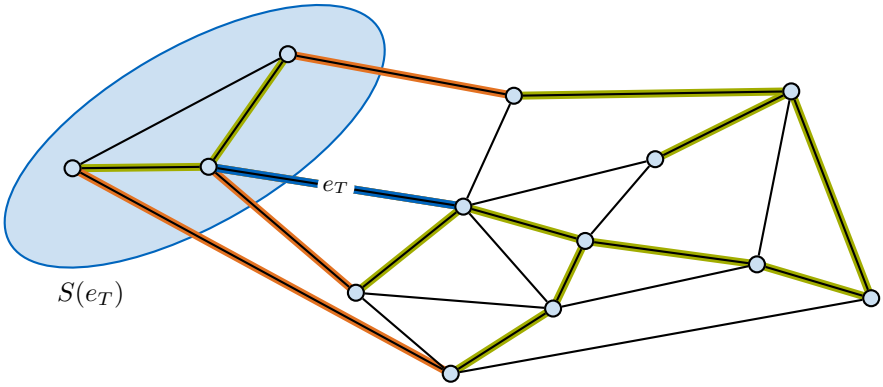

Fig. 2. Removing the edge e_T from the green spanning tree introduces a tree split. One of the two vertex sets we denote as $S(e_T)$ and define the capacity of the split as the sum of all capacities of edges crossing the boundary of $S(e_T)$, the blue and orange edges.

1.2 Formulation as a linear program

To find the desired approximation algorithm, we must first develop some lower bound for the optimal solution. A simple idea to create a valid (but probably not optimal) solution is to find any spanning tree $T = (V, E_T)$ of G and route any demand along its edges.

Since it is a spanning tree, there is a unique path from any vertex u to any vertex v in T which will be used to satisfy the complete demand. For any edge $e_T \in E_T$ the resulting flow f_{e_T} will be the sum of all demands between two nodes connected by this tree edge. These are exactly the nodes in different connected components created by removing the edge e_T.

Definition 3 (Tree Split). *Given a tree $T = (V, E_T)$ and an edge $e_T \in E_T$. Removing e_T from T splits it into two connected components, one of which we call $S(e_T)$ and the other one being $V \setminus S(e_T)$.*

We call $C(e_T)$ the capacity and $D(e_T)$ the demand of this split given by the sum of all capacities and demands connecting nodes in different connected components.

$$C(e_T) = \sum_{\substack{u \in S(e_T), \\ v \notin S(e_T)}} c_{uv}$$

$$D(e_T) = \sum_{\substack{u \in S(e_T), \\ v \notin S(e_T)}} c_{uv}$$

This definition illustrated by Fig. 2 allows us to write the flow generated by a simple solution to oblivious routing using a spanning tree T as $f_{e_T} = D(e_T)$ for all $e_T \in T$ and 0 otherwise. We now observe that for a given demand function any solution to the resulting multi-commodity flow problem will be bounded below by all tree splits.

Lemma 1. *For any tree T and any tree edge e_T, any routing in G must contain an edge e with congestion*

$$\rho_e \geq \frac{D(e_T)}{C(e_T)}.$$

Therefore, the optimal congestion ρ^* of the flow problem can be no better.

We will use this observation to proof our approximation guarantee. Suppose we find a spanning tree such that every edge has capacity of the capacity of the corresponding tree split divided by some factor $\alpha \geq 1$.

$$\forall e_T \in E_T. \quad c_{e_T} \geq \frac{1}{\alpha} C(e_T)$$

Using Lemma 1 we can bound the congestion of this spanning tree in relation to the optimal solution.

$$\begin{aligned} \rho_T &= \max_{e_T} \frac{D(e_T)}{c_{e_T}} \\ &\leq \alpha \max_{e_T} \frac{D(e_T)}{C(e_T)} \\ &\leq \alpha \rho^* \end{aligned}$$

Note that this factor α is a property of the tree and not any specific demand function, thus we know that this tree yields a solution of cost at most α times the optimal solution for any demand.

This tree however does not have to exist and therefore this approach will not yield any approximation guarantee. Since oblivious routing allows us to define multiple u-v-paths, a natural extension is to consider not only one spanning tree but a convex combination of multiple spanning trees. We denote each of these trees as some $T_i = (V, E_{T_i})$ and identify $e \in E_{T_i}$ with $e \in T_i$ for compactness. Solutions of this kind are a set of trees and factors $\{(T_i, \lambda_i)\}$ with $\lambda_i \geq 0$ and $\sum_i \lambda_i = 1$.

For a given demand function, the demand d_{uv} is routed through the unique u-v-paths in the trees T_i according to the convex fractions. For every edge e in the original graph we get

$$f_e = \sum_{\substack{i: \\ e \in T_i}} \lambda_i D_i(e)$$

where we denote the demand of the split introduced by T_i using some edge $e \in T_i$ as $D_i(e)$.

This solution scheme is still too strict though and we want to allow more sophisticated structures. Instead of using the spanning trees directly, we interpret every node on a u-v-path of T_i as an intermediate point on a path from u to v. Instead of using the direct connections given by the tree edges to traverse the intermediate points, we are allowed to take any path in G. A pair of spanning tree and set of paths connecting two neighboured nodes in the tree is called a path tree.

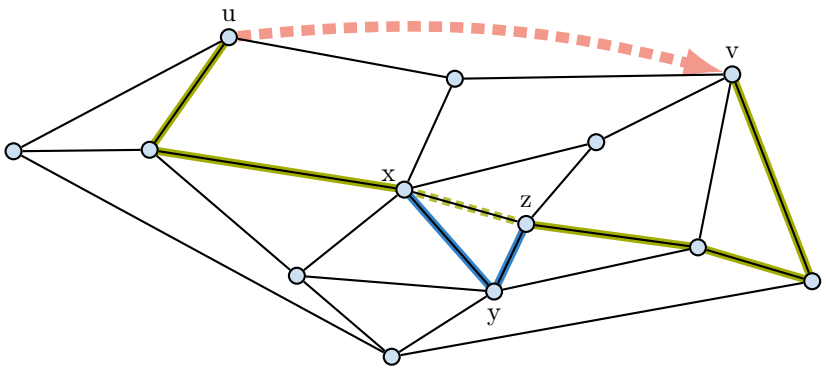

Fig. 3. A path tree is a pair of a spanning tree T and a mapping P from its edges to arbitrary paths in G. Instead of routing the demand d_{uv} along the green tree edges, the edge (x, z) is replaced by the blue edges $P((x, z)) = ((x, y), (y, z))$.

Definition 4 (Path Tree). *A path tree of an undirected graph $G = (V, E)$ is a pair (T, P) of a spanning tree T of G and a function $P : E_T \to E$ identifying every edge $e_T = (x, y)$ of T with a path in G from x to y.*

Note that the same edge in G may well be used by multiple different paths in the same path tree. To now route any demand d_{uv} using a path tree, we move along the unique u-v-path in the tree and stitch together the paths corresponding to the edges in the path.

Since if routing with spanning tree T_i the flow through every tree edge e_T would be $D_i(T_i)$, the same amount of flow must now be added to every edge on the path $P_i(e_T)$. If we now consider a convex combination of path trees, the flow through every edge $e \in E$ is

$$f_e = \sum_i \lambda_i \sum_{\substack{e_T \in T_i: \\ e \in P_i(e_T)}} D_i(e_T).$$

Suppose we find a set of path trees such that

$$\forall e \in E. \quad c_e \geq \frac{1}{\alpha} \sum_i \lambda_i \sum_{\substack{e_T \in T_i: \\ e \in P_i(e_T)}} C_i(e_T)$$

we can again bound its congestion by the optimal solution using Lemma 1.

$$\begin{aligned} \rho = & \max_e \frac{f_e}{c_e} \\ & \leq \alpha \max_e \frac{\sum_i \lambda_i \sum_{\substack{e_T \in T_i: \\ e \in P_i(e_T)}} D_i(e_T)}{\sum_i \lambda_i \sum_{\substack{e_T \in T_i: \\ e \in P_i(e_T)}} C_i(e_T)} \\ & \leq \alpha \max_e \max_i \frac{D_i(e)}{C_i(e)} \leq \alpha \rho^* \end{aligned}$$

We will now show that there always is a solution using path trees such that $\alpha \in \mathcal{O}(\log n)$ and sketch how to find such a set of trees. We denote the finite but exponentially sized set of all path trees over G as $\mathcal{I}$. We can then formulate a linear program which enforces the assumption that every edge capacity is large enough and chooses a set of path trees such that α is minimal.

$$\begin{aligned} \min_{\alpha,\lambda} \quad & \alpha \\ \text{s. t.} \quad & \sum_{i\in\mathcal{I}} \lambda_i \sum_{\substack{e_T\in T_i:\\ (u,v)\in P_i(e_T)}} C_i(e_T) \le \alpha c_{uv} \qquad \forall u,v \in V \\ & \sum_{i\in\mathcal{I}} \lambda_i = 1 \\ & \lambda \ge 0 \end{aligned} \tag{1}$$

To gain an $\mathcal{O}(\log n)$ approximation algorithm from this linear program, we have to show that α is of at most logarithmic size and that it is possible to solve this linear program with exponentially sized sums in polynomial time, yielding a solution of polynomially many path trees.

1.3 Approximation guarantee using the dual

To proof the bound for the value of the linear program, we will consider the dual program and show that it is logarithmically bounded. The dual problem has exponentially many constraints, one for each path tree, and a decision variable $\ell_{uv} \in \mathcal{L}$ for every pair of vertices.

$$\begin{aligned} \max_{z,\mathcal{L}} \quad & z \\ \text{s. t.} \quad & \sum_{u,v\in V} c_{uv}\ell_{uv} = 1 \\ & z \le \sum_{e_T\in T_i} C_i(e_T) \sum_{(u,v)\in P_i(e_T)} \ell_{uv} \qquad \forall i \in \mathcal{I} \\ & \mathcal{L} \ge 0 \end{aligned} \tag{2}$$

We will rewrite this dual program to be able to apply a result about the approximation of metrics with tree metrics. To this end we will interpret the variables ℓ_{uv} as distances between vertices and obtain a shortest path metric $d_\ell(u,v)$ which we will approximate later. For an edge $e = (x,y)$ we denote $d_\ell(e) \coloneqq d_\ell(x,y)$.

We first observe that in Eq. (2) for any given i, the length of $P_i(e_T)$ is always at least $d_\ell(e_T)$ and since all constraints bound z above and there is some tree choosing the shortest path, we can replace the constraints by

$$z \le \sum_{e_T\in T_i} C_i(e_T) d_\ell(e_T) \qquad \forall i \in \mathcal{I}.$$

We can further reduce the constraints to the smallest constraint, again because we are bounding z above.

$$z \leq \min_{i \in \mathcal{I}} \sum_{e_T \in T_i} C_i(e_T) d_\ell(e_T)$$

Since we are only left with one constraint concerning z and it is not otherwise needed in the dual program we can move it into the objective function and obtain a smaller LP. While it is no longer of exponential size, it might still take exponential time to find the minimum.

$$\begin{aligned} \max_{\mathcal{L}} \quad & \min_{i \in \mathcal{I}} \sum_{e_T \in T_i} C_i(e_T) d_\ell(e_T) \\ \text{s.t.} \quad & \sum_{u,v \in V} c_{uv} \ell_{uv} = 1 \\ & \mathcal{L} \geq 0 \end{aligned}$$

While the nonnegativity of the distances is actually desired, we now want to show that the last remaining constraint does not in fact reduce the solution space. Suppose $\sum_{u,v \in V} c_{uv} \ell_{uv} = \beta > 0$, the products of of capacities and lengths sum up to an arbitrary positive value. We can then transform the set of lengths $\mathcal{L}$ to a feasible solution of the dual problem by scaling every length by $\frac{1}{\beta}$. This, however, changes the objective function by the same factor. If we drop the constraint enforcing $\beta = 1$, we have to divide the objective function by β. This yields a rather compact representation of the dual program.

$$\begin{aligned} \max_{\mathcal{L}} \quad & \min_{i \in \mathcal{I}} \frac{\sum_{e_T \in T_i} C_i(e_T) d_\ell(e_T)}{\sum_{u,v \in V} c_{uv} \ell_{uv}} \\ \text{s.t.} \quad & \mathcal{L} \geq 0 \end{aligned} \tag{3}$$

To proof the desired approximation guarantee we now approximate d_ℓ using a result obtained about tree metrics from [WS11].

Theorem 1 (Tree Metric). *For any nonnegative set of costs c_{uv} and any metric d_ℓ there exists a tree metric (V, M) such that*

$$d_\ell(u, v) \leq M_{uv} \qquad \forall u, v \in V$$

$$\sum_{u,v \in V} c_{uv} M_{uv} \leq \mathcal{O}(\log n) \sum_{u,v \in V} c_{uv} d_\ell(u, v).$$

To be able to apply this result to our dual, we first need to show one more lemma. We need to rewrite the enumerator in Eq. (3) to not sum over tree edges but all edges in the graph.

Lemma 2. *Let T be a spanning tree and (V, M) a tree metric of $G = (V, E)$. Then it holds that*

$$\sum_{(x,y) \in E_T} C(x, y) M_{xy} = \sum_{(u,v) \in E} c_{uv} M_{uv}.$$

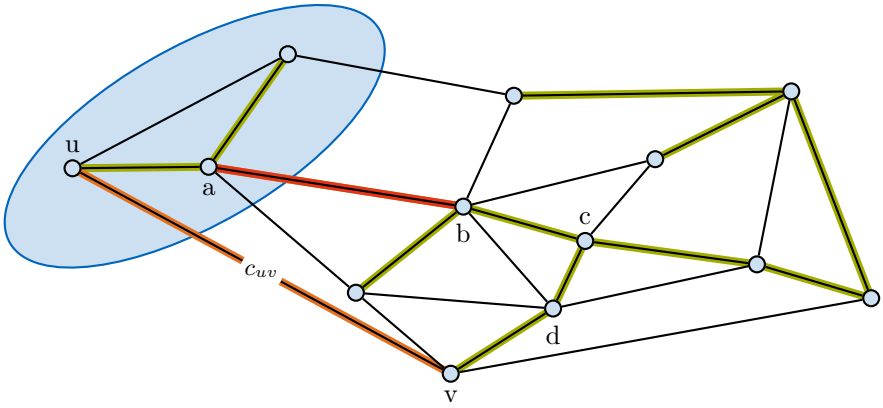

Fig. 4. The left hand side contains a product of capacity c_{uv} and distance M_{ab} iff c_{uv} is part of the tree split introduced by the tree edge (a, b). A capacity is part of a tree split iff u and v are not in the same connected component in the split. This is the case iff the tree edge lies on the path between u and v. The sum of all tree edges between u and v is M_{uv} by definition, yielding the right hand side of the sum.

Proof. See Fig. 4. □

Combining all observations and lemmas now bounds the value of both linear programs logarithmically, directly following from the logarithmic bound of the tree metrics.

Theorem 2. *The primal and dual linear program have a value of* $\mathcal{O}(\log n)$.

Proof. We proof the value of the dual program, the value of the primal program then follows by strong duality.

For any lengths $\mathcal{L}$ we know that for the minimizing tree in Eq. (3) we have the following chain of inequalities, using the definition of tree metrics, the previous lemma and the observation that the distance $d_\ell(u, v)$ is never larger than ℓ_{uv} since d_ℓ is a shortest path metric.

$$\begin{aligned}\sum_{e_T \in T_i} C_i(e_T) d_\ell(e_T) &\leq \sum_{e_T \in T_i} C_i(e_T) M_{e_T} \\ &= \sum_{u,v \in V} c_{uv} M_{uv} \\ &\leq \mathcal{O}(\log n) \sum_{u,v \in V} c_{uv} d_\ell(u, v) \\ &\leq \mathcal{O}(\log n) \sum_{u,v \in V} c_{uv} \ell_{uv}\end{aligned}$$

Dividing by the sum directly gives the value guarantee:

$$\frac{\sum_{e_T \in T_i} C_i(e_T) d_\ell(e_T)}{\sum_{u,v \in V} c_{uv} \ell_{uv}} \leq \mathcal{O}(\log n)$$

□

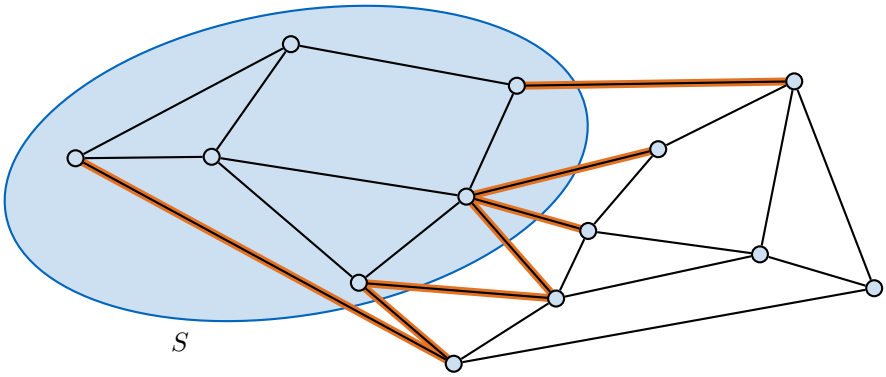

Fig. 5. The cost $c(\delta(S))$ of the bisection S is defined as the sum of the nonnegative costs of all orange edges leaving the set S.

While this proofs the existence of a solution of value $\mathcal{O}(\log n)$, it still remains to be shown that polynomially many trees are enough and the solution can actually be found in polynomial time. While this is done rigorously in [WS11], the idea is to rewrite the linear program in such a way to not actually search for the minimal α but enforce an $\alpha \in \mathcal{O}(\log n)$. This results in a linear program which can be solved using the ellipsoid method in polynomial time resulting in a set of polynomially many path trees. This then proofs the existence of the desired $\mathcal{O}(\log n)$ approximation algorithm.

2 Minimum Bisection

The knowledge about the existence of an $\mathcal{O}(\log n)$ approximation algorithm for the oblivious routing problem can be applied to find approximation algorithms for a number of other problems, some of which are described in [Räc08]. As an example, we will now give a definition of the minimum bisection problem and show that it can be approximated using oblivious flow.

Minimum bisection has a similar structure to oblivious flow as it is also defined over undirected graphs with a function mapping the edges to non negative numbers. Instead of interpreting these numbers as a capacity, they are now understood as costs. We want to find a set containing half of the vertices such that the sum of all costs of edges leaving this set is minimal.

Problem 2 (Minimum Bisection). Given an undirected Graph $G = (V, E)$ and an edge-cost function $c : E \to \mathbb{R}_0^+$. Find a set $S \subset V$ containing half the vertices with minimal split cost $c(\delta(S))$.

$$\delta(S) \coloneqq \{(x, y) \in E \mid x \in S \wedge y \notin S\}$$
$$c(\delta(S)) \coloneqq \sum_{\substack{e \in E: \\ e \in \delta(S)}} c_e$$

We will now show that the set of path trees obtained by solving the oblivious flow problem on G with capacity function c contains a tree that defines a bisection of G which costs logarithmically more than an optimal solution. We call this bisection a minimum tree bisection. It is obtained by reweighing the edges of the tree to cost as much as the tree splits introduced by them and then solving the minimum bisection problem on those trees. This can be solved in polynomial time with a straightforward dynamic programming approach described in [WS11].

Definition 5 (Minimum Tree Bisection). *Given a spanning tree T of G with an edge $e_T \in E_T$. We define a new cost function $c_T : E_T \to \mathbb{R}_0^+$ based on the tree splits induced by the edges of T. The cost of a bisection S is the sum of tree split costs of all tree edges cut by S.*

$$c_T(e_T) = C(e_T)$$
$$c_T(\delta(S)) = \sum_{\substack{e_T \in E_T: \\ e_T \in \delta(S)}} C(e_T)$$

The minimum tree bisection X of T is an optimal solution of the minimum bisection problem with respect to the cost function c_T.

Assuming we can find minimum tree bisections in polynomial time, the following algorithm describes an $\mathcal{O}(\log n)$ approximation algorithm for the minimum bisection problem. While the polynomial running time is clear, it remains to show that the obtained solution actually satisfies the approximation guarantee.

Algorithm 1. *Given graph $G = (V, E)$ and cost function $c : E \to \mathbb{R}_0^+$.*

1. *Interpret costs $c(e)$ as capacities*
2. *Solve oblivious routing on G, obtaining trees T_i*
3. *Find minimum tree bisections X_i for all trees T_i*
4. *Choose the X_i with lowest $c(\delta(X_i))$*

To proof the guarantee we will need two lemmas connecting the cost of a split in the original graph G with the costs in a single spanning tree T relative to its cost function c_T and a convex combination of trees.

Lemma 3. *Let $\{(T_i, \lambda_i)\}$ be an $\mathcal{O}(\log n)$-approximation to the oblivious flow problem. Then for any cut $S \subseteq V$ the convex combination of tree bisections is bounded above by the cost of S times a logarithmic factor.*

$$\sum_i \lambda_i c_{T_i}(\delta(S)) \leq \mathcal{O}(\log n) c(\delta(S))$$

Proof. We remember the first constraint of the primal program in Eq. (1). Since we have already proven that $\alpha \in \mathcal{O}(\log n)$ we can replace it by this bound and sum up the inequalities for all edges in $\delta(S)$.

$$\sum_i \lambda_i \sum_{(u,v) \in \delta(S)} \sum_{\substack{e_T \in T_i: \\ (u,v) \in P_i(e_T)}} C_i(e_T) \leq \mathcal{O}(\log n) c(\delta(S))$$

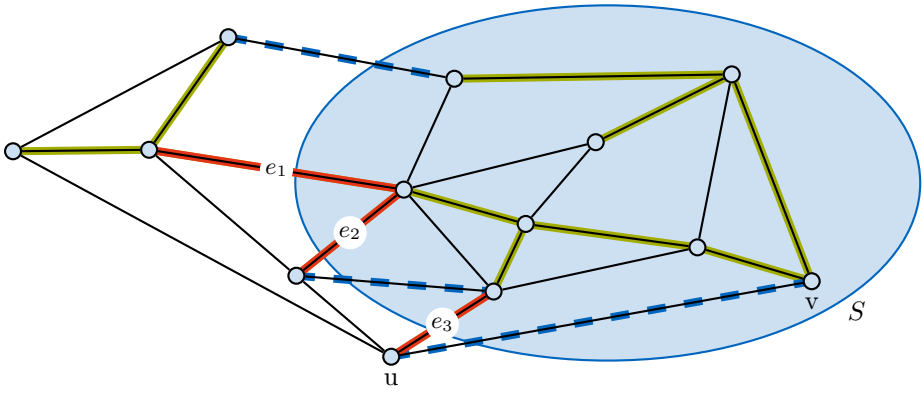

Fig. 6. The cost of S in the original graph is the sum of all red and blue dashed edge costs. While the red edges are tree edges and thus contained in the tree cost function, the blue dashed edge connecting u and v is contained in the tree split introduced by edge e_3 on the tree path between u and v.

The right hand side then sums up to the desired term, while we have to rewrite the left hand side a bit.

$$c_{T_i}(\delta(S)) = \sum_{\substack{e_T \in E_{T_i}: \\ e_T \in \delta(S)}} C_i(e_T) \leq \sum_{(u,v)\in\delta(S)} \sum_{\substack{e_T \in T_i: \\ (u,v)\in P_i(e_T)}} C_i(e_T)$$

While the equality holds by definition, we observe that for every tree edge e_T contained in $\delta(S)$ there must be an edge in $P_i(e_T)$ which crosses the boundary of S since e_T starts in S and ends outside of S. Therefore, all summands in the left side are contained in the right side, the inequality holds. If we apply this for all T_i, we have proven the lemma. □

While we can bound the convex combination of minimum tree bisections obtained by the oblivious flow approximation by the original cost function, we can also show that the original cost is always smaller than the cost of the same set relative to some tree cost function.

Lemma 4. *For any spanning tree T and any cut $S \subseteq V$ the cost of the cut is bounded above by the tree bisection of T.*

$$c(\delta(S)) \leq c_T(\delta(S))$$

Proof. An edge (u, v) is contained in $\delta(S)$ iff it starts in S and ends outside of S. Since there is an unique path from u to v in T there must be an edge e_T on this path that is also contained in $\delta(S)$. But then (u, v) must be contained in the tree split introduced by e_T and is therefore contained in the right hand side costs by definition. See Fig. 6 for an illustration of this proof. □

These lemmas allow us to quickly prove the approximation guarantee of the algorithm.

Theorem 3. *Algorithm 1 is an $\mathcal{O}(\log n)$ approximation algorithm for the minimum bisection problem.*

Proof. Let X^* be an optimal minimum bisection of G and $\{(T_i, \lambda_i)\}$ be a set of trees obtained from the oblivious flow algorithm with minimum tree bisections X_i. We consider the convex combination of the costs of all minimum tree bisections.

$$\begin{aligned}\sum_i \lambda_i c(\delta(X_i)) &\leq \sum_i \lambda_i c_{T_i}(\delta(X_i)) \\ &\leq \sum_i \lambda_i c_{T_i}(\delta(X^*) \\ &\leq \mathcal{O}(\log n) c(\delta(X^*))\end{aligned}$$

Using Lemma 4 we know that every single tree bisection X_i can only cost more relative to its respective cost function c_{T_i}. But since the X_i are optimal solutions for the trees T_i, the global optimal solution X^* cannot be better. We now apply Lemma 3 to show that the original convex combination of bisections is bounded by our desired bound. Since the combination of nonnegative costs is bounded, so must be the smallest cost or otherwise the inequality cannot hold. This proofs the correctness of the algorithm. □

References

ACF+03. Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Räcke. Optimal oblivious routing in polynomial time. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 383–388. ACM, 2003.

BKR03. 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*, pages 24–33. ACM, 2003.

CLRS01. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* The MIT Press, 2 edition, 2001.

Räc08. Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 255–264. ACM, 2008.

WS11. David P Williamson and David B Shmoys. *The design of approximation algorithms.* Cambridge University Press, 2011.