Finding the KT Partition of a Weighted Graph in Near-Linear Time

Simon Apers
CNRS and IRIF, Paris, France

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland

Troy Lee
Centre for Quantum Software and Information, University of Technology Sydney, Australia

Abstract

In a breakthrough work, Kawarabayashi and Thorup (J. ACM'19) gave a near-linear time deterministic algorithm to compute the weight of a minimum cut in a simple graph $G = (V, E)$. A key component of this algorithm is finding the $(1+\varepsilon)$-KT partition of G, the coarsest partition $\{P_1, \ldots, P_k\}$ of V such that for every non-trivial $(1+\varepsilon)$-near minimum cut with sides $\{S, \bar{S}\}$ it holds that P_i is contained in either S or $\bar{S}$, for $i = 1, \ldots, k$. In this work we give a near-linear time randomized algorithm to find the $(1+\varepsilon)$-KT partition of a *weighted* graph. Our algorithm is quite different from that of Kawarabayashi and Thorup and builds on Karger's framework of tree-respecting cuts (J. ACM'00).

We describe a number of applications of the algorithm. (i) The algorithm makes progress towards a more efficient algorithm for constructing the polygon representation of the set of near-minimum cuts in a graph. This is a generalization of the cactus representation, and was initially described by Benczúr (FOCS'95). (ii) We improve the time complexity of a recent quantum algorithm for minimum cut in a simple graph in the adjacency list model from $\widetilde{\mathcal{O}}(n^{3/2})$ to $\widetilde{\mathcal{O}}(\sqrt{mn})$, when the graph has n vertices and m edges. (iii) We describe a new type of randomized algorithm for minimum cut in simple graphs with complexity $\mathcal{O}(m + n\log^6 n)$. For graphs that are not too sparse, this matches the complexity of the current best $\mathcal{O}(m+n\log^2 n)$ algorithm which uses a different approach based on random contractions.

The key technical contribution of our work is the following. Given a weighted graph G with m edges and a spanning tree T of G, consider the graph H whose nodes are the edges of T, and where there is an edge between two nodes of H iff the corresponding 2-respecting cut of T is a non-trivial near-minimum cut of G. We give a $\mathcal{O}(m\log^4 n)$ time deterministic algorithm to compute a spanning forest of H.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graph theory

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2022.32

Category APPROX

Related Version *Full Version*: https://arxiv.org/abs/2111.01378 [1]

Funding *Paweł Gawrychowski*: Partially supported by the Bekker programme of the Polish National Agency for Academic Exchange (PPN/BEK/2020/1/00444).
Troy Lee: Supported in part by the Australian Research Council Grant No: DP200100950.

1 Introduction

Given a weighted and undirected graph G with n vertices and m edges[1], the minimum cut problem is to find the minimum weight $\lambda(G)$ of a set of edges whose removal disconnects G. When G is unweighted, this is simply the minimum number of edges whose removal disconnects G, also known as the edge connectivity of G. The minimum cut problem is a fundamental problem in theoretical computer science whose study goes back to at least the 1960s when the first polynomial time algorithm computing edge connectivity was given by Gomory and Hu [12]. In the current state-of-the-art, there are near-linear time randomized algorithms for the minimum cut problem in weighted graphs [9, 15, 21] and near-linear time *deterministic* algorithms in the case of simple graphs[2] [14, 18]. Very recently, Li [19] has given an almost-linear time (i.e. time $\mathcal{O}(m^{1+o(1)})$) deterministic algorithm for weighted graphs as well.

The best known algorithms for weighted graphs all rely on a framework developed by Karger [15] which, for an input graph G, relies on finding $\mathcal{O}(\log n)$ spanning trees of G such that with high probability one of these spanning trees will contain at most 2 edges from a minimum cut of G. In this case the cut is said to 2-respect the tree. A key insight of Karger is that, given a spanning tree T of G, the problem of finding a 2-respecting cut of T that has minimum weight in G can be solved deterministically in near-linear time, specifically time $\mathcal{O}(m \log^2 n)$. After standing for 20 years, the bound for this minimum-weight 2-respecting cut problem was recently improved by Gawrychowski, Mozes, and Weimann [9], who gave a deterministic $\mathcal{O}(m \log n)$ time algorithm, and independently by Mukhopadhyay and Nanongkai [21] who gave a randomized algorithm with complexity $\mathcal{O}(m \log n + n \log^4 n)$.

The best algorithms in the case of a simple graph G rely on a quite different approach, pioneered by Kawarabayashi and Thorup [18]. This approach begins by finding the minimum degree d of a vertex in G. Then the question becomes if there is a non-trivial cut, i.e. a cut where both sides of the corresponding bipartition have cardinality at least 2, whose weight is less than d. This problem is solved by finding what we call the $(1+\varepsilon)$-*KT partition* of the graph. Let $\mathcal{B}_\varepsilon^{nt}(G)$ be the set of all bipartitions $\{S, \bar{S}\}$ of the vertex set corresponding to non-trivial cuts whose weight is at most $(1+\varepsilon)\lambda(G)$. The $(1+\varepsilon)$-KT partition of G is the coarsest partition $\{P_1, \ldots, P_k\}$ of the vertex set such that for any $\{S, \bar{S}\} \in \mathcal{B}_\varepsilon^{nt}(G)$ it holds that P_i is contained in either S or $\bar{S}$, for each $i = 1, \ldots, k$. If one considers the multigraph G' formed from G by identifying vertices in the same set P_i, then G' preserves all non-trivial $(1+\varepsilon)$-near minimum cuts of G. Kawarabayashi and Thorup further show that for any $\varepsilon < 1$ the graph G' only has $\widetilde{\mathcal{O}}(n)$ edges. This bound crucially uses that the original graph is simple. The edge connectivity of G is thus the minimum of d and the edge connectivity of G'. One can use Gabow's deterministic $\mathcal{O}(\lambda m' \log n)$ edge connectivity algorithm [8] for a multigraph with m' edges and edge connectivity λ to check in time $\widetilde{\mathcal{O}}(nd \log n) = \widetilde{\mathcal{O}}(m)$ if the edge connectivity of G' is less than d and, if so, compute it. In the most technical part of their work, Kawarabayashi and Thorup give a deterministic algorithm to find the $(1+\varepsilon)$-KT partition of a simple graph G in time $\widetilde{\mathcal{O}}(m)$, giving an $\widetilde{\mathcal{O}}(m)$ time deterministic algorithm overall for edge connectivity. The key tool in their algorithm is the PageRank algorithm, which they use for finding low conductance cuts in the graph.

---

[1] Throughout this paper we will use n and m to denote the number of vertices and edges of the input graph.

[2] A simple graph is an unweighted graph with no self loops and at most one edge between any pair of vertices.

The KT partition has proven to be a very useful concept. Rubinstein, Schramm, and Weinberg [23] also go through the $(1+\varepsilon)$-KT partition to give a near-optimal $\widetilde{\mathcal{O}}(n)$ randomized query algorithm determining the edge connectivity of a simple graph in the *cut query* model. In the cut query model one can query a subset of the vertices S and receive in return the number of edges with exactly one endpoint in S. En route to their result, [23] also improved the bound on the number of inter-component edges in the $(1+\varepsilon)$-KT partition of a simple graph to $\mathcal{O}(n)$, for any $\varepsilon < 1$. In the case $\varepsilon = 0$ this was independently done by Lo, Schmidt, and Thorup [20]. The KT partition approach is also used in the current best *randomized* algorithm for edge connectivity, which runs in time $\mathcal{O}(\min\{m + n\log^2 n, m\log n\})$ [10].[3]

1.1 Main result

In this work we give the first near-linear time randomized algorithm to find the $(1+\varepsilon)$-KT partition of a *weighted* graph, for $0 \le \varepsilon \le 1/16$. An interesting aspect of our algorithm is that it uses Karger's 2-respecting cut framework to find the $(1+\varepsilon)$-KT partition, thereby combining the aforementioned major lines of work on the minimum cut problem.

We describe the result in more detail. Let $G = (V, E, w)$ be a weighted graph, where E is the set of edges and $w : E \to \mathbb{R}_+$ assigns a positive weight to each edge. For a set $S \subseteq V$ let $\Delta_G(S)$ be the set of all edges of G with exactly one endpoint in S. A *cut* of G is a set of edges of the form $\Delta_G(S)$ for some $\emptyset \neq S \subsetneq V$. We call S and $\bar{S}$ the shores of the cut. Let $w(\Delta_G(S)) = \sum_{e\in\Delta(S)} w(e)$. We use $\lambda(G) = \min_{\emptyset\neq S\subsetneq V} w(\Delta(S))$ for the minimum weight of a cut in G.

We will be interested in partitions of V and the partial order on partitions induced by *refinement*. For two partitions $\mathcal{X}, \mathcal{Y}$ of V we say that $\mathcal{X} \preceq \mathcal{Y}$ iff for every $X \in \mathcal{X}$ there is a $Y \in \mathcal{Y}$ with $X \subseteq Y$. In this case we say $\mathcal{X}$ is a *refinement* of $\mathcal{Y}$. The *meet* of two partitions $\mathcal{X}$ and $\mathcal{Y}$, denoted $\mathcal{X} \wedge \mathcal{Y}$, is the partition $\mathcal{Z}$ such that $\mathcal{Z} \preceq \mathcal{X}, \mathcal{Z} \preceq \mathcal{Y}$ and for any other partition $\mathcal{W}$ satisfying these two conditions $\mathcal{W} \preceq \mathcal{Z}$. In other words, $\mathcal{X} \wedge \mathcal{Y}$ is the greatest lower bound on $\mathcal{X}$ and $\mathcal{Y}$ under $\preceq$. Explicitly, $\mathcal{X} \wedge \mathcal{Y}$ is the partition consisting of all non-empty pairwise intersections between sets from $\mathcal{X}$ and $\mathcal{Y}$. For a set of partitions $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ we write $\bigwedge \mathcal{D} = \mathcal{D}_1 \wedge \cdots \wedge \mathcal{D}_K$.

For our applications we need to consider not only minimum cuts, but also near-minimum cuts. For $\varepsilon \ge 0$, let $\mathcal{B}_\varepsilon(G) = \{\{S, \bar{S}\} : w(\Delta_G(S)) \le (1+\varepsilon)\lambda(G)\}$ be the set of all bipartitions of V corresponding to $(1+\varepsilon)$-near minimum cuts. Let $\mathcal{B}^{nt}_\varepsilon(G) \subseteq \mathcal{B}_\varepsilon(G)$ be the set of all the non-trivial cuts in $\mathcal{B}_\varepsilon(G)$. The $(1+\varepsilon)$-KT partition of G is exactly $\bigwedge \mathcal{B}^{nt}_\varepsilon(G)$.

Both $\bigwedge \mathcal{B}_\varepsilon(G)$ and $\bigwedge \mathcal{B}^{nt}_\varepsilon(G)$ are important sets for understanding the structure of (near)-minimum cuts in a graph. Consider first $\bigwedge \mathcal{B}_0(G)$, the meet of the set of all bipartitions corresponding to minimum cuts. This set arises in the cactus decomposition of G [7], a compact representation of all minimum cuts of G. A cactus is a connected multigraph where every edge appears in exactly one cycle. The edge connectivity of a cactus is 2 and the minimum cuts are obtained by removing any two edges from the same cycle. A *cactus decomposition* of a graph G is a cactus H on $\mathcal{O}(n)$ vertices and a mapping $\phi : V(G) \to V(H)$ such that $\Delta_G(\phi^{-1}(S))$ is a mincut of G iff $\Delta_H(S)$ is a mincut of H. The mapping ϕ does not have to be injective, so multiple vertices of G can map to the same vertex of H. In this case, however, the cactus decomposition property means that all vertices in $\phi^{-1}(\{v\})$ must be on the same side of every minimum cut of G, for every $v \in V(H)$. Thus as v ranges over $V(H)$

[3] The bound quoted in [10] is $\mathcal{O}(m + n\log^3 n)$ but the improvement to Karger's algorithm by [9] reduces this to $\mathcal{O}(m + n\log^2 n)$.

the sets $\phi^{-1}(\{v\})$ give the elements of $\bigwedge \mathcal{B}_0(G)$ (note that $\phi^{-1}(\{v\})$ can also be empty). A cactus decomposition of a weighted graph can be constructed by a randomized algorithm in near-linear time [16], thus this also gives a near-linear time randomized algorithm to compute $\bigwedge \mathcal{B}_0(G)$.

Lo, Schmidt, and Thorup [20] give a version of the cactus decomposition that only represents the non-trivial minimum cuts. In fact, they give a deterministic $\mathcal{O}(n)$ time algorithm that converts a standard cactus into one representing the non-trivial minimum cuts. Combining this with the near-linear time algorithm to compute a cactus decomposition, this gives a near-linear time randomized algorithm to compute $\bigwedge \mathcal{B}_0^{nt}(G)$ as well.

The situation changes once we go to near-minimum cuts, which can no longer be represented by a cactus, but require the deformable polygon representation from [3–5]. This construction is fairly intricate, and the best known randomized algorithm to construct a deformable polygon representation of the $(1+\varepsilon)$-near mincuts of a graph builds on the Karger-Stein algorithm and takes time $\mathcal{O}(n^{2(1+\varepsilon)})$ [3, Section 6.3]. A prerequisite to constructing a deformable polygon representation is being able to compute $\bigwedge \mathcal{B}_\varepsilon(G)$ as, analogously to the case of a cactus, these sets will be the "atoms" that label regions of the polygons.

Our main result in this work is to give a randomized algorithm to compute $\bigwedge \mathcal{B}_\varepsilon(G)$ and $\bigwedge \mathcal{B}_\varepsilon^{nt}(G)$ in time $\mathcal{O}(m \log^5 n)$.

▶ **Theorem 1.** *Let $G = (V, E, w)$ be a graph with n vertices and m edges. For $0 \le \varepsilon \le 1/16$ let $\mathcal{B}_\varepsilon = \{\{S, \bar{S}\} : w(\Delta(S)) \le (1+\varepsilon)\lambda(G)\}$ and $\mathcal{B}_\varepsilon^{nt} \subseteq \mathcal{B}_\varepsilon$ be the subset of $\mathcal{B}_\varepsilon$ containing only non-trivial cuts. Both $\bigwedge \mathcal{B}_\varepsilon$ and $\bigwedge \mathcal{B}_\varepsilon^{nt}$ can be computed with high probability by a randomized algorithm with running time $\mathcal{O}(m \log^5 n)$.*

In the rest of this paper, we focus on computing $\bigwedge \mathcal{B}_\varepsilon^{nt}$. It is easy to construct $\bigwedge \mathcal{B}_\varepsilon$ from $\bigwedge \mathcal{B}_\varepsilon^{nt}$ deterministically in $\mathcal{O}(n)$ time (see full version [1]).

1.2 Applications

By building on our KT partition algorithm, we make progress on a number of problems.

1. The polygon representation is a compact representation of the set of near-minimum cuts of a weighted graph, originally described by Benczúr [3, 4] and Benczúr-Goemans [5]. It extends the cactus representation [7], which only works for the set of exact minimum cuts, and has played a key role in recent breakthroughs on the traveling salesperson problem [11, 17]. For a general weighted graph the polygon representation has size $O(n^2)$, and Benczúr has given a randomized algorithm to construct a polygon representation of the $(1+\varepsilon)$-near mincuts of a graph in time $\mathcal{O}(n^{2(1+\varepsilon)})$ [3, Section 6.3] by building on the Karger-Stein algorithm. It is an open question whether we can construct a polygon representation in time $\widetilde{\mathcal{O}}(n^2)$ for $\varepsilon > 0$. In his thesis [3, pg. 126], Benczúr says, "It already seems hard to directly identify the system of atoms within the $\widetilde{\mathcal{O}}(n^2)$ time bound," where the system of atoms is defined analogously to the $(1+\varepsilon)$-KT partition but for the set of all $(1+\varepsilon)$-near minimum cuts, not just the non-trivial ones. One can easily construct the set of atoms from a $(1+\varepsilon)$-KT partition, thus our KT partition algorithm gives a $\widetilde{\mathcal{O}}(m)$ time algorithm for this task as well, making progress on this open question.
2. The $(1+\varepsilon)$-KT partition of a weighted graph is exactly what is needed to give an optimal *quantum* algorithm for minimum cut: Apers and Lee [2] showed that the quantum query and time complexity of minimum cut in the adjacency matrix model is $\widetilde{\Theta}(n^{3/2}\sqrt{\tau})$ for a weighted graph where the ratio of the largest to smallest edge weights is τ, with the algorithm proceeding by finding a $(1+\varepsilon)$-KT partition.

In the case where the graph is instead represented as an adjacency list, they gave an algorithm with query complexity $\widetilde{\mathcal{O}}(\sqrt{mn\tau})$ but whose running time is larger at $\widetilde{\mathcal{O}}(\sqrt{mn\tau} + n^{3/2})$. The bottleneck in the time complexity is the time taken to find a $(1+\varepsilon)$-KT partition of a weighted graph with $\widetilde{\mathcal{O}}(n)$ edges. Using the near-linear time randomized algorithm we give to find a $(1+\varepsilon)$-KT partition here improves the time complexity of this algorithm to $\widetilde{\mathcal{O}}(\sqrt{mn\tau})$, matching the query complexity.
Both quantum algorithms also used the following observation [2, Lemma 2]: if in a weighted graph G the ratio of the largest edge weight to the smallest is τ, then the total weight of inter-component edges in a $(1+\varepsilon)$-KT partition of G for $\varepsilon < 1$ is $\mathcal{O}(\tau n)$, which can be tight.

3. The best randomized algorithm to compute the edge connectivity of a simple graph is the 2-out contraction approach of Ghaffari, Nowicki, and Thorup [10], which has running time $\mathcal{O}(\min\{m + n\log^2 n, m\log n\})$. Using our algorithm to find a $(1+\varepsilon)$-KT partition in a weighted graph we can follow Karger's 2-respecting tree approach to compute the edge connectivity of a simple graph in time $\mathcal{O}(m + n\log^6 n)$, thus also achieving the optimal bound on graphs that are not too sparse.

A detailed treatment of these applications is deferred to the full version [1]. Apart from these examples, we are hopeful that our near-optimal randomized algorithm for finding the KT partition of a weighted graph will find further applications.

2 KT partition algorithm

We now give a more detailed treatment of our algorithm to compute the KT partition $\bigwedge \mathcal{B}_\varepsilon^{nt}$. The first obstacle we face is that the number of near-minimum cuts in G can be $\Omega(n^2)$, so we cannot afford to consider all of them. An idea to get around this is to try the following:

1. Efficiently find a "small" subset $\mathcal{B}' \subseteq \mathcal{B}_\varepsilon^{nt}$ such that $\bigwedge \mathcal{B}' = \bigwedge \mathcal{B}_\varepsilon^{nt}$. We call such a subset a *generating set*.

A greedy argument shows that such a subset $\mathcal{B}'$ exists of size at most $n-1$. We initialize $\mathcal{B}' = \{\{S, \bar{S}\}\}$ for some element $\{S, \bar{S}\}$ in $\mathcal{B}_\varepsilon^{nt}$. We then iterate through the elements $\{T, \bar{T}\}$ of $\mathcal{B}_\varepsilon^{nt}$ and add it to $\mathcal{B}'$ iff $\bigwedge \mathcal{B}' \cup \{T, \bar{T}\} \neq \bigwedge \mathcal{B}'$. Each bipartition added to $\mathcal{B}'$ increases the number of elements in $\bigwedge \mathcal{B}'$ by at least 1. As this size can be at most n, and begins with size 2 the total number of sets at termination is at most $n-1$. This shows that a small generating set exists, but there still remains the problem of finding such a generating set *efficiently*.

Assuming we get past the first obstacle, there remains a second obstacle. The most straightforward algorithm to compute the meet of k partitions of a set of size n takes time $\Theta(kn\log n)$, which is again too slow if $k = \Theta(n)$. Thus we will also need to

2. Exploit the structure of $\mathcal{B}'$ to compute $\bigwedge \mathcal{B}'$ efficiently.

Apers and Lee [2] give an approach to accomplish (1) and (2) following Karger's framework of tree respecting cuts. Karger shows that in near-linear time one can compute a set of $K \in \mathcal{O}(\log n)$ spanning trees $T_1, \ldots, T_K$ of G such that every $(1+\varepsilon)$-near minimum cut of G 2-respects at least one of these trees. Let $\mathcal{B}_i \subseteq \mathcal{B}_\varepsilon^{nt}$ be the bipartitions corresponding to non-trivial near-minimum cuts that 2-respect T_i. To compute $\bigwedge \mathcal{B}_\varepsilon^{nt}$ it suffices to compute $\mathcal{C}_i = \bigwedge \mathcal{B}_i$ for each $i = 1, \ldots, K$ and then compute $\bigwedge_{i=1}^K \mathcal{C}_i$. The latter can be done in time $\mathcal{O}(n\log^2 n)$ by the aforementioned algorithm. This leaves the problem of computing $\bigwedge \mathcal{B}_i$.

A key observation from [2] gives a generating set $\mathcal{B}'_i$ for $\mathcal{B}_i$ of size $\mathcal{O}(n)$. It is constructed as follows. First we add bipartitions to $\mathcal{B}_i$ that correspond to near-minimum cuts that *1-respect* T_i. This is a set of size $\mathcal{O}(n)$, and Karger has shown that all near-minimum cuts that 1-respect a tree can be found in time $\mathcal{O}(m)$.

Next, we focus on the cuts that strictly *2-respect* T_i. There can be $O(n^2)$ such cuts. To handle them one creates a graph H whose nodes are the *edges* of T_i and where there is an edge between nodes e and f iff the 2-respecting cut of T_i defined by $\{e, f\}$ is a near-minimum cut in $\mathcal{B}_i$. Let F be a spanning forest of H. We then add to $\mathcal{B}'_i$ the 2-respecting cuts indexed by the edges in $E(F)$. The resulting set $\mathcal{B}'_i$ has size $\mathcal{O}(n)$ and it follows from [2, Lemma 29]that the resuling $\mathcal{B}'_i$ is a generating set for $\mathcal{B}_i$.

Apers and Lee give a *quantum* algorithm to find a spanning forest of H with running time $\widetilde{\mathcal{O}}(n^{3/2})$. They then give a randomized algorithm to compute $\bigwedge \mathcal{B}'_i$ in time $\widetilde{\mathcal{O}}(n)$. As our main technical contribution, we give a deterministic algorithm to find a spanning forest of H in time $\mathcal{O}(m \log^4 n)$. We also replace the randomization used in the algorithm to compute $\bigwedge \mathcal{B}'_i$ with an appropriate data structure to give an $\widetilde{\mathcal{O}}(n)$ deterministic algorithm to compute the meet. The details of this last procedure are deferred to the full version [1].

3 Spanning tree of near-minimum 2-respecting cuts in near-linear time

By the preceding discussion, we reduced the problem of constructing the KT partition to that of finding a spanning forest in a new graph H. This graph is derived from the original graph G and a given spanning forest T of G. Here we describe a deterministic near-linear time algorithm for this task, which is our main technical contribution.

Before describing the spanning forest algorithm, it is interesting to compare the problem of finding a spanning forest of H with the original problem solved by Karger of finding a minimum-weight 2-respecting cut of T. To find a spanning forest of H we potentially have to find $\Omega(n)$ many $(1+\varepsilon)$-near minimum cuts, which we accomplish with only an additional logarithmic overhead in the running time. The first insight to how this might be possible is to note that Karger's original algorithm to find the minimum weight 2-respecting cut actually does something stronger than needed. Let $\mathrm{cost}(e, f)$ be the weight of the 2-respecting cut of T defined by $\{e, f\}$. For *every* edge e of T Karger's algorithm attempts to find an $f^* \in \arg\min_f \mathrm{cost}(e, f)$. It does not always succeed in this task, but if the candidate f' returned for edge e is not such a minimizer, then for $f^* \in \arg\min_f \mathrm{cost}(e, f)$ it must be the case that the candidate g returned for f^* satisfies $\mathrm{cost}(f^*, g) \leq \mathrm{cost}(e, f^*)$. In this way, the algorithm still succeeds to find a minimum weight 2-respecting cut in the end.

In contrast, we give an algorithm that for every edge e of T actually finds

$$f^* \in \arg\min_f \{\mathrm{cost}(e, f) : \{e, f\} \text{ defines a non-trivial cut}\} \ .$$

We then show that this suffices to implement a round of Borůvka's spanning forest algorithm [22] on H in near-linear time. Borůvka's spanning forest algorithm consists of $\log n$ rounds and maintains the invariant of having a partition $\{S_1, \ldots, S_k\}$ of the vertex set and a spanning tree for each set S_i. The algorithm terminates when there is no outgoing edge from any set of the partition, at which point the collection of spanning trees for the sets of the partition is a spanning forest of H. The sets of the partition are initialized to be individual nodes of H.

In each round of Borůvka's algorithm the goal is to find an outgoing edge from each set S_i of the partition which is not already a connected component. Consider a node e of H with $e \in S_i$. We can find the best partner f for e and check if $\{e, f\}$ indeed gives rise to a non-trivial $(1+\varepsilon)$-near minimum cut and so is an edge of H. The problem is that f

could also be in S_i in which case the edge $\{e, f\}$ is not an outgoing edge of S_i as desired. To handle this, we maintain a data structure that allows us to find both the best partner f for e, but also the best partner f' for e that lies in a different set of the partition from f. We call this operation a *categorical top two* query. If there actually is an edge of H with one endpoint e and the other endpoint outside of S_i then either $\{e, f\}$ or $\{e, f'\}$ will be such an edge. Following the approach of [9] to the minimum-weight 2-respecting cut problem, combined with an efficient data structure for handling categorical top two queries, we are able to do this for all nodes e of H in near-linear time, which allows us to implement a round of Borůvka's algorithm in near-linear time.

3.1 Spanning tree of near-minimum 2-respecting cuts in near-linear time

We give a more precise description of the algorithm after settling on some notation. Let $G = (V, E, w)$ be a weighted undirected graph. We will assume throughout that G is connected, and in particular that $m \geq n-1$, as the KT partition of a disconnected graph can be easily determined from its connected components. Let T be a spanning tree of G. We will choose an $r \in V$ with degree 1 in T to be the root of T. We view T as a directed graph with all edges directed away from r. With some abuse of notation, we will also use T to refer to this directed version. If we remove any edge $e \in E(T)$ from T then T becomes disconnected into two components. We use $e^{\downarrow} \subseteq V$ to denote the set of vertices in the component not containing the root, and $T_e \subseteq E(T)$ to denote the set of edges in the subtree rooted at the head of e, i.e. the edges in the subgraph of T induced by $e^{\downarrow}$. We further use the shorthand $\text{cost}(e) = w(\Delta(e^{\downarrow}))$ for the weight of the cut with shore $e^{\downarrow}$.

Two edges $e, f \in E(T)$ define a unique cut in G which we denote by $\text{cut}_T(e, f)$ (or $\text{cut}(e, f)$ if it is clear from the context which T we are referring to). The cut depends on the relationship between e and f. If $e \in T_f$ or $f \in T_e$ then we say that e and f are *descendant* edges. Without loss of generality, say that $f \in T_e$. Then the cut defined by e and f is $\text{cut}(e, f) = \Delta(e^{\downarrow} \setminus f^{\downarrow})$. If e and f are not descendant edges, then we say they are *independent*. For independent edges we see that $\text{cut}(e, f) = \Delta(e^{\downarrow} \cup f^{\downarrow})$. In both cases we use $\text{cost}(e, f)$ to denote the weight of the corresponding cut.

In a KT partition we are only interested in non-trivial cuts. We first state the following simple claim that characterizes when $\text{cut}(e, f)$ is trivial, the proof of which is in the full version [1].

▶ **Proposition 2.** *Let $G = (V, E, w)$ be a connected graph with n vertices and let T be a spanning tree of G with root r. For $e, f \in E(T)$ if $\text{cut}(e, f)$ is trivial then*

1. *If e, f are independent then they must be the unique edges incident to the root.*
2. *If e, f are descendant then there is a vertex $v \in V$ such that e is the edge incoming to v and f is the unique edge outgoing from v, or vice versa.*

By choosing a root r for T that has degree 1 we avoid the case of item 1 of Proposition 2. Thus we only have to worry about trivial cuts when e, f are descendant.

With that out of the way, we now turn to our main theorem, which is the key routine in our $(1+\varepsilon)$-KT partition algorithm.

▶ **Theorem 3.** *Let $G = (V, E, w)$ be a connected graph with n vertices and m edges and let T be a spanning tree of G. For a given parameter β, define the graph H, with $V(H) = E(T)$ and $E(H) = \{\{e, f\} \in E(T)^{(2)} : \text{cost}(e, f) \leq \beta \text{ and } \text{cut}(e, f) \text{ non-trivial}\}$. There is a deterministic algorithm that given adjacency list access to G and T outputs a spanning forest of H in $\mathcal{O}(m \log^4 n)$ time.*

We prove Theorem 3 by following Borůvka's algorithm to find a spanning forest of H. Throughout the algorithm we maintain a subgraph F of H that is a forest, initialized to be the empty graph on vertex set $V(H) = E(T)$. At the end of the algorithm, F will be a spanning forest of H. The algorithm proceeds in rounds. In each round, for every tree in the forest, we find an edge connecting it to another tree in the forest, if such an edge exists. If H has k connected components, then in each round the number of trees in F minus k goes down by at least a factor of 2, and so the algorithm terminates in $\mathcal{O}(\log n)$ rounds.

The main work is implementing a round of Borůvka's algorithm. We will think of the nodes of F as having colors, where nodes in the same tree of the forest have the same color, and nodes in distinct trees have distinct colors. The goal of a single round is to find, for each color c, a pair of edges $e, f \in T$ such that $c = \text{color}(e) \neq \text{color}(f)$ and $\{e, f\} \in E(H)$, or detect that there is no such pair with these properties, in which case the nodes colored c in F already form a connected component of H. As we need to refer to such pairs often we make the following definition.

▶ **Definition 4** (partner). *Let T and H be as in Theorem 3. Given an assignment of colors to the edges of T we say that f is a partner for e if $\{e, f\} \in E(H)$ and $\text{color}(e) \neq \text{color}(f)$.*

We will actually do something stronger than what is required to implement a round of Borůvka's algorithm, which we encapsulate in the following code header.

Algorithm 1 RoundEdges.

Input: Adjacency list access to G, a spanning tree T of G, a parameter β, and an assignment of colors to each $e \in E(T)$.

Output: For every $e \in E(T)$ output a partner $f \in E(T)$, or report that no partner for e exists.

The implementation of RoundEdges is our main technical contribution. Let us first see how to use RoundEdges to find a spanning forest of H.

▶ **Lemma 5.** *Let G, T and H be as in Theorem 3. There is a deterministic algorithm that makes $\mathcal{O}(\log n)$ calls to* RoundEdges *and in $\mathcal{O}(n \log n)$ additional time outputs a spanning forest of H.*

Proof. We construct a spanning forest of H by maintaining a collection of trees F that will be updated in rounds by Borůvka's algorithm until it becomes a spanning forest. We initialize $F = (E(T), \emptyset)$ and give all $e \in E(T)$ distinct colors. We maintain the invariants that F is a forest and that nodes in the same tree have the same color and those in different trees have distinct colors.

Consider a generic round where F contains q trees. We call RoundEdges with the current color assignment. For every e which has one we obtain a partner f such that $\{e, f\} \in E(H)$ and $\text{color}(e) \neq \text{color}(f)$. For each color class c we select one e with $\text{color}(e) = c$ which has a returned partner (if it exists) and let X be the set of selected edges. We then find a maximal subset of edges $X' \subseteq X$ that do not create a cycle among the color classes by computing a spanning forest of the graph whose supervertices are given by the color classes and edges given by X. We add the edges in X' to $E(F)$. Finally we merge the color classes of the connected components in F by appropriately updating the color assignments, and we pass the updated forest and color assignments to the next round of the algorithm. Each of the steps in a single round can be executed in $\mathcal{O}(n)$ time.

We have that $|X'| \geq (q - \mathrm{cc}(H))/2$ where $\mathrm{cc}(H)$ is the number of connected components of H. Each edge from X' added to F decreases the number of trees in F by one. Thus $q - \mathrm{cc}(H)$ decreases by at least a factor of 2 in each round and the algorithm terminates after $\mathcal{O}(\log n)$ rounds. The time spent outside of the calls to RoundEdges is $\mathcal{O}(n)$ for each of the $\mathcal{O}(\log n)$ rounds. This is $\mathcal{O}(n \log n)$ overall. ◀

If a node e has a partner f, then $\{e, f\}$ can either be a pair of descendant or independent edges. To implement RoundEdges we will separately handle these cases, as described in the next two subsections.

3.2 Descendant edges

We follow the approach from [9] originally designed to find a single pair $\{e, f\}$ of descendant edges that minimizes $\mathrm{cost}(e, f)$ over all $e, f \in E(T)$ in $\mathcal{O}(m \log n)$ time. Their approach actually does something stronger (as does Karger's original algorithm): for every $e \in E(T)$ it finds the best match in the subtree T_e, i.e., it returns an edge $f^* \in \arg\min\{\mathrm{cost}(e, f) \mid f \in T_e\}$. In order to implement the descendant edge part of RoundEdges we have three additional complications to handle:

1. The edge f^* might have the same color as e.
2. The resulting $\mathrm{cut}(e, f^*)$ might be a trivial cut.
3. Edge e may have no partner in T_e but still have a partner f such that $e \in T_f$. This partnership may not be discovered when we are looking for partners of f if there is another $g \in T_f$ with $\mathrm{cost}(f, g) \leq \mathrm{cost}(e, f)$.

Item 1 can be easily solved by, in addition to finding f^*, also finding $g^* \in \arg\min\{\mathrm{cost}(e, f) \mid f \in E(T_e), \mathrm{color}(f) \neq \mathrm{color}(f^*)\}$. Phrasing things in this way, rather than simply looking for the edge h with color different from e which minimizes $\mathrm{cost}(e, h)$, helps to limit the dependence of the query on e and thus reduce the query time. If there is an $f \in T_e$ with $\mathrm{color}(f) \neq \mathrm{color}(e)$ and $\mathrm{cost}(e, f) \leq \beta$ then at least one of f^*, g^* will satisfy this too.

For item 2, we use the result of Proposition 2 that descendant edges that give rise to trivial cuts have a very constrained structure. This allow us to avoid trivial cuts when looking for a partner of e.

Item 3 is relatively subtle and does not arise in the minimum weight 2-respecting cut problem. To explain the issue we have to first say something about the high level structure of our implementation of RoundEdges. We will perform an Euler tour of T and, when the tour visits edge e for the first time, we will look for a partner f for e in T_e. The issue is the following, which we explain in the context of the very first round of Borůvka's algorithm so we do not have to worry about nodes having different colors. Suppose that in the graph H the only edge incident to node e is a node f with $e \in T_f$. Thus in the execution of RoundEdges we want to find f as a partner of e. When the Euler tour is at e we will not find any suitable partner for e, as there is none in T_e. We would like to identify f as a partner for e when the Euler tour visits f for the first time. However, if there is a $g \in T_f$ with $\mathrm{cost}(f, g) < \mathrm{cost}(f, e)$ then the algorithm will return g as a partner of f rather than e. To handle this we will actually make two passes over T. In the first pass, when we visit edge e for the first time we look for a partner f in T_e. In the second pass, we handle the case where the partner of e might be an ancestor of e. To do this we need to de-activate nodes. When the Euler tour visits f for the first time, we first find the lowest cost partner for f in T_f. We then de-activate this node, and again find the best active partner for f in T_f. Repeating this process, we will eventually find e if $\{e, f\}$ is indeed an edge of H and e, f have different colors.

Now we turn to more specific implementation details. A key idea in [9] is that we can do an Euler tour of T while maintaining a data structure such that when we first visit an edge e we can easily look up $\text{cost}(e, f)$ for any $f \in T_e$. The way this is maintained can be best understood by noting that for $f \in T_e$:

$$\begin{aligned}\text{cost}(e, f) &= w(\Delta(e^{\downarrow} \setminus f^{\downarrow})) \\ &= w(e^{\downarrow} \setminus f^{\downarrow}, (e^{\downarrow})^c) + w(e^{\downarrow} \setminus f^{\downarrow}, f^{\downarrow}) \\ &= \text{cost}(e) + \underbrace{\text{cost}(f) - 2w(f^{\downarrow}, (e^{\downarrow})^c)}_{\text{score}_e(f)} \ , \end{aligned} \tag{1}$$

where for convenience we defined $\text{score}_e(f) = \text{cost}(f) - 2w(f^{\downarrow}, (e^{\downarrow})^c)$, where the superscript c denotes taking the complement.

We begin the algorithm by computing $\text{cost}(e)$ for every $e \in E(T)$, which can be done in $\mathcal{O}(m)$ time [15]. We then do an Euler of T while maintaining a data structure such that, when we are considering $e \in E(T)$, for every $f \in T_e$ the value of the data structure at location f is $\text{cost}(e, f)$. For $f \notin T_e$ this will not in general be the case.

As can be seen from Equation (1), the key to maintaining this data structure is how to update the values $w(f^{\downarrow}, (e^{\downarrow})^c)$ when we descend edge e. Consider the case where we are currently at edge $e' = (z, x)$ and move to a descending edge $e = (x, y)$. For two vertices u, v let $p(u, v)$ be the set of edges on the path from u to v in T, and let $\text{lca}(u, v)$ be their lowest common ancestor in T. For $f \in T_e$ we see that

$$w(f^{\downarrow}, (e^{\downarrow})^c) = w(f^{\downarrow}, (e'^{\downarrow})^c) + \sum_{\substack{\{u,v\} \in E \\ f \in p(u,v), \text{lca}(u,v) = x}} w(\{u, v\}) \ . \tag{2}$$

By its definition in (1) we can compute $\text{score}_e(f)$ from $\text{score}_{e'}(f)$ by *subtracting* $2w(\{u, v\})$ from for every $\{u, v\} \in E$ such that $f \in p(u, v)$ and $\text{lca}(u, v) = x$. We implement this step for all f by looping over all $\{u, v\} \in E$ with $\text{lca}(u, v) = x$. After this update we have that $\text{cost}(e, f) = \text{cost}(e) + \text{score}(f)$ for every $f \in T_e$. This shows how to descend down T while keeping the invariant. The full tree is then explored by taking an Euler tour through T, and whenever we go back up in the tree we revert the score updates. This allows us to find candidate $f \in T_e$ for every $e \in E(T)$. To bound the number of updates, note that each of the m edges has a unique lca, and we only do an update corresponding to an edge when the lca is visited by the Euler tour. Since the Euler tour visits every vertex at most twice, the number of updates is at most $2m$. In addition, the number of categorical top two queries is $n - 1$.

The resulting algorithm is formalized in the full version [1], and it leads to the following theorem.

▶ **Theorem 6.** *Given an assignment* e.color *for each* $e \in E(T)$*, there is a deterministic algorithm that runs in time* $\mathcal{O}(m \log^2 n)$ *and for each* e *finds an* f *such that*

1. $\{e, f\} \in H$
2. $e \in T_f$ *or* $f \in T_e$
3. e.color $\neq f$.color

if such an f *exists.*

3.3 Independent edges

The goal now is to find, for every edge $e \in E(T)$, a partner $f \in E(T)$ such that e, f are independent, or decide that there is no such f. As we chose the root of T to have degree 1, by Proposition 2 we do not have to worry about trivial cuts in the independent edge case.

Instead of considering all edges $e \in E(T)$ one-by-one, we first find a so-called *heavy path decomposition* of T [13, 24], which is a partition of the edges of T into *heavy paths*. We define this partition recursively: first, find the heavy path starting at the root by repeatedly descending to the child of the current node with the largest subtree. This creates the topmost heavy path starting at the root (called its head) and terminating at a leaf (called its tail). Second, remove the topmost heavy path from T and repeat the reasoning on each of the obtained smaller trees. The crucial property is that, for any node u, the path from u to the root in T intersects at most $\log n$ heavy paths.

We can now iterate over all pairs of heavy paths h, h' to look for a partner $f \in h'$ for every $e \in h$. We cannot literally carry out this plan as the number of pairs of heavy paths can be $\Omega(n^2)$ and so we cannot explicitly consider every pair. We show next that many pairs h, h' result in a trivial case and that all these trivial pairs can be solved together in one batch. We then bound the number of non-trivial pairs and show that in near-linear time we can explicitly process all of them. The idea of processing pairs of heavy paths, and explicitly considering only the non-trivial ones, was introduced in the context of 2-respecting cuts by Mukhopadhyay and Nanongkai [21] (see also [9]).

Consider two distinct heavy paths h, h', where h is the path $u_1 - u_2 - \cdots - u_q$ and h' is the path $v_1 - v_2 - \ldots v_{q'}$. We let $e_i = (u_i, u_{i+1})$ for $i = 1, \ldots, q-1$ and $f_i = (v_i, v_{i+1})$ for $i = 1, \ldots, q'-1$. It can be that not all pairs e_i, f_j are independent, see Figure 1. However, we can easily identify the subpaths of h, h' containing pairwise independent edges in constant time by computing the lowest common ancestor v of the tails of h, h'. If $v = v_{p'}$ lies on h' then e_i, f_j will be independent for $1 \leq i < q$ and $p' \leq j < q'$, and similarly if v lies on h. In general we assume that p, p' have been determined so that e_i, f_j are independent for all $p' \leq i < q$ and $p' \leq j < q'$, and that these pairs comprise all of the independent pairs on h, h'. We can associate to h, h' a $(q-1)$-by-$(q'-1)$ matrix $M^{(h,h')}$ where for $p' \leq i < q$ and $p' \leq j < q'$

$$\begin{aligned} M^{(h,h')}[i,j] &= \text{cost}(e_i, f_j) \\ &= \text{cost}(e_i) + \text{cost}(f_j) - 2w(e_i^{\downarrow}, f_j^{\downarrow}) \ , \end{aligned} \tag{3}$$

and $M^{(h,h')}$ is undefined otherwise.[4] All values of $\text{cost}(e)$ can be computed in $\mathcal{O}(m)$ total time [15]. To efficiently evaluate $M^{(h,h')}$, we will prepare a list $L(h, h')$ of all edges that contribute to $w(e^{\downarrow}, f^{\downarrow})$ for independent e, f with $e \in h, f \in h'$. For many h, h' the list $L(h, h')$ will be empty, leading to the trivial case mentioned above. The following lemma bounds the size of all the non-empty lists and shows they can be constructed efficiently (proof in full version [1]).

▶ **Lemma 7.** *The total length of all lists $L(h, h')$ is $\mathcal{O}(m \log^2 n)$ and all non-empty lists $L(h, h')$ can be constructed deterministically in time $\mathcal{O}(m \log^2 n)$.*

We can now describe how to find a partner f for every e such that e, f are independent. The algorithm first solves together in one batch the case where the partner of $e \in h$ is in a heavy path h' where $L(h, h')$ is empty. After that we explicitly consider all h, h' with $L(h, h')$ non-empty. We consider these two cases in the next two subsections.

[4] We could restrict $M^{(h,h')}$ to the submatrix on which it is defined, but find it notationally easier for the i, j indices in $M^{(h,h')}$ to match the edge labels.

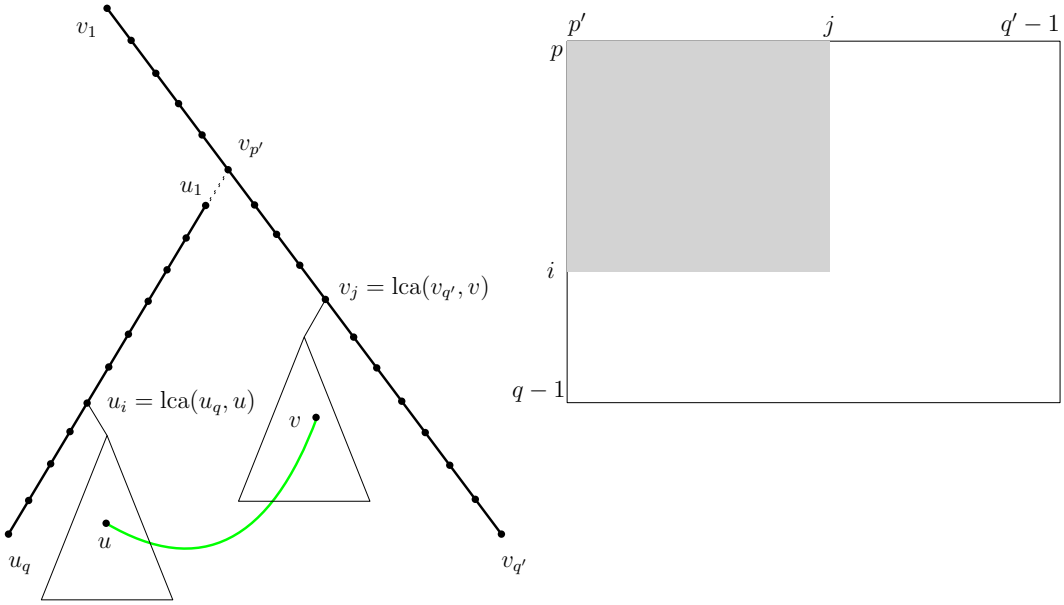

Figure 1 Contribution of an edge $\{u, v\} \in L(h, h')$ (denoted in green on the left) to $M^{(h,h')}[\cdot, \cdot]$ (denoted in grey on the right).

3.3.1 Empty lists

▶ **Lemma 8.** *There is a deterministic algorithm that in time $\mathcal{O}(m + n)$ finds a partner for every edge $e \in E(T)$ that has a partner f such that e, f are independent and $e \in h, f \in h'$ with $L(h, h')$ empty.*

Proof. The key observation is that if $L(h, h')$ is empty then $\mathrm{cost}(e, f) = \mathrm{cost}(e) + \mathrm{cost}(f)$ by Equation (3). As can be seen from Equation (1) and Equation (3), for any edge f' it always holds that $\mathrm{cost}(e, f') \leq \mathrm{cost}(e) + \mathrm{cost}(f')$, whether e, f' are descendant or independent. Thus in this case it suffices for us to find *any* f' of color different from e such that $\mathrm{cost}(e) + \mathrm{cost}(f') \leq \beta$, and $\mathrm{cut}(e, f')$ is non-trivial as this ensures $\mathrm{cost}(e, f') \leq \beta$. We are guaranteed such an f' exists as f satisfies this.

We can compute $\mathrm{cost}(f')$ for every $f' \in E(T)$ in time $\mathcal{O}(m)$ [15]. Then in time $\mathcal{O}(n)$ with one pass over $E(T)$ we compute the edge f_1 of lowest cost and the edge f_2 of lowest cost that is of color different to f_1. We then repeat this categorical top two query twice more, each time excluding all previously found edges. At the end we obtain edges $f_1, \ldots, f_6$. We claim that for every e, at least one of these must be a valid partner.

Consider any particular e. The first categorical top two query can only fail to find a valid partner for e if one of f_1, f_2 creates a trivial cut with e. In this case, the second categorical top two query can only fail if one of f_3, f_4 creates a trivial cut with e as well. By Proposition 2, however, there are at most two possible edges that can create a trivial cut with e, thus in this case the third categorical top two query must succeed and we find a valid partner for e. ◀

3.3.2 Non-empty lists

The more difficult case is to find partners among pairs h, h' with $L(h, h')$ non-empty. While we defer the proof details to the full paper [1], we describe the key insight of the algorithm, which relies on the special structure of $M^{(h,h')}$. As above, say that h is the path $u_1 - u_2 - \cdots - u_q$

and h' is the path $v_1 - v_2 - \dots v_{q'}$, and let $e_i = (u_i, u_{i+1})$ for $i = 1, \dots, q-1$ and $f_i = (v_i, v_{i+1})$ for $i = 1, \dots, q'-1$. Further suppose e_i, f_j are independent for all $p \le i < q, p' \le j < q'$. We have that $M^{(h,h')}[i,j] = \text{cost}(e_i) + \text{cost}(f_j) - 2w(e_i^{\downarrow}, f_j^{\downarrow})$ for $p \le i < q, p' \le j < q'$. Recall that $L(h,h')$ is defined precisely as the list of edges that contribute to $w(e^{\downarrow}, f'^{\downarrow})$ for independent $e \in h, f' \in h'$. The contribution of a specific edge $\{u,v\} \in L(h,h')$ can be understood as follows: let u_i be the lowest common ancestor of u and u_q, and v_j be the lowest common ancestor of v and $v_{q'}$. Then the weight of $\{u,v\}$ contributes to $M[a,b]$ for every $p \le a \le i$, $p' \le b \le j$. This is depicted in Figure 1. We can compute these indices i and j for every $\{u,v\} \in L(h,h')$. This takes constant time per edge using an appropriate LCA structure [6], and so total time $\mathcal{O}(|L(h,h')|)$.

Our algorithm crucially builds on this "sparse" representation of $M^{(h,h')}$ to efficiently transfer non-empty lists. The following lemma is proven in the full version [1].

▶ **Lemma 9.** *Let $\mathcal{F} = \{e \mid \exists h, h', f : e \in h, f \in h', e, f$ are partners and $L(h,h')$ non-empty$\}$. There is a deterministic algorithm to find a partner for every $e \in \mathcal{F}$ in time $\mathcal{O}(m \log^3 n)$.*

References

1. Simon Apers, Paweł Gawrychowski, and Troy Lee. Finding the KT partition of a weighted graph in near-linear time. *CoRR*, abs/2111.01378, 2021. `arXiv:2111.01378`.
2. Simon Apers and Troy Lee. Quantum complexity of minimum cut. In *Proceedings of the 36th Computational Complexity Conference (CCC '21)*, pages 28:1–28:3. LIPIcs, 2021.
3. András Benczúr. *Cut structures and randomized algorithms in edge-connectivity problems*. PhD thesis, MIT, 1997.
4. András A. Benczúr. A representation of cuts within 6/5 times the edge connectivity with applications. In *Proceedings of 36th Annual Symposium on Foundations of Computer Science (FOCS '95)*, pages 92–102. IEEE Computer Society, 1995. `doi:10.1109/SFCS.1995.492466`.
5. András A. Benczúr and Michel X. Goemans. Deformable polygon representation and near-mincuts. In Martin Grötschel, Gyula O. H. Katona, and Gábor Sági, editors, *Building Bridges: Between Mathematics and Computer Science*, pages 103–135. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. `doi:10.1007/978-3-540-85221-6_3`.
6. Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *Proceedings of 4th Latin American Symposium on Theoretical Informatics (LATIN '00)*, pages 88–94. Springer, 2000.
7. Efim A Dinitz, Alexander V Karzanov, and Michael V Lomonosov. On the structure of the system of minimum edge cuts in a graph. *Issledovaniya po Diskretnoi Optimizatsii (Studies in Discrete Optimization)*, pages 290–306, 1976. Appeared in Russian.
8. Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, 1995. `doi:10.1006/jcss.1995.1022`.
9. Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Minimum cut in $O(m \log^2 n)$ time. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP '20)*, volume 168 of *LIPIcs*, pages 57:1–57:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
10. Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA '20)*, pages 1260–1279. SIAM, 2020. `doi:10.1137/1.9781611975994.77`.
11. Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '11)*, pages 550–559. IEEE, 2011.

12 Ralph E. Gomory and Te C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. URL: http://www.jstor.org/stable/2098881.

13 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

14 Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. *SIAM Journal on Computing*, 49(1):1–36, 2020. doi:10.1137/18M1180335.

15 David R. Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, 2000. Announced at STOC 1996.

16 David R. Karger and Debmalya Panigrahi. A near-linear time algorithm for constructing a cactus representation of minimum cuts. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '09)*, pages 246–255. SIAM, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496798.

17 Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In *53rd Annual ACM-SIGACT Symposium on Theory of Computing (STOC '21)*, pages 32–45, 2021.

18 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *Journal of the ACM*, 66(1):4:1–4:50, 2019. Announced at STOC 2015. doi:10.1145/3274663.

19 Jason Li. Deterministic mincut in almost-linear time. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC '21)*, pages 384–395. ACM, 2021. doi:10.1145/3406325.3451114.

20 On-Hei S. Lo, Jens M. Schmidt, and Mikkel Thorup. Compact cactus representations of all non-trivial min-cuts. *Discrete Applied Mathematics*, 303:296–304, 2020. doi:10.1016/j.dam.2020.03.046.

21 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC '20)*, pages 496–509. ACM, 2020.

22 Jaroslav Nesetril, Eva Milková, and Helena Nesetrilová. Otakar Borůvka on the minimum spanning tree problem: Translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1-3):3–36, 2001. doi:10.1016/S0012-365X(00)00224-7.

23 Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS '18)*, pages 39:1–39:16. LIPIcs, 2018. doi:10.4230/LIPIcs.ITCS.2018.39.

24 Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.