2018 IEEE 59th Annual Symposium on Foundations of Computer Science

Faster Exact and Approximate Algorithms for k-cut

Anupam Gupta
Computer Science Department
CMU
Pittsburgh, USA
anupamg@cs.cmu.edu

Euiwoong Lee
Courant Institute of Mathematical Sciences
NYU
New York City, USA
euiwoong@cims.nyu.edu

Jason Li
Computer Science Department
CMU
Pittsburgh, USA
jmli@cs.cmu.edu

***Abstract*—In the k-cut problem, we are given an edge-weighted graph G and an integer k, and have to remove a set of edges with minimum total weight so that G has at least k connected components. The current best algorithms are an $O(n^{(2-o(1))k})$ randomized algorithm due to Karger and Stein, and an $\tilde{O}(n^{2k})$ deterministic algorithm due to Thorup. Moreover, several 2-approximation algorithms are known for the problem (due to Saran and Vazirani, Naor and Rabani, and Ravi and Sinha).**

It has remained an open problem to (a) improve the runtime of exact algorithms, and (b) to get better approximation algorithms. In this paper we show an $O(k^{O(k)}\, n^{(2\omega/3+o(1))k})$-time algorithm for k-CUT. Moreover, we show an $(1+\varepsilon)$-approximation algorithm that runs in time $O((k/\varepsilon)^{O(k)}\, n^{k+O(1)})$, and a 1.81-approximation in fixed-parameter time $O(2^{O(k^2)}\,\mathrm{poly}(n))$.

***Keywords*-minimum k-cut, graph algorithms, parameterized algorithm, approximation algorithms**

I. INTRODUCTION

In this paper we consider the k-CUT problem: given an edge-weighted graph $G = (V, E, w)$ and an integer k, delete a minimum-weight set of edges so that G has at least k connected components. This problem is a natural generalization of the global min-cut problem, where the goal is to break the graph into $k = 2$ pieces. This problem has been actively studied in theory of both exact and approximation algorithms, where each result brought new insights and tools on graph cuts.

It is not *a priori* clear how to obtain poly-time algorithms for any constant k, since guessing one vertex from each part only reduces the problem to the NP-hard MULTIWAY CUT problem. Indeed, the first result along these lines was the work of Goldschmidt and Hochbaum [1] who gave an $O(n^{(1/2-o(1))k^2})$-time exact algorithm for k-CUT. Since then, the exact exponent in terms of k has been actively studied. The current best runtime is achieved by an $\tilde{O}(n^{2(k-1)})$ randomized algorithm due to Karger and Stein [2], which performs random edge contractions until the remaining graph has k nodes, and shows that the resulting cut is optimal with probability at least $\Omega(n^{-2(k-1)})$. The asymptotic runtime of $\tilde{O}(n^{2(k-1)})$ was later matched by a deterministic algorithm of Thorup [3]. His algorithm was based on tree-packing theorems; it showed how to efficiently find a tree for which the optimal k-cut crosses it $2k-2$ times. Enumerating over all possible $2k-2$ edges of this tree gives the algorithm.

These elegant $O(n^{2k})$-time algorithms are the state-of-the-art, and it has remained an open question to improve on them. An easy observation is that the problem is closely related to k-CLIQUE, so we may not expect the exponent of n to go below $(\omega/3)k$. Given the interest in fine-grained analysis of algorithms, where in the range $[(\omega/3)k, 2k-2]$ does the correct answer lie?

Our main results give faster deterministic and randomized algorithms for the problem.

Theorem I.1 (Faster Randomized Algorithm)**.** *Let W be a positive integer. There is a randomized algorithm for k-CUT on graphs with edge weights in $[W]$ with runtime*

$$\widetilde{O}(k^{O(k)} n^{k+\lfloor (k-2)/3 \rfloor \omega + 1 + ((k-2) \bmod 3)} W) \approx O(k^{O(k)} n^{(1+\omega/3)k}),$$

that succeeds with probability $1 - 1/\mathrm{poly}(n)$.

Theorem I.2 (Even Faster Deterministic Algorithm)**.** *Let W be a positive integer. For any $\varepsilon > 0$, there is a deterministic algorithm for exact k-CUT on graphs with edge weights in $[W]$ with runtime*

$$k^{O(k)} n^{(2\omega/3+\varepsilon)k+O(1)} W \approx O(k^{O(k)} n^{(2\omega/3)k}).$$

In the above theorems, ω is the matrix multiplication constant, and $\widetilde{O}$ hides polylogarithmic terms. While the deterministic algorithm from Theorem I.2 is asymptotically faster, the randomized algorithm is better for small values of k. Indeed, using the current best value of $\omega < 2.373$ [4], Theorem I.1 gives a randomized algorithm for exact k-CUT on unweighted graphs which improves upon the previous best $\widetilde{O}(n^{2k-2})$-time algorithm of Karger and Stein for all $k \in [8, n^{o(1)}]$. For $k \le 6$, faster algorithms were given by Levine [5].

2575-8454/18/$31.00 ©2018 IEEE
DOI 10.1109/FOCS.2018.00020

Approximation algorithms: The k-CUT problem has also received significant attention from the approximation algorithms perspective. There are several $2(1-1/k)$-approximation algorithms that run in time $\mathrm{poly}(n,k)$ [6], [7], [8], which cannot be improved assuming the Small Set Expansion Hypothesis [9]. Recently, we gave an 1.9997-approximation algorithm that runs in $2^{O(k^6)}n^{O(1)}$ [10]. In this current paper, we give a $(1+\varepsilon)$-approximation algorithm for this problem much faster than the current best exact algorithms; prior to our work, nothing better was known for $(1+\varepsilon)$-approximation than for exact solutions.

Theorem I.3 (Approximation)**.** *For any $\varepsilon > 0$, there is a randomized (combinatorial) algorithm for k-CUT with runtime $(k/\varepsilon)^{O(k)}n^{k+O(1)}$ time on general graphs, that outputs a $(1+\varepsilon)$-approximate solution with probability $1-1/\mathrm{poly}(n)$.*

The techniques from the above theorem, combined with the previous ideas in [10], immediately give an improved FPT approximation guarantees for the k-CUT problem:

Theorem I.4 (FPT Approximation)**.** *There is a deterministic 1.81-approximation algorithm for the k-CUT problem that runs in time $2^{O(k^2)} \cdot n^{O(1)}$.*

Due to page restrictions in this extended abstract, we defer the proofs of Theorems I.3 and I.4 to the full version of this paper.

Limitations: Our exact algorithms raise the natural question: how fast can exact algorithms for k-CUT be? We give a simple reduction showing that while there is still room for improvement in the running time of exact algorithms, such improvements can only improve the constant in front of the k in the exponent, assuming a popular conjecture on algorithms for the CLIQUE problem.

Claim I.5 (Relationship to Clique)**.** *Any exact algorithm for the k-CUT problem for graphs with edge weights in $[n^2]$ can solve the k-CLIQUE problem in the same runtime. Hence, assuming k-CLIQUE cannot be solved in faster than $n^{\omega k/3}$ time, the same lower bound holds for the k-cut problem.*

A. Our Techniques

Our algorithms build on the approach pioneered by Thorup: using tree-packings, he showed how to find a tree T such that it crosses the optimal k-cut at most $2k-2$ times. (We call such a tree a *Thorup tree*, or *T-tree*.) Now brute-force search over which edges to delete from the T-tree (and how to combine the resulting parts together) gave an $\widetilde{O}(n^{2k-2})$-time deterministic algorithm. This last step, however, raises the natural question—*having found such a T-tree, can we use the structure of the k-CUT problem to beat brute force?* Our algorithms answer the question in the affirmative, in several different ways. The main ideas behind our algorithm are dynamic programming and fast matrix-multiplication, carefully combined with the fixed-parameter tractable algorithm technique of color-coding, and random sampling in general.

Fast matrix multiplication: Our idea to apply fast matrix multiplication starts with the crucial observation that if (i) the T-tree T is "tight" and crosses the optimal k-cut only $k-1$ times, and (ii) these edges are "incomparable" and do not lie on a root-leaf path, then the problem of finding these $k-1$ edges can be modeled as a max-weight clique-like problem! (And hence we can use matrix-multiplication ideas to speed up their computation.) An important property of this special case is that choosing an edge e to cut fixes one component in the k-CUT solution — by incomparability, the subtree below e cannot be cut anymore. The cost of a k-cut can be determined by the weight of edges between each pair of components (just like being a clique is determined by pairwise connectivity), so this case can be solved via an algorithm similar to k-CLIQUE.

Randomized algorithm: Our randomized algorithm removes these two assumptions step by step. First, while the above intuition crucially relies on assumption (ii), we give a more sophisticated dynamic program using color-coding schemes for the case where the edges are not incomparable. Moreover, to remove assumption (i), we show a randomized reduction that given a tree that crosses the optimal cut as many as $2k-2$ times, finds a "tight" tree with only $k-1$ crossings (which is the least possible), at the expense of a runtime of $O(k^2n)^{k-1}$. Note that guessing which edges to delete is easily done in n^{k-1} time, but adding edges to regain connectivity while not increasing the number of crossings can naively take a factor of m^{k-1} more time. We lose only a $k^{2(k-1)}$ factor using our random-sampling based algorithm, using that in an optimal k-CUT a split cluster should have more edges going to its own parts than to other clusters.

Deterministic algorithm: The deterministic algorithm proceeds along a different direction and removes both assumptions (i) and (ii) at once. We show that by deleting some $O(\log k)$ carefully chosen edges from the T-tree T, we can break it into three forests such that we only need to delete about $2k/3$ edges from each of these forests. Such a deletion is not possible when T is a star, but appropriately extending T by introducing Steiner nodes admits this deletion. (And $\Theta(\log k)$ is tight in this extension.) For each forest, there are $n^{2k/3}$ ways

to cut these edges, and once a choice of $2k/3$ edges is made, the forest will not be cut anymore. This property allows us to bypass (ii) and establish desired pairwise relationships between choices to delete $2k/3$ edges in two forests. Indeed, we set up a tripartite graph where one part corresponds to the choices of which $\leq 2k/3$ edges to cut in one forest and the cost of the min k-cut is the weight of the min-weight triangle, which we find efficiently using fast matrix multiplication. Some technical challenges arise because we need to some components for some forests may only have Steiner vertices, but we overcome these problems using color-coding.

Approximation schemes: The $(1+\varepsilon)$-approximation algorithm again uses the $O(k^2n)^{k-1}$-time randomized reduction, so that we have to cut exactly $k-1$ edges from a "tight" T-tree T. An exact dynamic program for this problem takes time $\approx n^k$ — as it should, since even this tight case captures clique, when T is a star and hence these $k-1$ edges are incomparable. And again, we need to handle the case where these $k-1$ edges are not incomparable. For the former problem, we replace the problem of finding cliques by approximately finding "partial vertex covers" instead. (In this new problem we find a set of $k-1$ vertices that minimize the total number of edges incident to them.) Secondly, in the DP we cannot afford to maintain the "boundary" of up to k edges explicitly any more. We show how to maintain an "ε-net" of nodes so that carefully "rounding" the DP table to only track a small $f(k)$-sized set of these rounded subproblems incurs only a $(1+\varepsilon)$-factor loss in quality.

Our approximate DP technique turns out to be useful to get a 1.81-approximation for k-CUT in FPT time, improving on our previous approximation of ≈ 1.9997 [10]. In particular, the *laminar cut problem* from [10] also has a tight T-tree structure, and hence we can use (a special case of) our approximate DP algorithm to get a $(1+\varepsilon)$-approximation for laminar cut, instead of the $2-\varepsilon$-factor previously known. Combining with other ideas in the previous paper, this gives us the 1.81-approximation.

Again, the full details of the $(1+\varepsilon)$-approximation algorithm that the 1.81-approximation FPT algorithm are deferred to the full version.

B. Related Work

The first non-trivial exact algorithm for the k-CUT problem was by Goldschmidt and Hochbaum, who gave an $O(n^{(1/2-o(1))k^2})$-time algorithm [1]; this is somewhat surprising because the related MULTIWAY CUT problem is NP-hard even for $k=3$. They also proved the problem to be NP-hard when k is part of the input. Karger and Stein improved this to an $O(n^{(2-o(1))k})$-time randomized Monte-Carlo algorithm using the idea of random edge-contractions [2]. Thorup improved the $O(n^{4k+o(1)})$-time deterministic algorithm of Kamidoi et al. [11] to an $\tilde{O}(n^{2k})$-time deterministic algorithm based on tree packings [3]. Better algorithms are known for small values of $k \in [2,6]$ [12], [13], [14], [15], [16], [17], [5].

Approximation algorithms: The first such result for k-CUT was a $2(1-1/k)$-approximation of Saran and Vazirani [6]. Later, Naor and Rabani [7], and also Ravi and Sinha [8] gave 2-approximation algorithms using tree packing and network strength respectively. Xiao et al. [18] extended Kapoor [19] and Zhao et al. [20] and generalized Saran and Vazirani to give an $(2-h/k)$-approximation in time $n^{O(h)}$. On the hardness front, Manurangsi [9] showed that for any $\varepsilon > 0$, it is NP-hard to achieve a $(2-\varepsilon)$-approximation algorithm in time $\mathrm{poly}(n,k)$ assuming the Small Set Expansion Hypothesis.

In recent work [10], we gave a 1.9997-approximation for k-CUT in FPT time $f(k)\mathrm{poly}(n)$; this does not contradict Manurangsi's work, since k is polynomial in n for his hard instances. We improve that guarantee to 1.81 by getting a better approximation ratio for the "laminar" k-cut subroutine, improving from $2-\varepsilon$ to $1+\varepsilon$. This follows as a special case of the techniques we develop in the proof of Theorem I.3, and is also deferred to the full version.

FPT algorithms: Kawarabayashi and Thorup give the first $f(\mathsf{Opt}) \cdot n^2$-time algorithm [21] for unweighted graphs. Chitnis et al. [22] used a randomized color-coding idea to give a better runtime, and to extend the algorithm to weighted graphs. Here, the FPT algorithm is parameterized by the cardinality of edges in the optimal k-CUT, not by the number of parts k. For more details on FPT algorithms and approximations, see the book [23], and the survey [24].

C. Preliminaries

For a graph $G=(V,E,w)$, consider some collection of disjoint sets $\mathcal{S} = \{S_1,\ldots,S_r\}$. Let $E_G(\mathcal{S}) = E_G(S_1,\ldots,S_r)$ denote the set of edges in $E_G[\cup_{i=1}^r S_r]$ (i.e., among the edges both of whose endpoints lie in these sets) whose endpoints belong to different sets S_i. For any vertex set S, let ∂S denote the edges with exactly one endpoint in S; hence $E_G(\mathcal{S}) = \cup_{S_i \in \mathcal{P}} \partial S_i$. For a collection of edges $F \subseteq E$, let $w(F) := \sum_{e \in F} w(e)$ be the sum of weights of edges in F. In particular, for a k-CUT solution $\{S_1,\ldots,S_k\}$, the value of the solution is $w(E_G(S_1,\ldots,S_k))$.

For a rooted tree $T = (V_T, E_T)$, let $T_v \subseteq V_T$ denote the subtree of T rooted at $v \in V_T$. For an edge $e \in E_T$ with child vertex v, let $T_e := T_v$. Finally, for any set $S \subseteq V_T$, $T_S = \sum_{v \in S} T_v$.

For some sections, we make no assumptions on the edge weights of G, while in other sections, we will assume that all edge weights in G are integers in $[W]$, for a fixed positive integer W. We default to the former unrestricted case, and explicitly mention transitioning to the latter case when needed.

II. A Fast Randomized Algorithm

In this section, we present a randomized algorithm to solve k-Cut exactly in time $\widetilde{O}(k^{O(k)} n^{(1+\omega/3)k})$, proving Theorem I.1. Section II-A introduces our high-level ideas based on Thorup's tree packing results. Section II-B shows how to refine Thorup's tree to a good tree that crosses the optimal k-cut exactly $k-1$ times, and Section II-C presents an algorithm given a good tree.

A. Thorup's Tree Packing and Thorup's Algorithm

Our starting point is a transformation from the general k-Cut problem to a problem on trees, inspired by Thorup's algorithm [3] based on greedy tree packings. We will be interested in trees that cross the optimal partition only a few times. We fix an optimal k-Cut solution, $\mathcal{S}^* := \{S_1^*, \ldots, S_k^*\}$. Let $OPT := E_G(S_1^*, \ldots, S_k^*)$ be edges in the solution, so that $w(OPT)$ is the solution value.

Definition II.1 (T-trees)**.** *A tree T of G is a ℓ-T-tree if it crosses the optimal cut at most ℓ times; i.e., $E_T(S_1^*, \ldots, S_k^*) \leq \ell$. If $\ell = 2k-2$, we often drop the quantification and call it a T-tree. If $\ell = k-1$, the minimum value possible, then we call it a tight T-tree.*

Our first step is the same as in [3]: we compute a collection $\mathcal{T}$ of $n^{O(1)}$ trees such that there exists a T-tree, i.e., a tree $T \in \mathcal{T}$ that crosses OPT at most $2k-2$ times.

Theorem II.2 ([3], Theorem 1)**.** *For $\alpha \in (0, \frac{9}{10})$, let $\mathcal{T}$ be a greedy tree packing with at least $3m(k/\alpha)^3 \ln(nmk/\alpha)$ trees. Then, on the average, the trees $T \in \mathcal{T}$ cross each minimum k-cut less than $2(k-1+2\alpha)$ times. Furthermore, the greedy tree packing algorithm takes $\widetilde{O}(k^3 m^2)$ time.*

The running time comes from the execution of $\widetilde{O}(k^3 m)$ minimum spanning tree computations. Note that, since our results are only interesting when $k \geq 7$, resulting in algorithms of running time $\Omega(n^{7+2\omega})$, we can completely ignore the running time of the greedy tree packing algorithm, which is only run once. Letting $\alpha := 1/8$, we get the following corollary.

Corollary II.3. *We can find a collection of $\widetilde{O}(k^3 m)$ trees such that for a random tree $T \in \mathcal{T}$, $|E_T(S_1^*, \ldots, S_k^*)| \leq 2k - 3/2$ in expectation. In particular, there exists a T-tree $T \in \mathcal{T}$.*

In other words, if we choose such a T-tree $T \in \mathcal{T}$, we get the following problem: find the best way to cut $\leq 2k-2$ edges of T, and then merge the connected components into exactly k components $S_1, \ldots, S_k$ so that $E_G(S_1, \ldots, S_k)$ is minimized. Thorup's algorithm accomplishes this task using brute force: try all possible $O(n^{2k-2})$ ways to cut and merge, and output the best one. This gives a runtime of $\widetilde{O}(k^3 n^{2k-2} m)$, or even $\widetilde{O}(n^{2k-2} m)$ with a more careful analysis [3]. The natural question is: can we do better than brute-force?

For the min-cut problem (when $k = 2$), Karger was able to speed up this step from $O(n^{2k-2}) = O(n^2)$ to $\widetilde{O}(n)$ using dynamic tree data structures [15]. However, this case is special: since there are ≤ 3 components produced from cutting the $\leq 2k-2 = 2$ tree edges, only one pair of components need to be merged. For larger values of k, it is not clear how to generalize the use of clever data structures to handle multiple merges.

Our randomized algorithm gets the improvement in three steps:

- First, instead of trying all possible trees $T \in \mathcal{T}$, we only look at a random subset of $\Omega(k \log n)$ trees. By Corollary II.3 and Markov's inequality, the probability that a random tree satisfies $|E_T(S_1^*, \ldots, S_k^*)| \geq 2k-1$ is $\leq (2k-3/2)/(2k-1) = 1 - \Omega(1/k)$. Therefore, by trying $\Omega(k \log n)$ random trees, we find a T-tree T w.h.p.
- Next, given a T-tree T from above, we show how to find a collection of $\approx n^{k-1}$ trees such that, with high probability, one of these trees T' is a tight T-tree, i.e., it intersects OPT in exactly $k-1$ edges. We show this in §II-B.
- Finally, given a tight T-tree T' from the previous step, we show how to solve the optimal k-Cut in time $\approx O(n^{(\omega/3)k})$, much like the k-Clique problem [25]. The runtime is not coincidental; the $W[1]$ hardness of k-Cut derives from k-Clique, and hence techniques for the former must work also for the latter. We show this in §II-C.

B. A Small Collection of "Tight" Trees

In this section we show how to find a collection of $\approx n^{k-1}$ trees such that, with high probability, one of

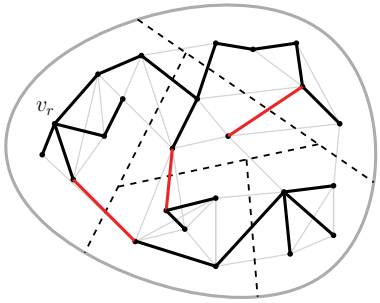

Figure II.1: The red edges are deletion-worthy edges in this T-tree; the dashed lines mark the optimal components.

these trees T' is a tight T-tree. Formally,

Lemma II.4. *There is an algorithm that takes as input a tree T such that $|E_T(S_1^*, \ldots, S_k^*)| \leq 2k-2$, and produces a collection of $k^{O(k)}n^{k-1}\log n$ trees, such that one of the new trees T' satisfies $|E_{T'}(S_1^*, \ldots, S_k^*)| = k-1$ w.p. $1 - 1/\mathrm{poly}(n)$. The algorithm runs in time $k^{O(k)}n^{k-1}m\log n$.*

The algorithm proceeds by iterations. In each iteration, our goal is to remove one edge of T and then add another edge back in, so that the result is still a tree. In doing so, the value of $|E_T(S_1^*, \ldots, S_k^*)|$ can either decrease by 1, stay the same, or increase by 1. We call an iteration *successful* if $|E_T(S_1^*, \ldots, S_k^*)|$ decreases by 1. Throughout the iterations, we will always refer to T as the current tree, which may be different from the original tree. Finally, if $|E_T(S_1^*, \ldots, S_k^*)| = \ell$ initially, then after $\ell - (k-1)$ consecutive successful iterations, we have the desired tight T-tree T'.

Assume we know ℓ beforehand; we can easily discharge this assumption later. For an intermediate tree T in the algorithm, we say that component S_i^* is **unsplit** if S_i^* induces exactly one connected component in T, and **split** otherwise. Initially, there are at most $(k-1)-\ell$ split components, possibly fewer if some components induce many components in T. Moreover, if all $\ell - (k-1)$ iterations are successful, all components are unsplit at the end.

Lemma II.5. *The probability of any iteration being successful, i.e., reducing the number of tree-edges belonging to the optimal cut, is at least $\Omega(1/nk^2)$.*

Proof: Each successful iteration has two parts: first we must delete a "deletion-worthy" edge (which happens with probability $1/(n-1)$), and then we add a "good" connecting edge (which happens with probability $\Omega(1/k^2)$). The former just uses that a tree has $n-1$ edges, but the latter must use that there are many good edges crossing the resulting cut—a naive analysis may only give $\Omega(1/m)$ for the second part.

We first describe the edges in T that we would like to delete. These are the edges such that if we delete one of them, then we are likely to make a successful iteration (after selectively adding an edge back in). We call these edges **deletion-worthy**. Let us first root the tree $T = (V, E_T)$ at an arbitrary, fixed root $v_r \in V$. For any edge e, let T_e denote the subtree below it obtained by deleting the edge e.

Definition II.6. *A* deletion-worthy *edge $e \in E_T$ satisfies the following two properties:*

(1) The edge crosses between two parts of the optimal partition, i.e., $e \in E_T(S_1^*, \ldots, S_k^*)$.

(2) There is exactly one part $S_i^* \in \mathcal{S}^*$ satisfying $S_i^* \cap T_e \neq \emptyset$ and $S_i^* - T_e \neq \emptyset$. In other words, exactly one component of $\mathcal{S}^*$ intersects T_e but is not completely contained in T_e. Note that, by condition (1), S_i^* is necessarily split.

Claim II.7. *If there is a split component S_i^*, there exists a deletion-worthy edge $e \in E_T$.*

Proof: For each S_i^*, contract every connected component of S_i^* induced in T, so that split components contract to multiple vertices. Root the resulting tree at v_r, and take a vertex $v \in V$ of maximum depth whose corresponding component S_i^* is split. It is easy to see that $v \neq v_r$ and the parent edge of v in the rooted tree is deletion-worthy. ■

Finally, we describe the deletion part of our algorithm. The procedure is simple: *choose a random edge in T to delete*. With probability $\geq 1/(n-1)$, we remove a deletion-worthy edge in T. This gives rise to the n^{-1} factor in the probability of a successful iteration.

Now we show that, conditioned on deleting a deletion-worthy edge, we can selectively add an edge to produce a successful iteration with probability $k^{-O(1)}$. In particular, we add a random edge in $E_G(T_e, V - T_e)$—i.e., an edge from subtree under e to the rest of the vertices—where the probability is weighted by the edge weights in $E_G(T_e, V - T_e)$. We show that this makes the iteration successful with probability $\Omega(1/k^2)$. (Recall that the iteration is successful if the number of tree edges lying in the optimal cut decreases by 1.)

First of all, it is clear that adding any edge in $E_G(T_e, V - T_e)$ will get back a tree. Next, to lower bound the probability of success, we begin with an auxiliary lemma.

Claim II.8. *Given a set of $k+1$ components*

$S_1, \ldots, S_{k+1}$ that partition V, we have
$$w(OPT) \leq \left(1 - \binom{k+1}{2}^{-1}\right) \cdot w(E_G(S_1, \ldots, S_{k+1})).$$

Proof: Consider merging two components S_i, S_j uniformly at random. Every edge in $E(S_1, \ldots, S_{k+1})$ has probability $\binom{k+1}{2}^{-1}$ of disappearing from the cut, so the expected new cut is
$$\left(1 - \binom{k+1}{2}^{-1}\right) \cdot E(S_1, \ldots, S_{k+1}),$$
and $w(OPT)$ can only be smaller. ■

For convenience, define $C := S_i^* \cap T_e$, where S_i^* is the split component corresponding to the deletion-worthy edge e we just deleted. Observe that the only edges in $E(T_e, V - T_e)$ that are not in OPT must be in $E(C, S_i^* - C)$; this is because, of the components S_j^* intersecting T_e, only S_i^* is split. Therefore,
$$w(E(T_e, V - T_e)) \leq w(OPT) + w(E(C, S_i^* - C)),$$
and the probability of selecting an edge in $E(C, S_i^* - C)$ is
$$\frac{w(E(C, S_i^* - C))}{w(E(T_e, V - T_e))} \geq \frac{w(E(C, S_i^* - C))}{w(OPT) + w(E(C, S_i^* - C))}. \tag{II.1}$$

Claim II.9. $w(E(C, S_i^* - C)) \geq \left(\left(1 - \binom{k+1}{2}^{-1}\right)^{-1} - 1\right) \cdot w(OPT) = \Omega(1/k^2) \cdot w(OPT)$.

Proof: The set of edges $OPT \cup E(C, S_i^* - C)$ cuts the graph G into $k+1$ components. Claim II.8 implies this set has total weight $\geq \left(1 - \binom{k+1}{2}^{-1}\right)^{-1} w(OPT)$. Observing that the edges of OPT and $E(C, S_i^* - C)$ are disjoint from each other completes the proof. ■

Using the above claim in (II.1) means the probability of selecting an edge in $E(C, S_i^* - C)$ is $\Omega(1/k^2)$. Hence the probability of an iteration being successful is $\Omega(1/(nk^2))$, completing the proof of Lemma II.5. ■

Since we have ℓ iterations, the probability that each of them is successful is $\ell^{-O(\ell)} n^{-\ell}$. If we repeat this algorithm $\ell^{O(\ell)} n^\ell \log n$ times, then with probability $1 - 1/\text{poly}(n)$, one of the final trees T' will satisfy $|E_{T'}(S_1^*, \ldots, S_k^*)| = k-1$. We can remove the assumption of knowing ℓ by trying all possible values of $\ell \in [k-1, 2k-2]$, giving a collection of $k^{O(k)} n^{k-1} \log n$ trees in running time $k^{O(k)} n^{k-1} m \log n$. This completes the proof of Lemma II.4.

C. Solving k-CUT on "Tight" Trees

In the previous section, we found a collection of $\approx n^k$ trees such that, with high probability, the intersection of one of these trees with the optimal k-cut OPT consists of only $k-1$ edges. In this section, we show that given this tree we can find the optimal k-cut in time $\approx n^{\omega k/3}$. This will follow from Lemma II.10 below. In this section, we restrict the edge weights of our graph G to be positive integers in $[W]$.

Lemma II.10. *There is an algorithm that takes a tree T and outputs, from among all partitions $\{S_1, \ldots, S_k\}$ that satisfy $|E_T(S_1, \ldots, S_k)| = k-1$, a partition $\mathcal{S}^\dagger := \{S_1^\dagger, \ldots, S_k^\dagger\}$ minimizing the number of inter-cluster edges $E_G(S_1^\dagger, \ldots, S_k^\dagger)$, in time $\widetilde{O}(k^{O(k)} n^{\lfloor (k-2)/3 \rfloor \omega + 2 + (k-2) \bmod 3} W)$.*

Given a tree $T = (V, E_T)$ and a set $F \subseteq E_T$ of tree edges, deleting these edges gives us a vertex partition $\mathcal{S}_F = \{S_1, \ldots, S_{|F|+1}\}$. Let $\mathsf{Cut}(F)$ be the set of edges in G that go between the clusters in $\mathcal{S}_F$; i.e.,
$$\mathsf{Cut}(F) := E(S_1, \ldots, S_{|F|+1}). \tag{II.2}$$
Put another way, these are the edges $(u, v) \in E$ such that the unique u-v path in T contains an edge in F. Note that Lemma II.10 seeks a set $E^\dagger \subseteq E_T$ of size $k-1$ that minimizes $w(\mathsf{Cut}(F))$.

1) A Simple Case: Incomparable Edges: Our algorithm builds upon the algorithm of Nešetřil and Poljak [25] for k-CLIQUE, using Boolean matrix multiplication to obtain the speedup from the naive $O(n^k)$ brute force algorithm. It is instructive to first consider a restricted setting to highlight the similarity between the two algorithms. This setting is as follows: we are given a vertex $v_r \in V$ and the promise that if the input tree $T = (V, E_T)$ is rooted at v_r, then the optimal $k-1$ edges $E^\dagger := E_T(S_1^\dagger, \ldots, S_k^\dagger)$ to delete are *incomparable*. By incomparable, we mean any root-leaf path in T contains at most one edge in $E^\dagger$.

Like the algorithm of [25], our algorithm creates an auxiliary graph $H = (V_H, E_H)$ on $O(n^{\lceil k/3 \rceil})$ nodes. Our graph construction differs slightly in that it always produces a tripartite graph, and that this graph has edge weights. In this auxiliary graph, we will call the vertices *nodes* in order to differentiate them from the vertices of the tree.

- The nodes in graph H will form a tripartition $V_1 \cup V_2 \cup V_3 = V_H$. For each r, let $\mathcal{F}_r \subseteq 2^E$ be the family of all sets of exactly r edges in E_T that are pairwise incomparable in T. For each $i = 1, 2, 3$, define $r_i := \lfloor \frac{(k-1)+(i-1)}{3} \rfloor$ so that $r_1 + r_2 + r_3 = k-1$. For each $i = 1, 2, 3$ and each $F \in \mathcal{F}_{r_i}$, add a node v_i^F to V_i representing set F.
- Consider a pair (V_a, V_b) of parts in the tripartition with $(a, b) \in (1, 2), (2, 3), (3, 1)$. Consider a pair of sets $F^a := \{e_1^a, \ldots, e_{r_a}^a\} \in \mathcal{F}_{r_a}$, $F^b := \{e_1^b, \ldots, e_{r_b}^b\} \in \mathcal{F}_{r_b}$; recall these are sets of r_a

and r_b incomparable edges in T. If the edges in F^a are also pairwise incomparable with the edges in F^b, then add an edge $(v_a^{F^a}, v_b^{F^b}) \in V_a \times V_b$ of weight

$$w_H(v_a^{F^a}, v_b^{F^b}) := \sum_{i=1}^{r_a} w(E(T_{e_i^a}, V - T_{e_i^a})) - \sum_{i=1}^{r_a}\sum_{j=i+1}^{r_a} w(E(T_{e_i^a}, T_{e_j^a})) - \sum_{i=1}^{r_a}\sum_{j=1}^{r_b} w(E(T_{e_i^a}, T_{e_j^b})).$$

Observe that every triple of nodes in graph H that form a triangle together represent $r_1 + r_2 + r_3 = k-1$ many incomparable edges. Moreover, the weights are set up so that for any triangle $(v_1^{F^1}, v_2^{F^2}, v_3^{F^3}) \in V_1 \times V_2 \times V_3$ such that $F := F^1 \cup F^2 \cup F^3 = \{e_1, \ldots, e_{k-1}\}$, the total weight of the edges is equal to

$$w_H(v_1^{F^1}, v_2^{F^2}) + w_H(v_2^{F^2}, v_3^{F^3}) + w_H(v_3^{F^3}, v_1^{F^1}) = \sum_{i=1}^{k-1} w(E(T_{e_i}, V - T_{e_i})) - \sum_{i=1}^{k-1}\sum_{j=i+1}^{k-1} w(E(T_{e_i}, T_{e_j})). \tag{II.3}$$

A straightforward counting argument shows that this is exactly $w(E(T_{e_1}, \ldots, T_{e_{k-1}})) = \mathsf{Cut}(F)$, the solution value of cutting the edges in F.

Hence, the problem reduces to computing a minimum weight triangle in graph H. While the minimum weight triangle problem is unlikely to admit an $O(N^{3-\varepsilon})$ time algorithm on a graph with N vertices with arbitrary edge weights, the problem does admit an $\widetilde{O}(MN^{\omega})$ time algorithm when the graph has integral edge weights in the range $[-M, M]$ [26]. Since the original graph G has integral edge weights in $[W]$, the edge weights in H must be in the range $[-O(Wm), O(Wm)]$. Therefore, we can set $N := O(n^{\lceil (k-1)/3 \rceil})$ and $M := O(Wm)$ to obtain an $\widetilde{O}(Wn^{\lceil (k-1)/3 \rceil \omega} m)$ time algorithm in this restricted setting.

2) The General Algorithm: Now we prove Lemma II.10 in full generality, and show how to find $E^\dagger$. The ideas we use here will combine the matrix-multiplication idea from the restricted case of incomparable edges, together with dynamic programming.

> Given a tree edge $e \in E_T$, and an integer $s \in [k-2]$, let $\mathsf{State}(e, s)$ denote a set of edges F in subtree T_e such that $|F| = s - 1$ and $\mathsf{Cut}(\{e\} \cup F)$ is minimized.

In other words, $\mathsf{State}(e, s)$ represents the optimal way to cut edge e along with $s-1$ edges in T_e. For ease of presentation, we assume that this value is unique. Observe that, once all of these states are computed, the remaining problem boils down to choosing an integer $\ell \in [k-1]$, integers $s_1, \ldots, s_\ell$ whose sum is $k-1$, and incomparable edges $e_1, \ldots, e_\ell$ that minimizes

$$\mathsf{Cut}\left(\bigcup_{i=1}^{\ell} \mathsf{State}(e_i, s_i)\right) = \sum_{i=1}^{k-1} \mathsf{State}(e_i, s_i) - \sum_{i=1}^{k-1}\sum_{j=i+1}^{k-1} w(E(T_{e_i}, T_{e_j})).$$

Comparing this expression to (II.3) suggests that this problem is similar to the incomparable case in §II-C1, a connection to be made precise later.

We now compute states for all edges $e \in E_T$, which we do from bottom to top (leaf to root). When e is a leaf edge, the states are straightforward: $\mathsf{State}(e, 1) = \mathsf{Cut}(\{e\})$ and $\mathsf{State}(e, s) = \infty$ for $s > 1$. Also, for each edge $e \in E_T$, define $\mathrm{desc}(e)$ to be all "descendant edges" of e, formally defined as all edges $f \in E_T - e$ whose path to the root contains edge e.

Fix an edge $e \in E_T$ and an $s \in [k-2]$, for which we want to compute $\mathsf{State}(e, s)$. Suppose we order the edges in T_e in an arbitrary but fixed order. Let us now figure out some properties for this (unknown) value of $\mathsf{State}(e, s)$. As a thought experiment, let $F^\dagger$ be the list of all the "maximal" edges in $\mathsf{State}(e, s)$—in other words, $f \in F^\dagger$ iff $f \in \mathsf{State}(e, s)$ and $f \notin \mathrm{desc}(f')$ for all $f' \in \mathsf{State}(e, s)$. Let $\ell^\dagger := |F^\dagger|$ and $F^\dagger = (e_1^\dagger, \ldots, e_{\ell^\dagger}^\dagger)$ be the sequence in the defined order, and for each $e_i^\dagger$, let $s_i^\dagger := 1 + |\mathrm{desc}(e_i^\dagger) \cap \mathsf{State}(e, s)|$. Observe that $\sum_i s_i^\dagger = s - 1$, and that we must satisfy

$$\mathsf{State}(e, s) = \bigcup_{i=1}^{\ell^\dagger} \left(\{e_i^\dagger\} \cup \mathsf{State}(e_i^\dagger, s_i^\dagger) \right). \tag{II.4}$$

Also,

$$w(\mathsf{State}(e, s)) = E(T_e, V - T_e) + \sum_{i=1}^{\ell^\dagger} w(E(G[T_e]) \cap \mathsf{Cut}\left(\{e_i^\dagger\} \cup \mathsf{State}(e_i^\dagger, s_i^\dagger)\right)) - \sum_{i=1}^{\ell^\dagger}\sum_{j=i+1}^{\ell^\dagger} w(E_{G[T_e]}[T_{e_i^\dagger}, T_{e_j^\dagger}]),$$

since the only edges double-counted in the first summation of $w(\mathsf{State}(e, s))$ are those connecting different $T_{e_i^\dagger}, T_{e_j^\dagger}$.

Given these "ideal" values $\ell^\dagger$ and $\{s_i^\dagger\}$, our algorithm repeats the following procedure multiple times:

- Pick a number ℓ uniformly at random in $[s-1]$. Then, let function $\sigma : [\ell] \to [s-1]$ be chosen uniformly at random among all $\leq (s-1)^\ell$ possible functions satisfying $\sum_{i=1}^{\ell} \sigma(i) = s-1$. With probability $\geq (s-1)^{-(\ell^\dagger + 1)} = k^{-O(k)}$, we correctly

guess $\ell = \ell^\dagger$ and $\sigma(i) = s_i^\dagger$ for each $i \in [\ell]$.[1]

- Construct an auxiliary graph H as follows. As in §II-C1, H has a tripartition $V_1 \cup V_2 \cup V_3 = V_H$, and assume there is an arbitrary but fixed total ordering on the edges of the tree. For each r, let $\mathcal{F}_r \subseteq 2^E$ be the family of all sets of exactly r edges in E_T that are pairwise incomparable in T. For each $i = 1, 2, 3$, let $r_i := \lfloor \frac{\ell+(i-1)}{3} \rfloor$ so that $r_1 + r_2 + r_3 = \ell$, and for each $F \in \mathcal{F}_{r_i}$, add a node v_i^F to V_i representing the edges F *as a sequence in the total order*.
  Also, define $R_i := \sum_{j=1}^{i-1} r_i$ for $i = 1, 2, 3, 4$. Note that $R_1 = 0$ and $R_4 = r_1 + r_2 + r_3 = \ell$. Our intention is map the integer values $\{\sigma(R_i + 1), \sigma(R_i + 2), \ldots, \sigma(R_{i+1})\}$ to the sequences represented by nodes in V_i, as we will see later. Consider each tripartition pair (V_a, V_b) with $(a, b) \in (1, 2), (2, 3), (3, 1)$. For each pair $F^a \in \mathcal{F}_{r_a}$, $F^b \in \mathcal{F}_{r_b}$ represented as ordered sequences $F^a = (e_1^a, \ldots, e_{r_a}^a)$ and $F^b = (e_1^b, \ldots, e_{r_b}^b)$, if the edges in F^a are pairwise incomparable with the edges in F^b, then add an edge $(v_a^{F^a}, v_b^{F^b}) \in V_a \times V_b$ in the auxiliary graph of weight

$$\begin{aligned} w_H(v_a^{F^a}, v_b^{F^b}) := &\sum_{i=1}^{r_a} w\Big(\mathsf{State}\big(e_i^a, \sigma(R_a + i)\big)\Big) \\ &- \sum_{i=1}^{r_a} \sum_{j=i+1}^{r_a} w(E_{G[T_e]}(T_{e_i^a}, T_{e_j^a})) \\ &- \sum_{i=1}^{r_a} \sum_{j=1}^{r_b} w(E_{G[T_e]}(T_{e_i^a}, T_{e_j^b})). \end{aligned}$$

  For any triangle $(v_1^{F^1}, v_2^{F^2}, v_3^{F^3}) \in V_1 \times V_2 \times V_3$ such that $F := F^1 \cup F^2 \cup F^3$ has ordered sequence $(e_1, \ldots, e_\ell)$, the total weight of the edges is equal to

$$\begin{aligned} &w_H(v_1^{F^1}, v_2^{F^2}) + w_H(v_2^{F^2}, v_3^{F^3}) + w_H(v_3^{F^3}, v_1^{F^1}) \\ &= \sum_{i=1}^{\ell} w(\mathsf{State}(e_i^a, \sigma(i))) \\ &\quad - \sum_{i=1}^{\ell} \sum_{j=i+1}^{\ell} w(E_{G[T_e]}(T_{e_i}, T_{e_j})). \end{aligned}$$

A straightforward counting argument shows that this is exactly

$$w\Big(\mathsf{Cut}\big(\{e\} \cup \bigcup_{i=1}^{\ell} \mathsf{State}(e_i, \sigma(i))\big)\Big) - w(E(T_e, V - T_e)).$$

Thus, the weight of each triangle, with $w(E(T_e, V - T_e))$ added to it, corresponds to the cut value of one possible solution to $\mathsf{State}(e, s)$. Moreover, if we guess ℓ and $\sigma : [\ell] \to [s-1]$ correctly, then this triangle will exist in auxiliary graph H, and we will compute the correct state if we compute the minimum weight triangle in $\widetilde{O}(Wn^{\lceil \ell/3 \rceil \omega} m)$ time. Since the probability of guessing $\ell, \sigma(\cdot)$ correctly is $k^{-O(k)}$, we repeat the guessing $k^{O(k)} \log n$ times to succeed w.h.p. in time $\widetilde{O}(k^{O(k)} n^{\lceil (k-2)/3 \rceil \omega} mW)$. This concludes the computation of each $\mathsf{State}(e, s)$; since there are $O(kn)$ such states, the total running time becomes $\widetilde{O}(k^{O(k)} n^{\lceil (k-2)/3 \rceil \omega + 1} mW)$.

Lastly, to compute the final k-CUT value, we let $s := k - 1$ and construct the same auxiliary graph H, except that $k - 2$ is replaced by $k - 1$ and the relevant graph $G[T_e]$ becomes the entire G. By the same counting arguments, the weight of triangle $(v_1^{F^1}, v_2^{F^2}, v_3^{F^3}) \in V_1 \times V_2 \times V_3$ such that $F := F^1 \cup F^2 \cup F^3$ has ordered sequence $(e_1, \ldots, e_\ell)$ is exactly

$$w\Big(\mathsf{Cut}\big(\{e\} \cup \bigcup_{i=1}^{\ell} \mathsf{State}(e_i, \sigma(i))\big)\Big).$$

Again, by repeating the procedure $k^{O(k)} \log n$, we compute an optimal k-CUT w.h.p., in time $\widetilde{O}(k^{O(k)} n^{\lceil (k-1)/3 \rceil \omega} mW)$. Note that this time is dominated by the running time $\widetilde{O}(k^{O(k)} n^{\lceil (k-2)/3 \rceil \omega + 1} mW)$ of computing the states.

In order to get the runtime claimed in Theorem I.1, we need a couple more ideas—however, they can be skipped on the first reading, so we defer them to the full version of this paper.

III. A Faster Deterministic Algorithm

In this section, we show how to build on the randomized algorithm of the previous section and improve it in two ways: we give a deterministic algorithm, with a better asymptotic runtime. (The algorithm of the previous section has a better runtime for smaller values of k.) Formally, the main theorem of this section is the following:

Theorem I.2 (Even Faster Deterministic Algorithm)**.** *Let W be a positive integer. For any $\varepsilon > 0$, there is a deterministic algorithm for exact k-CUT on graphs with edge weights in $[W]$ with runtime $k^{O(k)} n^{(2\omega/3+\varepsilon)k+O(1)} W \approx O(k^{O(k)} n^{(2\omega/3)k})$.*

Our main idea is a more direct application of matrix multiplication, without paying the $O(n^k)$ overhead in the previous section. Instead of converting a given T-tree to a "tight" tree where matrix multiplication can be combined with dynamic programming, with only $n^{O(\log k)}$ overhead, we partition the given T-tree to subforests that are amenable to direct matrix multiplication approach.

As in §II we build on the framework of Thorup [3], where the k-CUT problem reduces to $n^{O(1)}$ instances

[1] Of course, we could instead brute force over all $k^{O(k)}$ possible choices of ℓ and σ.

of the following problem: given the graph G and a spanning tree T, find a way to cut $\leq 2k-2$ edges from T, and then merging the connected components of T into k connected components, that minimizes the number of cut edges in G. Again, the optimal k-cut is denoted by $\mathcal{S}^* = \{S_1^*, \ldots, S_k^*\}$.

For the rest of this section, let T be some spanning tree in the instance that crosses the optimal k-cut in $(r-1) \leq 2k-2$ edges. If we delete these $r-1$ edges from T, this gives us r components, which we denote by $C_1^*, \ldots, C_r^*$ — these are a refinement of $\mathcal{S}^*$, and hence can be then be merged together to give us $\mathcal{S}^*$. Let $E_T^* := E_T(C_1^*, \ldots, C_r^*) = E_T(S_1^*, \ldots, S_k^*)$ be these $r-1$ cut edges in T.

A. Balanced Separators

We first show the existence of a small-size *balanced separator* in the following sense: there exist forests F_1, F_2, F_3 whose vertices partition $V(T)$, such that

(i) we can delete $O(\log k)$ edges in T to get the forests, i.e., $|E(T) - \bigcup_{i=1}^{3} E(F_i)| = O(\log k)$, and
(ii) we want to cut few edges from each forest, i.e., $|E(F_i) \cap E_T^*| \leq \lceil 2k/3 \rceil$ for each i.

Of course, small-size balanced edge separators typically do not exist in general trees, such as if the tree is a star. So we first apply a degree-reducing step. This operation reduces the maximum degree of the tree to 3, at a cost of introducing "Steiner" vertices, which are handled later.

Lemma III.1 (Degree-Reduction)**.** *Given a tree $T = (V_T, E_T)$, we can construct a tree $T' = (V_{T'}, E_{T'})$, where $V_{T'} = V_T \cup X$, where X are called the Steiner vertices, such that*

1. *T' has maximum degree 3.*
2. *$|V(T')| \leq 2|V(T)|$*
3. *For every way to cut r edges in T and obtain components $C_1, \ldots, C_{r+1}$, there is a way to cut r edges in T' and obtain components $C_1', \ldots, C_{r+1}'$ such that each C_i is precisely $C_i' \cap V_T$.*

Proof: Root the tree T at an arbitrary root, and select any non-Steiner vertex $v \in V_T$ with more than two children. Replace the star composed of v and its children with an arbitrary binary tree with v as the root and its children as the leaves. This process does not introduce any new vertex with more than two children, so we can repeat it until it terminates, giving us a tree T' of maximum degree 3. Every star of z edges adds exactly $z-1$ Steiner nodes, and there are $\leq |V_T| - 1$ edges initially, so $\leq |V_T| - 2$ Steiner vertices are added throughout the process, and $|V_{T'}| \leq 2|V_T|$. Finally, if we cut some r edges $(u_i, v_i) \in E_T$ where v_i is the parent of u_i, then we can cut the parent edge of each u_i in T' to obtain the required components. ∎

Having applied Lemma III.1 to T to get T', Property (3) shows that we can still delete $\leq 2k-2$ edges in T' to obtain the components of the optimal solution before merging. To avoid excess notation, we assume that T itself is a tree of degree ≤ 3, possibly with Steiner nodes. From now on, our task is to delete $\leq 2k-2$ edges of T and merge them into k components, *each of which containing at least one non-Steiner vertex*, that minimizes the number of cut edges in G. To show that the aforementioned forests F_1, F_2, F_3 exist in the new tree T, we introduce the following easy lemma:

Lemma III.2. *Let T be a tree of degree ≤ 3 and $F \subseteq E(T)$ be a subset of the edges. For any integer $r \in [1, |F|-1]$, there exists a vertex partition A, B of $V(T)$ such that $|E_T(A,B)| = O(\log(r+1))$, and the induced subgraphs $T[A]$ and $T[B]$ have at most r and at most $|F| - r$ edges from F, respectively.*

Proof: We provide an algorithm that outputs a collection of $O(\log r)$ disjoint subtrees whose union comprises A. Root T at a degree-1 vertex, and find a vertex of maximal depth whose rooted subtree contains $> r$ edges from F. The degree condition ensures that v has ≤ 2 children, and by maximality, all of v's children have $\leq r$ edges in F in their subtrees. Moreover, the edges in T_v is precisely the union of the edge sets $E(T_u) \cup \{(u,v)\}$ for all children u of v. For convenience, define $E^+(T_u) := E(T_u) \cup \{(u,v)\}$ for a child u of v. So there must be a child u of v satisfying $|E^+(T_u) \cap F| \in (r/2, r]$.

If $|E^+(T_u) \cap F| = r$, then $(A, B) = (V(T_u), V(T) - V(T_u))$ is a satisfying partition with $|E_T(A,B)| = 1$, and we are done. Otherwise, recurse on the tree $T' := T[V(T) - V(T_u)]$ where we remove (u,v) and the subtree below it, with the parameters $r' := r - |E^+(T_u) \cap F|$ and $F' := F \setminus E^+(T_u)$ to get partition (A', B'), and set $A := A' \cup V(T_u)$ and $B := B'$. By recursion, we guarantee that

$$\begin{aligned} |E(T[A]) \cap F| &\leq |E(T[A']) \cap F'| + |E^+(T_u) \cap F| \\ &\leq (r - |E^+(T_u) \cap F|) + |E^+(T_u) \cap F| \\ &= r \end{aligned}$$

and

$$\begin{aligned} |E(T[B]) \cap F'| &= |E(T[B']) \cap F'| \\ &\leq |F'| - (r - |E^+(T_u) \cap F|) \\ &= (|F| - |E^+(T_u) \cap F|) \\ &\quad - (r - |E^+(T_u) \cap F|) \\ &= |F| - r. \end{aligned}$$

Since the value of r drops by at least half each time, there are $O(\log r)$ steps of the recursion. Each step can

only add the additional edge (u, v) to $|E_T(A, B)|$, so $|E_T(A, B)| = O(\log r)$. ∎

Corollary III.3. *There exist forests F_1, F_2, F_3 whose vertices partition $V(T)$ such that*

(i) the number of crossing edges is $|E(T) - \bigcup_{i=1}^{3} E(F_i)| = O(\log |E_T^*|)$, and
(ii) $|E(F_i) \cap E_T^*| \leq \lceil |E_T^*|/3 \rceil$ for each i.

Proof: We apply Lemma III.2 with $F := E_T^*$ and $r := \lceil |E_T^*|/3 \rceil$ to obtain the separation (A, B), and then set $F_1 := T[A]$. Before applying the lemma again on B, we first connect the connected components of B arbitrarily into a tree; let F^+ denote the added edges. Then, we apply with $F := E_T^* - E[F_1]$ and $r := \lceil |E_T^*|/3 \rceil$ to obtain separation (A', B'), and then set $F_2 := T[A'] - F^+$ and $F_3 := T[B'] - F^+$. ∎

Given this result, our algorithm starts by trying all possible $n^{O(\log k)}$ ways to delete $O(\log k)$ edges of T and partition the connected components into three forests. By Corollary III.3, one of these attempts produces the desired F_1, F_2, F_3 satisfying the two properties.

B. Matrix Multiplication

The balanced partitioning procedure from the previous section gives us three forests F_1, F_2, F_3, such that the optimal solution cuts at most $2k/3$ edges in each — and then combines the resulting pieces together. The algorithm now computes these solutions separately for each forest, and then uses matrix multiplication to combine these solutions together.

Indeed, for each $F_i \in \{F_1, F_2, F_3\}$, the algorithm computes all $O(n^{\lceil 2k/3 \rceil})$ ways to cut $\leq \lceil 2k/3 \rceil$ edges in F_i, followed by all $3^{O(k)}$ ways to label each of the $\leq \lceil 2k/3 \rceil + O(\log k)$ connected components with a label in $[k]$. For each one forest, note that some of these components might only contain Steiner vertices of the tree; we call these the *Steiner components*, and the other the *normal components*. For each subset $S \subseteq [k]$, let $\mathcal{F}_i^S$ denote all possible ways to cut and label F_i in the aforementioned manner such that the set of labels that are attributed to at least one normal component is precisely S.

The algorithm now enumerates over every possible triple of subsets $S_1, S_2, S_3 \subseteq [k]$ (not necessarily disjoint) whose union is exactly $[k]$. Note that there are at most 7^k of these triples. For each triple S_1, S_2, S_3, we construct the following tripartite auxiliary graph $H = (V_H, E_H)$ on $O(k^{O(k)} n^{\lceil 2k/3 \rceil})$ vertices,

with tripartition $V_H = V_1 \uplus V_2 \uplus V_3$. For each $i = 1, 2, 3$, each element in $\mathcal{F}_i^{S_i}$ is a tuple (X_i, σ_i) where $X_i \subseteq F_i$ is a set of edges that we cut from F_i, and σ_i is a labeling of the normal components in the resulting forest so that the label set is exactly S_i. Now for each $(X, \sigma) \in \mathcal{F}_i^{S_i}$, add a node $v_i^{(X,\sigma)}$ to V_i. Moreover, for each tripartition pair (V_a, V_b) with $(a, b) \in (1, 2), (2, 3), (3, 1)$, and for each way $(X_a, \sigma_a) \in \mathcal{F}_a^{S_a}$ to cut F_a into components $C_1^a, \ldots, C_{r_a}^a$ with labels $\sigma_a(1), \ldots, \sigma_a(r_a)$, and for each way $(X_b, \sigma_b) \in \mathcal{F}_b^{S_b}$ to cut F_b into components $C_1^b, \ldots, C_{r_b}^b$ with labels $\sigma_b(1), \ldots, \sigma_b(r_b)$, we add an edge $(v_a^{(X_a,\sigma_a)}, v_b^{(X_b,\sigma_b)}) \in V_a \times V_b$ of weight

$$\begin{aligned} &w_H(v_a^{(X_a,\sigma_a)}, v_b^{(X_b,\sigma_b)}) \\ &:= \sum_{i=1}^{r_a} \sum_{j=i+1}^{r_a} \mathbb{1}[\sigma_a(i) \neq \sigma_a(j)] \cdot w(E_G[C_i^a, C_j^a]) \\ &\quad + \sum_{i=1}^{r_a} \sum_{j=1}^{r_b} \mathbb{1}[\sigma_a(i) \neq \sigma_b(j)] \cdot w(E_G[C_i^a, C_j^b]), \end{aligned} \tag{III.5}$$

where $\mathbb{1}$ is the indicator function, taking value 1 if the corresponding statement is true and 0 otherwise. Finally, the algorithm computes the minimum weight triangle in H.

A straightforward counting argument shows that the weight of each triangle $(v_1^{(X_1,\sigma_i)}, v_2^{(X_2,\sigma_2)}, v_3^{(X_3,\sigma_3)})$ in H is exactly the value of the cut in G obtained by merging all components in F^1, F^2, F^3 with the same label together. In particular, for the correct triple S_1, S_2, S_3 for E_T^*, there is a triangle in H whose weight is the cost of the optimal solution, and the algorithm will find it, proving the correctness of the algorithm.

As for running time, the algorithm has an $n^{O(\log k)} 7^k$ overhead for the guesswork of finding the forests (F_1, F_2, F_3) and the correct triple (S_1, S_2, S_3) of subsets of labels. This is followed by computing matrix multiplication on a graph with $k^{O(k)} n^{\lceil 2k/3 \rceil}$ nodes, with edge weights in $[-Wm, Wm]$. Altogether, this takes $k^{O(k)} n^{(2\omega/3+\varepsilon)k+O(1)} W$ for any $\varepsilon > 0$, proving Theorem I.2.

REFERENCES

[1] O. Goldschmidt and D. S. Hochbaum, "A polynomial algorithm for the k-cut problem for fixed k," *Math. Oper. Res.*, vol. 19, no. 1, pp. 24–37, 1994. [Online]. Available: http://dx.doi.org/10.1287/moor.19.1.24

[2] D. R. Karger and C. Stein, "A new approach to the minimum cut problem," *Journal of the ACM (JACM)*, vol. 43, no. 4, pp. 601–640, 1996.

[3] M. Thorup, "Minimum k-way cuts via deterministic greedy tree packing," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*. ACM, 2008, pp. 159–166.

[4] F. Le Gall, "Powers of tensors and fast matrix multiplication," in *Proceedings of the 39th international symposium on symbolic and algebraic computation*. ACM, 2014, pp. 296–303.

[5] M. S. Levine, "Fast randomized algorithms for computing minimum {3, 4, 5, 6}-way cuts," in *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2000, pp. 735–742.

[6] H. Saran and V. V. Vazirani, "Finding k-cuts within twice the optimal," *SIAM Journal on Computing*, vol. 24, no. 1, pp. 101–108, 1995.

[7] J. Naor and Y. Rabani, "Tree packing and approximating k-cuts," in *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (Washington, DC, 2001)*. SIAM, Philadelphia, PA, 2001, pp. 26–27.

[8] R. Ravi and A. Sinha, "Approximating k-cuts using network strength as a Lagrangean relaxation," *European J. Oper. Res.*, vol. 186, no. 1, pp. 77–90, 2008. [Online]. Available: http://dx.doi.org/10.1016/j.ejor.2007.01.040

[9] P. Manurangsi, "Inapproximability of Maximum Edge Biclique, Maximum Balanced Biclique and Minimum k-Cut from the Small Set Expansion Hypothesis," in *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 80, 2017, pp. 79:1–79:14. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/7500

[10] A. Gupta, E. Lee, and J. Li, "An FPT algorithm beating 2-approximation for k-cut," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, 2018, pp. 2821–2837. [Online]. Available: https://doi.org/10.1137/1.9781611975031.179

[11] Y. Kamidoi, N. Yoshida, and H. Nagamochi, "A deterministic algorithm for finding all minimum k-way cuts," *SIAM J. Comput.*, vol. 36, no. 5, pp. 1329–1341, 2006/07. [Online]. Available: http://dx.doi.org/10.1137/050631616

[12] H. Nagamochi and T. Ibaraki, "Computing edge-connectivity in multigraphs and capacitated graphs," *SIAM J. Discrete Math.*, vol. 5, no. 1, pp. 54–66, 1992. [Online]. Available: http://dx.doi.org/10.1137/0405004

[13] J. Hao and J. B. Orlin, "A faster algorithm for finding the minimum cut in a directed graph," *J. Algorithms*, vol. 17, no. 3, pp. 424–446, 1994, third Annual ACM-SIAM Symposium on Discrete Algorithms (Orlando, FL, 1992). [Online]. Available: http://dx.doi.org/10.1006/jagm.1994.1043

[14] M. Burlet and O. Goldschmidt, "A new and improved algorithm for the 3-cut problem," *Oper. Res. Lett.*, vol. 21, no. 5, pp. 225–227, 1997. [Online]. Available: http://dx.doi.org/10.1016/S0167-6377(97)00043-6

[15] D. R. Karger, "Minimum cuts in near-linear time," *J. ACM*, vol. 47, no. 1, pp. 46–76, 2000. [Online]. Available: http://dx.doi.org/10.1145/331605.331608

[16] H. Nagamochi and T. Ibaraki, "A fast algorithm for computing minimum 3-way and 4-way cuts," *Math. Program.*, vol. 88, no. 3, Ser. A, pp. 507–520, 2000. [Online]. Available: http://dx.doi.org/10.1007/PL00011383

[17] H. Nagamochi, S. Katayama, and T. Ibaraki, "A faster algorithm for computing minimum 5-way and 6-way cuts in graphs," *J. Comb. Optim.*, vol. 4, no. 2, pp. 151–169, 2000. [Online]. Available: http://dx.doi.org/10.1023/A:1009804919645

[18] M. Xiao, L. Cai, and A. C.-C. Yao, "Tight approximation ratio of a general greedy splitting algorithm for the minimum k-way cut problem," *Algorithmica*, vol. 59, no. 4, pp. 510–520, 2011.

[19] S. Kapoor, "On minimum 3-cuts and approximating k-cuts using cut trees," in *Integer programming and combinatorial optimization (Vancouver, BC, 1996)*, ser. Lecture Notes in Comput. Sci. Springer, Berlin, 1996, vol. 1084, pp. 132–146. [Online]. Available: http://dx.doi.org/10.1007/3-540-61310-2_11

[20] L. Zhao, H. Nagamochi, and T. Ibaraki, "Approximating the minimum k-way cut in a graph via minimum 3-way cuts," *J. Comb. Optim.*, vol. 5, no. 4, pp. 397–410, 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1011620607786

[21] K.-i. Kawarabayashi and M. Thorup, "The minimum k-way cut of bounded size is fixed-parameter tractable," in *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*. IEEE, 2011, pp. 160–169.

[22] R. Chitnis, M. Cygan, M. Hajiaghayi, M. Pilipczuk, and M. Pilipczuk, "Designing FPT algorithms for cut problems using randomized contractions," *SIAM J. Comput.*, vol. 45, no. 4, pp. 1171–1229, 2016. [Online]. Available: http://dx.doi.org/10.1137/15M1032077

[23] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized algorithms*. Springer, Cham, 2015. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-21275-3

[24] D. Marx, "Parameterized complexity and approximation algorithms," *The Computer Journal*, vol. 51, no. 1, pp. 60–78, 2007.

[25] J. Nešetřil and S. Poljak, "On the complexity of the subgraph problem," *Commentationes Mathematicae Universitatis Carolinae*, vol. 26, no. 2, pp. 415–419, 1985.

[26] V. V. Williams and R. Williams, "Subcubic equivalences between path, matrix and triangle problems," in *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*. IEEE, 2010, pp. 645–654.