A Parameterized Approximation Scheme for Min k-Cut

Daniel Lokshtanov*, Saket Saurabh†, and Vaishali Surianarayanan*,
*Department of Computer Science, University of California, Santa Barbara, USA
{daniello,vaishali}@ucsb.edu
†The Institute of Mathematical Sciences, HBNI, Chennai, India, and University of Bergen, Norway
saket@imsc.res.in

***Abstract*—In the MIN k-CUT problem, input is an edge weighted graph G and an integer k, and the task is to partition the vertex set into k non-empty sets, such that the total weight of the edges with endpoints in different parts is minimized. When k is part of the input, the problem is NP-complete and hard to approximate within any factor less than 2. Recently, the problem has received significant attention from the perspective of parameterized approximation. Gupta *et al.* [SODA 2018] initiated the study of FPT-approximation for the MIN k-CUT problem and gave an 1.9997-approximation algorithm running in time $2^{\mathcal{O}(k^6)}n^{\mathcal{O}(1)}$. Later, the same set of authors [FOCS 2018] designed an $(1+\epsilon)$-approximation algorithm that runs in time $(k/\epsilon)^{\mathcal{O}(k)}n^{k+\mathcal{O}(1)}$, and a 1.81-approximation algorithm running in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$. More, recently, Kawarabayashi and Lin [SODA 2020] gave a $(5/3+\epsilon)$-approximation for MIN k-CUT running in time $2^{\mathcal{O}(k^2 \log k)}n^{\mathcal{O}(1)}$.**

In this paper we give a parameterized approximation algorithm with best possible approximation guarantee, and best possible running time dependence on said guarantee (up to Exponential Time Hypothesis (ETH) and constants in the exponent). In particular, for every $\epsilon > 0$, the algorithm obtains a $(1+\epsilon)$-approximate solution in time $(k/\epsilon)^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$. The main ingredients of our algorithm are: a simple sparsification procedure, a new polynomial time algorithm for decomposing a graph into highly connected parts, and a new exact algorithm with running time $s^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ on unweighted (multi-) graphs. Here, s denotes the number of edges in a minimum k-cut. The latter two are of independent interest.

I. INTRODUCTION

In this article we focus on the MIN k-CUT problem from the perspective of parameterized approximation. Here input is an edge weighted graph G and an integer k, and the task is to partition the vertex set into k non-empty sets, say $\tilde{P}$, such that the total weight of the edges with endpoints in different parts is minimized. We call such a partition as *optimal k-cut*, or *minimum k-cut* or simply a *k-cut*. For $k = 2$, this is just the classic GLOBAL MIN-CUT problem, which can be solved in polynomial time. In fact, for every fixed k, the problem is known to be polynomial time solvable [1]. However, when k is part of the input, the problem is NP-complete [1].

Traditionally, the problem has been extensively studied from three perspectives: (a) exact algorithms for small values of k; (b) combinatorial upper bound on the number of minimum k-cuts; and (c) approximation algorithms. In particular, already in early 90's, Goldschmidt and Hochbaum [1] gave the first polynomial-time algorithm for fixed k, with running time $\mathcal{O}(n^{(0.5-o(1))k^2})$. In 1996, Karger and Stein [2] obtained their ingenious *contraction algorithm*, a randomized algorithm with running time $\tilde{\mathcal{O}}(n^{2(k-1)})$[1]. The same algorithm immediately yields a proof that the number of k-cuts of minimum weight in any undirected graph on n vertices is upper bounded by $\tilde{\mathcal{O}}(n^{2(k-1)})$. In 2008, Thorup [3] obtained a deterministic algorithm that essentially matched the running time of Karger and Stein [2], and in 2019, Chekuri et al. [4] used a LP based approach to improve the running time by a factor of n. This algorithm is based on a tree-packing theorem which allows to efficiently find a tree that crosses the optimal k-cut at most $2k-2$ times. Then the algorithm simply enumerates over all possible sets of $2k-2$ edges of this tree and all ways of merging the components of the tree minus these $2k-2$ edges into k non-empty groups. From the perspective of approximation algorithms there are several $2(1-\frac{1}{k})$-approximation algorithms, that run in time polynomial in n and k [5], [6], [7]. This approximation ratio cannot be improved assuming the Small Set Expansion Hypothesis (SSE) [8].

Thus, as of early 2018, the status was as follows. The direction of polynomial time approximation algorithms was essentially settled, with factor $2(1-\frac{1}{k})$ approximation algorithms and matching lower bounds. The fastest algorithm for small k had running time $\tilde{\mathcal{O}}(n^{2(k-1)})$, and a $f(k)n^{o(k)}$ lower bound on the running time [9], [10] was known under the ETH. It remained an interesting research direction to find an algorithm with an n^{ck} running time and prove that the constant c is optimal under reasonable complexity assumptions. The best upper bound on the number of minimum k-cuts was $\tilde{\mathcal{O}}(n^{2(k-1)})$, while the best lower bound was essentially $\Omega((n/(k-1)^{k-1})$, what you obtain from the complete $(k-1)$-partite graph. Since early 2018 the problem has seen a remarkable level of

[1] $\tilde{\mathcal{O}}$ hides the poly-logarithimic factor in the running time.

2575-8454/20/$31.00 ©2020 IEEE
DOI 10.1109/FOCS46700.2020.00079

activity [11], [12], [13], [14], [15], [16], [17], with a new research direction of parameterized approximation being initiated, and the two remaining established research directions (exact algorithms and combinatorial bounds) essentially being completely resolved.

For exact algorithms for small values of k, Gupta et al. [12] made the first improvement in two decades, by designing an algorithm with running time $\mathcal{O}(k^{\mathcal{O}(k)}n^{(\frac{2\omega}{3}+o(1))k})$ for graphs with polynomial integer weights. Subsequently, for unweighted graphs, Li [15] obtained an algorithm with running time $\mathcal{O}(k^{\mathcal{O}(k)}n^{(1+o(1)k})$. For the original formulation of the problem (with general integer weights) Gupta et al. [14] showed an $\mathcal{O}(n^{(1.98+o(1))k})$-time algorithm. Their algorithm also implied a new and improved $\mathcal{O}(n^{(1.98+o(1))k})$ upper bound on the number of minimum k-cuts. Finally Gupta et al. [16] fully resolved the problem by showing that for every fixed $k \geq 2$, the Karger-Stein algorithm [2] outputs any fixed minimum k-cut with probability at least $\widehat{\mathcal{O}}(n^{-k})$, where $\widehat{\mathcal{O}}(\cdot)$ hides a $2^{\mathcal{O}(\ln \ln n)^2}$ factor. This immediately gives an extremal bound of $\widehat{\mathcal{O}}(n^k)$, on the number of minimum k-cuts in an n-vertex graph and an algorithm for MIN k-CUT in similar running time. Both the extremal bound and the running time of the algorithm are essentially tight (under reasonable assumptions). Indeed the extremal bound matches known lower bounds up to $\widehat{\mathcal{O}}(1)$ factors, while any further improvement to the exact algorithm would imply an improved algorithm for MAX-WEIGHT k-CLIQUE [18], [19], which has been conjectured not to exist.

The lower bound of $(2-\epsilon)$ on polynomial time approximation algorithms [8], coupled with the $f(k)n^{o(k)}$ running time lower bound for exact algorithms [9], [10] naturally leads to the question of parameterized approximation: how good approximation ratio can we achieve if we allow the algorithm to have running time $f(k)n^{\mathcal{O}(1)}$? In 2018, Gupta et al. [13] were the first to ask this question, and showed that relaxing the running time requirement from polynomial to $f(k)n^{\mathcal{O}(1)}$ does lead to an improved approximation guarantee, by giving an 1.9997-approximation algorithm for MIN k-CUT running in time $2^{\mathcal{O}(k^6)}n^{\mathcal{O}(1)}$. Subsequently the same set of authors [12] designed an $(1+\epsilon)$-approximation algorithm that runs in time $(k/\epsilon)^{\mathcal{O}(k)}n^{k+\mathcal{O}(1)}$, and a 1.81-approximation algorithm running in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$. More recently, Kawarabayashi and Lin [17] gave a $(5/3+\epsilon)$-approximation for MIN k-CUT running in time $2^{\mathcal{O}(k^2\log k)}n^{\mathcal{O}(1)}$. In this paper we bring the direction of parameterized approximation to its natural conclusion by giving an algorithm with best possible approximation guarantee, and best possible running time dependence on said guarantee (up to ETH and constants in the exponent). In particular, for every $\epsilon > 0$, the algorithm obtains a $(1+\epsilon)$-approximate solution in time $(k/\epsilon)^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$.

Theorem I.1. *There exists a randomized algorithm that given a graph G, weight function $w : V(G) \to \mathbb{Q}_{\geq 0}$, integer k, and an $\epsilon > 0$, runs in time $(k/\epsilon)^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ and outputs a partition $\tilde{P}$ of $V(G)$ into k non-empty parts. With probability at least $1-\frac{1}{n^{26}}$ the weight of $\tilde{P}$ is at most $(1+\epsilon)$ times the weight of an optimum k-cut of G. By weight of $\tilde{P}$, we mean the total weight of the edges with endpoints in different parts.*

Overview of the Algorithm. A preliminary step of our algorithm is to run the 2-approximation algorithm [5], [6], [7] to get an estimate of the value of $\mathsf{OPT}(G,k)$ (the weight of an optimal k-cut in G). A standard rounding procedure (similar to the well-known Knapsack PTAS [20]) reduces the problem in a $(1+\epsilon)$-approximation preserving manner to unweighted multi-graphs with at most m^2/ϵ edges. From now on, *throughout this paper*, we will assume that our input graph is an *unweighted multigraph*. As long as the graph has a non-trivial 2-cut (non-trivial means that there is at least one edge crossing the cut) of weight at most $\frac{\epsilon\cdot\mathsf{OPT}(G,k)}{k-1}$, we remove all edges of this 2-cut. This step will be repeated less than $k-1$ times, since if it is repeated $k-1$ times we have cut the graph into at least k connected components using at most $\epsilon\cdot\mathsf{OPT}(G,k)$ edges. Thus the procedure stops before this, and at that point we know that (i) we have included at most $\epsilon\cdot\mathsf{OPT}(G,k)$ edges in our solution, and (ii) the remaining graph has no non-trivial 2-cut with at most $\epsilon\cdot\mathsf{OPT}(G,k)$ edges. We now solve the problem independently on each connected component, after guessing how many parts each component is split into (this introduces a 4^k overhead in the running time). Thus from now on we will assume that our input graph is connected and has no non-trivial 2-cut of weight at most $\frac{\epsilon\cdot\mathsf{OPT}(G,k)}{k-1}$.

Having ensured that every non-trivial 2-cut has weight at least $\epsilon\cdot\frac{\mathsf{OPT}(G,k)}{k-1}$ we proceed as follows. We set a probability $p=\frac{100\log n}{\mathsf{OPT}(G,2)\cdot\epsilon^2}$. Then for every edge of G we flip a biased coin and keep the edge with probability p and delete it with probability $1-p$. Call the resulting subgraph G'. A relatively simple probability computation (very similar to that of randomized cut sparsifiers [21], [22]) combining Chernoff bounds with Karger and Stein's [2] bound on the number of minimum 2-cuts shows that with high probability it holds that for *every* partition $\tilde{P}$ of $V(G)=V(G')$ into k parts, the number of edges of G' crossing the partition $\tilde{P}$ is within a $1\pm\epsilon$ factor of p times the number of edges crossing it in G. Thus, for purposes of $(1+\epsilon)$-approximation one may just as well find a $(1+\epsilon)$-approximate solution in G'. However, in G' we know that

$$\begin{aligned}\mathsf{OPT}(G',k) &\simeq p\cdot\mathsf{OPT}(G,k)\\ &= \frac{100\log n}{\mathsf{OPT}(G,2)\cdot\epsilon^2}\cdot\mathsf{OPT}(G,k)\\ &\leq \frac{k\cdot 100\log n}{\epsilon^3}.\end{aligned}$$

In the last transition we used the assumption that $\mathsf{OPT}(G,2) \geq \epsilon\cdot\mathsf{OPT}(G,k)/k - 1$. This is a pretty small number of cut edges, which naturally leads to the idea of trying to apply parameterized algorithms with parameter s, the number of edges in the optimal k-cut. The sparsification procedure outlined above is captured by the following lemmas.

Lemma I.1*. [2] *There exists an algorithm that takes as input an integer $k > 0$, an unweighted graph G such that $\mathsf{cc}(G) < k$, and a real number $0 < \epsilon \leq 1$ and outputs a subgraph G_1 of G with $V(G') = V(G)$, such that $|E(G)|-|E(G_1)| \leq 2\epsilon\cdot\mathsf{OPT}(G,k)$. Further, if $\mathsf{cc}(G_1) < k$, then each non-trivial 2-cut of G_1 has weight at least $\frac{\epsilon\cdot\mathsf{OPT}(G,k)}{k-1}$.*

Lemma I.2*. *There exists a polynomial time algorithm that takes as input an integer $k > 0$, an unweighted graph G with $\mathsf{cc}(G) < k$, and a real number $\frac{1}{n} < \epsilon \leq 1$ and outputs a subgraph G_2 with $V(G_2) = V(G)$, and a real number r such that with probability at least $1 - \frac{1}{n^{26}}$, for all k-cuts $\tilde{P}$ in G, $(1-\epsilon)\cdot w(G,\tilde{P}) \leq w(G_2,\tilde{P})\cdot r \leq (1+\epsilon)\cdot w(G_2,\tilde{P})$.*

The Min k-Cut problem is also quite well studied with $s = \mathsf{OPT}(G,k)$ as a parameter. A brute force algorithm trying all edge sets of size s solves the problem in time $\mathcal{O}(n^{2s+\mathcal{O}(1)})$. Marx [24] posed as an open problem in 2004 whether Min k-Cut is *fixed parameter tractable* when parameterized by s, that is whether the problem admits an algorithm with running time $f(s)n^{\mathcal{O}(1)}$ for any function f. Kawarabayashi and Thorup [25] resolved this problem in the affirmative, and obtained an algorithm with running time $s^{s^{\mathcal{O}(s)}}n^2$. Chitnis et al. [26] gave an algorithm with running time $s^{\mathcal{O}(s^2)}n^{\mathcal{O}(1)}$, improving the dependence on s from double exponential to single exponential. Finally, Cygan et al. [11] showed that the problem is solvable in time $s^{\mathcal{O}(s)}n^{\mathcal{O}(1)}$. Unfortunately, applying the algorithm of Cygan et al. [11] directly on G' yields an algorithm with running time $(\frac{k\log n}{\epsilon^3})^{\mathcal{O}(\frac{k\log n}{\epsilon^3})}$ which grows super-polynomially with n, even for constant ϵ and k. Even if one obtains a parameterized algorithm for Min k-Cut with running time $2^{\mathcal{O}(s)}n^{\mathcal{O}(1)}$ (which is an interesting open problem in itself), this would only lead to a $2^{\mathcal{O}(\frac{k\log n}{\epsilon^3})} = n^{\mathcal{O}(\frac{k}{\epsilon^3})}$ time algorithm, which is slower than the classic exact $n^{2(k-1)}$ time algorithm of Karger and Stein [2]. Thus, at a glance, the parameterized approach seems to come close, but hit a dead end. The key insight of our algorithm is that even though s is "small", that k is much smaller, and that the hard instances for parameter s seem to have a value of k close to s. This leads to the natural problem of whether it is possible to design a parameterized algorithm with parameters k and s that substantially outperforms the algorithm of Cygan et al. [11] when $k \ll s$. Our main technical contribution is precisely such an algorithm. We state this algorithm as a separate theorem.

Theorem I.2. *There exists an algorithm that given an unweighted multigraph G and integers k and s, determines whether G has a k-cut of weight at most s in times $s^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$.*

The algorithm of Theorem I.2 can easily be extended using self reduction to also produce a partition $\tilde{P}$ of G with weight at most s, if it exists. The algorithm of Theorem I.2 is based on the $s^{\mathcal{O}(s)}n^{\mathcal{O}(1)}$ time algorithm of Cygan et al. [11].

Just as the algorithm of Cygan et al. [11], we compute a tree decomposition of the input graph G, such that the adhesions (cuts) of the decomposition are small, and the bags (the building blocks of the decomposition) are highly edge-connected. Unfortunately we are not able to use their decomposition theorem as a black box, because their running time to compute the decomposition is already $s^{\mathcal{O}(s)}n^{\mathcal{O}(1)}$. We make our own decomposition theorem with weaker properties than the one in [11], but computable in polynomial time. The construction of the decomposition theorem follows the template of Cygan et al. [11], replacing their most computationally intensive step with a polynomial time approximation algorithm. In particular we have the following theorem.

Theorem I.3*. *Given an n-vertex graph G and an integer s, one can in polynomial time compute a rooted compact tree decomposition (τ,χ) of G such that*

1) *every adhesion of (τ,χ) is of size at most s;*
2) *every bag of (τ,χ) is $((s+1)^5, s)$-edge-unbreakable.*

The most technically challenging part of our algorithm is how to compute an optimal k-cut in time $s^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$, when the decomposition into highly connected pieces is provided as input. The only known way how to exploit such a decomposition is using dynamic programming. However, the dynamic programming seems to require at least $2^{\Omega(s)}$ states, even for the stronger decomposition of Cygan et al. [11], let alone for our weaker structural decomposition theorem. Here, the fact that k is much smaller than s comes to rescue. We prove that the dynamic programming algorithm can be "trimmed" to only populate $s^{\mathcal{O}(k)}$ states of the dynamic programming table. This trimming procedure is done by inspecting how

[2] Proofs of theorems, lemmas, and claims labelled with $*$ have been omitted due to space constraints. For complete proofs, see the full version of the paper [23].

an optimal k-cut can interact with a tree obtained from Thorup's [3] tree-packing theorem. A substantial amount of technical weight lifting is required to show that the trimmed dynamic programming table for a bag of a tree decomposition can be computed efficiently from the trimmed dynamic programming tables of its children.

The algorithm of Theorem I.2 immediately completes the proof of our $(1+\epsilon)$-approximation algorithm (Theorem I.1). Specifically, applying this algorithm to the graph G' obtained by our sparsification procedure yields a $(1+\epsilon)$-approximation algorithm with running time

$$\begin{aligned}&\left(\frac{k\cdot 100\log n}{\epsilon^3}\right)^{\mathcal{O}(k)} n^{\mathcal{O}(1)}\\ &= (k/\epsilon)^{\mathcal{O}(k)}(\log n)^{\mathcal{O}(k)} n^{\mathcal{O}(1)}\\ &\leq (k/\epsilon)^{\mathcal{O}(k)}(k^{\mathcal{O}(k)}+n) n^{\mathcal{O}(1)}\\ &\leq 2^{\mathcal{O}(k\log(\frac{k}{\epsilon}))} n^{\mathcal{O}(1)}.\end{aligned}$$

Setting $\epsilon = \frac{1}{n^{\mathcal{O}(1)}}$, shows that a $2^{o(k\log(\frac{k}{\epsilon}))}n^{\mathcal{O}(1)}$ time algorithm would yield an exact algorithm with running time $n^{o(k)}$, violating ETH.

Guide to the paper. We start by giving the notations and preliminaries that we use throughout the paper in Section II. This section is best used as a reference, rather than being read linearly. Section III contains our main technical thrust: a new exact algorithm for unweighted multi graphs with running time $s^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$, the algorithm computes and uses a decomposition such as the one provided by Theorem I.3. We note that the other two parts of our main result, a $(1+\epsilon)$-approximation preserving reduction to instances with optimum k-cuts of size $\mathcal{O}(\frac{k\log n}{\epsilon^3})$, and Theorem I.3, the polynomial time edge-unbreakable decomposition theorem are deferred to the full version of the paper. Finally, in Section IV, we combine all the results obtained in the previous sections and complete the proof of our main result (Theorem I.1). We conclude the paper with some interesting open problems in Section V.

II. Notation and Preliminaries

In this section we give notations, and definitions that we use throughout the paper. Unless specified we will be using all general graph terminologies from the book of Diestel [27].

A. Graph and Set Theoretic Definitions and Notations

Given a graph G, we use $V(G)$ and $E(G)$ to denote the set of vertices and edges, respectively. We use $\mathsf{cc}(G)$ to denote the number of connected components of G. In this paper our graph could be a multigraph, that is, we allow parallel edges between vertices, even though we might start with a simple graph. Given a subset X of vertices in $V(G)$, we denote by $\delta(X)$ the number of edges in G that have exactly one end point in X. By the term *smoothing* a vertex v of degree 2 in a graph G, we mean the operation of removing v from the graph and adding an edge between two of its neighbors.

For a set U, an *ℓ-partition* of U is a family $\tilde{P} = \{P_1, P_2, \ldots, P_\ell\}$ of non-empty, pairwise disjoint sets whose union $\bigcup_{i=1}^{\ell} P_i$ is equal to U. A *partition* of U is an ℓ-partition for some positive integer ℓ. We refer to $P_1, \ldots, P_\ell$ as the *parts* of $\tilde{P}$ and refer to ℓ as the size of the partition and denote it by $|\tilde{P}|$. An edge uv with endpoints in U *crosses* $\tilde{P}$ if u and v are in different parts of $\tilde{P}$. The number of times an edge set X crosses P is defined as the number of edges in X that cross P and denoted by $\delta_X(\tilde{P})$. That is, $X' \subseteq X$ is the set containing all edges in X that crosses $\tilde{P}$ and $\delta_X(\tilde{P}) = |X'|$. The *projection* of a partition $\tilde{P}$ onto a subset S of U is the partition $\tilde{P}' = \{P_1 \cap S, P_2 \cap S, \ldots, P_\ell \cap S\}$ with the empty sets removed. A partition $\tilde{P}'$ is a *refinement* of $\tilde{P}$ if every part of $\tilde{P}'$ is a subset of some part of $\tilde{P}$, see Figure **??**(a). We say that a partition $\tilde{P}$ is *refined* by $\tilde{P}'$ if $\tilde{P}'$ is a refinement of $\tilde{P}$. The *combining* of a set of parts $\mathcal{P}$ in $\tilde{P}$ is the operation of removing all the parts in $\mathcal{P}$ from $\tilde{P}$ and adding their union $\bigcup_{P\in\mathcal{P}} P$ as a part in $\tilde{P}$.

A *k-cut* $\tilde{S}$ of a graph G is a k-partition of G. Let G be an edge weighted graph G, then the *weight of the k-cut*, denoted by $w(G,\tilde{S})$ is simply the sum of the weights of edges with endpoints in different parts of $\tilde{S}$. When G is an unweighted multigraph, then the weight of $\tilde{S}$ is the number of edges with endpoints in different parts of $\tilde{S}$ (if there are q edges between a pair of vertices then it adds q to the sum). We denote this also by $w(G,\tilde{S})$. In the cases where the graph G is clear from the context, we just use $w(\tilde{S})$ instead of $w(G,\tilde{S})$. An *optimal k-cut* or a *minimum k-cut* of G is a k-cut of G that has the *minimum weight* among all k-cuts of G. We denote the weight of an optimal k-cut by $\mathsf{OPT}(G,k)$. A *non-trivial cut* means that there is at least one edge crossing the cut. We remark that in some parts of the paper, we refer to a k-cut of a graph G as a k-partition of $V(G)$ but it will be clear from the context.

For a graph G, *an edge cut* is a pair $A, B \subseteq V(G)$ such that $A\cup B = V(G)$ and $A\cap B = \emptyset$. The order of an edge cut (A,B) is $|E(A,B)|$, that is, the number of edges with one endpoint in A and the other in B. Observe that edge cut and 2-cut are the same.

Definition II.1 (unbreakability)**.** *A set $X \subseteq V(G)$ is (q,s)-*edge-unbreakable *if every edge cut (A,B) of order at most s satisfies $|A\cap X| \leq q$ or $|B\cap X| \leq q$.*

B. T-tree and Treewidth

An important notion that underlies our algorithm is a notion of *T-trees* introduced in the seminal work of Thorup [3]. For an instance (G,k) of Min k-Cut, a *T-tree* of G is a spanning tree of G such that there

exists an optimal k-cut $S^\star$ of G such that T crosses $S^\star$ at most $2k-2$ times.

Another important notion that we need is of tree decomposition where bags are "highly connected". Towards this we first define tree decomposition, treewidth and associated notions and notations that we make use of. For a rooted tree τ and vertex $v \in V(\tau)$ we denote by τ_v the subtree of τ rooted at v. We refer to the vertices of τ as nodes.

A *tree decomposition* of a graph G is a pair (χ, τ) where χ is a rooted tree and τ is a function from $V(\tau)$ to $2^{V(G)}$ such that the following three conditions hold.

(T1) $\bigcup_{t \in V(\tau)} \chi(t) = V(G)$;

(T2) For every $uv \in E(G)$, there exists a node $t \in \tau$ such that $\chi(t)$ contains both u and v; and

(T3) For every vertex $u \in V(G)$, the set $T_u = \{t \in V(\tau) : u \in \chi(t)\}$, i.e., the set of nodes whose corresponding bags contain u, induces a connected subtree of τ.

For every $t \in V(\tau)$ a set $\chi(t) \subseteq V(G)$, is called a *bag*. For a tree decomposition (τ, χ) fix an edge $e = tt' \in E(\tau)$. The deletion of e from τ splits τ into two trees τ_1 and τ_2, and naturally induces a separation (X_1, X_2) in G with $X_i := \bigcup_{t \in V(\tau_i)} \chi(t)$, which we henceforth call *the separation associated with* e. The set $A_{\tau,\chi}(e) := X_1 \cap X_2 = \chi(t) \cap \chi(t')$ is called the *adhesion* of e. We suppress the subscript if the decomposition is clear from the context.

For $s, t \in V(\tau)$ we say that s *is a descendant of* t or that t *is an ancestor of* s if t lies on the unique path from s to the root; note that a node is both an ancestor and a descendant of itself. For a node t that is not a root of τ, by $A_{\tau,\chi}(t)$ we mean the adhesion $A_{\tau,\chi}(e)$ for the edge e connecting t with its parent in τ. We extend this notation to $A_{\tau,\chi}(r) = \emptyset$ for the root r. Again, we omit the subscript if the decomposition is clear from the context. For brevity, for $t \in V(\tau)$, we use A_t to denote $A_{\tau,\chi}(t)$.

We define the following functions for convenience:

$$\gamma(t) = \bigcup_{q \text{ descendant of } t} \chi(q),$$
$$\alpha(t) = \gamma(t) \setminus A_t,$$
$$G_t = G[\gamma(t)]$$

We say that a rooted tree decomposition (τ, χ) of G is *compact* if for every node $t \in V(\tau)$ for which $A_t \neq \emptyset$ we have that $G[\alpha(t)]$ is connected and $N_G(\alpha(t)) = A_t$. We can extend the function χ to subsets of $V(\tau)$ in the natural way: for a subset $X \subseteq V(\tau)$, $\chi(X) = \bigcup_{x \in X} \chi(x)$. By $\mathsf{children}(t)$, we denote the set of children of t in τ. For each subset X of nodes in $V(\tau)$, we define $G_X = G[\cup_{t \in X} \chi(V(\tau_t))]$.

C. Splitters

Let $[n]$ denote the set of integers $\{1, \ldots, n\}$. We start by defining the notion of splitters.

Definition II.2 ([28]). *An (n, k, ℓ) splitter $\mathcal{F}$ is a family of functions from $[n] \to [\ell]$ such that for all $S \subseteq [n]$, $|S| = k$, there exists a fucntion $f \in \mathcal{F}$ that splits S evenly. That is, for all $1 \leq j, j' \leq \ell$, $|f^{-1}(j') \cap S|$ and $|f'^{-1}(j) \cap S|$ differ by at most one.*

We will need following algorithm to compute splitters with desired parameters.

Theorem II.1 ([28]). *For all $n, k \geq 1$ one can construct an (n, k, k^2) splitter family of size $k^{\mathcal{O}(1)} \log n$ in time $k^{\mathcal{O}(1)} n \log n$.*

The following lemma is a simple application of Theorem II.1 and is used as a subroutine in our new algorithm for Min k-Cut.

Lemma II.1*. *There exists an algorithm that takes as input a set S, two positive integers s_1 and s_2 that are less than $|S|$, and outputs a family $\mathcal{S}$ of subsets of S having size $\mathcal{O}((s_1+s_2)^{\mathcal{O}(s_1)} \log |S|)$ such that for any two disjoint subsets X_1 and X_2 of S of size at most s_1 and s_2, $\mathcal{S}$ contains a subset X that satisfies $X_1 \subseteq X$ and $X_2 \cap X = \emptyset$ in time $\mathcal{O}((s_1+s_2)^{\mathcal{O}(s_1)} |S|^{\mathcal{O}(1)})$.*

III. A $s^{\mathcal{O}(k)}$ Time Algorithm for Min k-cut on Unweighted Graphs

In this section we give our new algorithm for Min k-Cut parameterized by s and k. It is a dynamic programming algorithm over an edge unbreakable tree decomposition (Theorem I.3) and uses T-trees introduced by Thorup [3] crucially.

Let G, k, s be input to Min k-Cut. Towards our proof, in polynomial time, we first compute a rooted compact tree decomposition (τ, χ) of G, using Theorem I.3, such that

1) every adhesion of (τ, χ) is of size at most s;
2) every bag of (τ, χ) is $((s+1)^5, s)$-edge-unbreakable.

Let $\tilde{P}^*$ be an optimum k-cut of G. Secondly, we compute a family $\mathcal{T}$ of spanning trees of G such that there exists a tree $T^* \in \mathcal{T}$ such that $E(T^*)$ crosses $\tilde{P}^*$ at most $2k-2$ times. Recall that T^* is said to be a T-tree of G. Such a family can be computed in polynomial time using Thorup's tree packing algorithm [3].

Finally, we assume that the input graph G is connected. Else, if $\mathsf{cc}(G) \geq k$, we return the connected components of G itself as the desired partition of G of value 0. Otherwise, if $\mathsf{cc}(G) \leq k$, then at the expense of $k^k s^k$ in time we can guess the values of k and s for each connected component.

Thus, throughout this section, we assume that we are given (τ, χ) and a tree T from $\mathcal{T}$ and remark that the notations $G, k, s, (\tau, \chi), T$ have fixed meanings. Also recall that in some parts, we refer to a l-cut of a graph G' as a l-partition of $V(G')$ but it will be clear from the context. We begin by defining additional terminologies

and proving some results that will helpful in defining the states of the dynamic programming.

Definition III.1. *For a tree T' and an integer c, a partition $\tilde{P}$ of $V(T')$ is* c-admissible *with respect to T' if $E(T')$ crosses $\tilde{P}$ at most c times. For a subset X of $V(T')$ a partition $\tilde{P}$ of X is* T'-feasible *if $\tilde{P}$ is the projection onto X of a $2k-2$-admissible partition of $V(T')$.*

We will denote the family of all T'-feasible partitions of X by $\mathcal{F}^X_{T'}$. Further, if $X = \emptyset$ then $\mathcal{F}^X_{T'}$ will contain the partition $P_\emptyset$, where $P_\emptyset$ is the partition of X that contains a single empty part.

Given a tree T', we define the partition of $V(T')$ obtained by removing a subset E' of edges from T' as *the partition of $V(T')$* in which each part is equal to the set of vertices in some tree in the forest obtained by deleting E' from T'.

Given a tree T' and $X \subseteq V(T')$, we define the *projection of T' onto X* as the tree $\mathsf{proj}(T', X)$ obtained by the following procedure. Initially, set $T'' = T'$, and exhaustively apply the following modifications to T'': (a) Delete from T'' all leaves of T'' that are not in X. (b) Smooth all degree 2 vertices v of T'' that are not in X. Once neither one of the operations (a), (b) can be applied the procedure returns T'' as the projection $\mathsf{proj}(T', X)$. An example of $\mathsf{proj}(\mathsf{T}', \mathsf{X})$ is depicted in Figure 1.

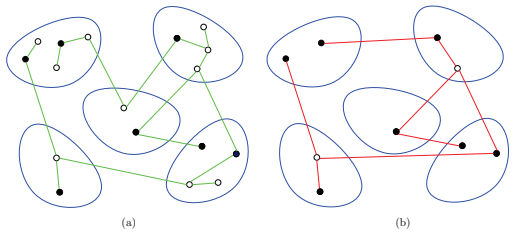

Fig. 1. (a) Tree T' is depicted by green edges and the set X by black vertices. (b) $\mathsf{proj}(\mathsf{T}', \mathsf{X})$ is depicted by red edges.

For the projection $\mathsf{proj}(T', X)$, we will denote its vertices and edges by $V^X_{T'}$ and $E^X_{T'}$ respectively. We encapsulate some useful properties of $\mathsf{proj}(T', X)$ in the following observation.

Observation III.1. *All leaves and degree 2 vertices of $\mathsf{proj}(T', X)$ are in X. Furthermore, the number of vertices in $\mathsf{proj}(T', X)$ is at most $2|X|$.*

The first part of the observation follows directly from the definition of the projection procedure, while the second follows from the well-known fact that in a tree, the number of vertices of degree at least three is at most the number of leaves. The following lemma shows that the set of T-feasible partitions of a set X does not change when T is projected on X. Thus, for enumerating T-feasible partitions of X it will be sufficient to work with the projection of T onto X instead, which is much more efficient. Since the proof of the Lemma III.1 is fairly straight forward, we state the lemma without proof and refer the reader to the full version of the paper for the proof.

Lemma III.1*. *For all $X \subseteq V(T)$, $\mathcal{F}^X_T = \mathcal{F}^X_{\mathsf{proj}(\mathsf{T},\mathsf{X})}$, Further $\mathcal{F}^X_T$ is of size $\mathcal{O}((4k|X|)^{2k})$ and can be computed in time proportional to its size.*

For the rest of this section, unless stated otherwise, we will assume that the weight of any partition of a set of vertices $X \subseteq V$ is computed with respect to $G[X]$.

We now define the states of the dynamic programming algorithm over the tree τ that we will use to compute the required value. For each node $t \in V(\tau)$ we define a function $f_t \colon \mathcal{F}^{A_t}_T \times \{1, \ldots, k\} \longrightarrow \{0, \ldots, s\} \cup \{\infty\}$ that our algorithm will compute. The domain of f_t consists of all pairs $(\tilde{P}_{A_t}, i)$ where $\tilde{P}_{A_t}$ is a T-feasible (meaning crosses T at most $2k-2$ times) partition of A_t, where A_t is the set of vertices that are common between node t and its parent in τ, and i is a positive integer less than or equal to k. On input $(\tilde{P}_{A_t}, i)$, f_t returns the smallest possible weight of a T-feasible i-partition $\tilde{P}$ of $V(G_t)$ such that $\tilde{P}_{A_t}$ is the projection of $\tilde{P}$ on A_t. However, if this weight is greater than s, or no such partition exists then $f_t(\tilde{P}_{A_t}, i)$ returns ∞. The main step of the algorithm of Theorem I.2 is an algorithm that computes $f_t(\tilde{P}_{A_t}, i)$ for every node t and pair $(\tilde{P}_{A_t}, i) \in \mathcal{F}^{A_t}_T \times \{1, \ldots, k\}$ assuming that the values of $f_{t'}$ have already been computed for all children t' of t. We now state that this step can be carried out efficiently.

Lemma III.2*. *There exists an algorithm that takes as input (τ, χ), a node $t \in V(\tau)$, a T-feasible partition $\tilde{P}_{A_t}$ of A_t and a positive integer $i \leq k$, together with the value of $f_{t'}(\tilde{P}'_{A_{t'}}, i')$ for every child t' of t, T-feasible partition $\tilde{P}'_{A_{t'}}$ of $A_{t'}$, and positive integer $i' \leq i$, and outputs $f_t(\tilde{P}_{A_t}, i)$ in time $s^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.*

Since a T-tree is a spanning tree of G such that there exists an optimal k-cut of G that is T-feasible, it follows that if T is a T-tree of G then $\mathsf{OPT}(G, k)$ is given by $f_r(P_\emptyset, k)$, where r is the root of τ, recall that $A_r = \emptyset$. If T is not a T-tree of G, then $f_r(P_\emptyset, k)$ is the weight of some k-cut of G of weight at most s or ∞ and thus $f_r(P_\emptyset, k) \geq \mathsf{OPT}(G, k)$. Therefore, we will now prove Theorem I.2 using Lemma III.2.

Proof of Theorem I.2. As explained in the beginning of the section, we assume that (a) the graph is connected, (b) we are given (τ, χ), satisfying properties mentioned in Theorem I.3 and (c) a tree T from $\mathcal{T}$ [3]. We design an algorithm that takes all these inputs and works as follows. Starting from the leaf nodes, in a bottom up manner, for each node $t \in \tau$, the algorithm computes $\mathcal{F}^{A_t}_T$ using Lemma III.1 and obtains the value of $f_t(\tilde{P}_{A_t}, i)$ for all pairs $(\tilde{P}_{A_t}, i) \in$

$\mathcal{F}_T^{A_t} \times \{1, \ldots, k\}$ using Lemma III.2. Finally it returns the value $f_r(P_\emptyset, k)$, where r is the root of τ.

Since $f_r(P_\emptyset, k) \geq \mathsf{OPT}(G, k)$ for any tree spanning tree T of G and $f_r(P_\emptyset, k) = \mathsf{OPT}(G, k)$ if T is a T-tree, the algorithm returns the desired output.

It is easy to see that $\mathcal{F}_T^{A_t}$ is of size $\mathcal{O}((4ks)^{2k})$ which is $s^{\mathcal{O}(k)}$ and can be computed in time $s^{\mathcal{O}(k)}$ from Lemma III.1 and the fact that $|A_t| \leq s$ for all $t \in \tau$. Thus the domain of f_t is of size $s^{\mathcal{O}(k)} \cdot k$ and each state can be computed in time $s^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ from Lemma III.2. Thus the algorithm runs in time $s^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. This completes the proof. □

For proving Lemma III.2, we will design an algorithm that takes all the inputs mentioned in Lemma III.2 and returns a positive integer v, which is the weight of some T-feasible i-partition $\tilde{P}$ of $V(G_t)$ having weight at most s such that $\tilde{P}_{A_t}$ is the projection of $\tilde{P}$ on A_t. If no such partition exists, the algorithm will return $v = \infty$. This automatically ensures that $f_t(\tilde{P}_{A_t}, i) \leq v$. The difficult part is to ensure that $f_t(\tilde{P}_{A_t}, i) \geq v$. If $f_t(\tilde{P}_{A_t}, i) = \infty$, this inequality trivially holds, and so it suffices to prove it for the cases when $f_t(\tilde{P}_{A_t}, i)$ is finite and a T-feasible i-partition $\tilde{P}$ of $V(G_t)$ such that $\tilde{P}_{A_t}$ is the projection of $\tilde{P}$ on A_t and having weight at most s exists. The proof of Lemma III.2 spans the remainder of this section and follows the route of "randomized contraction style Dynamic Programming" [26], [11] but is somewhat more complicated because at every step we need to use the T-tree T to speed up computations. We begin by giving names to some of the central objects of this proof. The notation for these tools will be used throughout the section and are summarized in Figure 2.

Let $\tilde{P}(\tilde{P}_{A_t}, i)$ be a smallest possible weight T-feasible i-partition of $V(G_t)$ such that $\tilde{P}_{A_t}$ is the projection of $\tilde{P}(\tilde{P}_{A_t}, i)$ on A_t having weight at most s. We say that $\tilde{P}(\tilde{P}_{A_t}, i)$ realises $f_t(\tilde{P}_{A_t}, i)$. We introduce the following notations assuming such a partition exists.

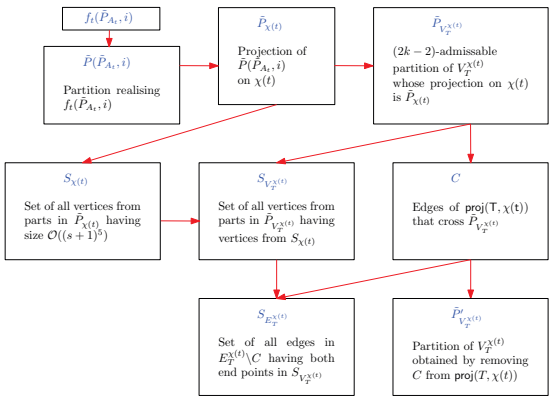

Fig. 2. Various notations associated with the state $f_t(\tilde{P}_{A_t}, i)$ along with their dependencies.

Let $\tilde{P}_{\chi(t)}$ be the projection of $\tilde{P}(\tilde{P}_{A_t}, i)$ onto $\chi(t)$. Since $\tilde{P}(\tilde{P}_{A_t}, i)$ is a T-feasible i-partition of $V(G_t)$, $\tilde{P}_{\chi(t)}$ must be a T-feasible partition of $\chi(t)$. Recall that the vertices and edges of $\mathsf{proj}(\mathsf{T}, \chi(\mathsf{t}))$ are denoted by $V_T^{\chi(t)}$ and $E_T^{\chi(t)}$ respectively. An example of $\chi(t)$ along with A_t and $\mathsf{proj}(T, \chi)$ is illustrated in Figure 3. From Lemma III.1 it follows that $\tilde{P}_{\chi(t)}$ is

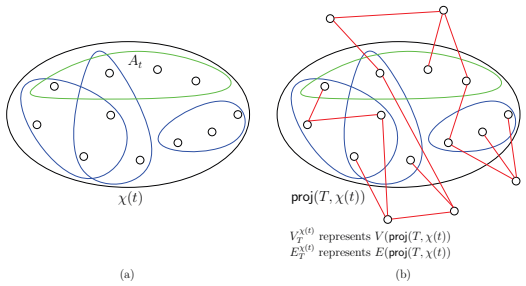

Fig. 3. (a) An example of a bag $\chi(t)$ in (τ, χ) along with the adhesions (blue) associated with it, the adhesion A_t is in green. (b) An example of $\mathsf{proj}(T, \chi(t))$ for the bag $\chi(t)$ in (a).

the projection onto $\chi(t)$ of some $2k-2$-admissible partition $\tilde{P}_{V_T^{\chi(t)}}$ of $V_T^{\chi(t)}$. Let C be the set of edges of $\mathsf{proj}(\mathsf{T}, \chi(\mathsf{t}))$ that cross $\tilde{P}_{V_T^{\chi(t)}}$, it is easy to see that $|C| \leq 2k-2$. Let $\tilde{P}'_{V_T^{\chi(t)}}$ be the partition of $V_T^{\chi(t)}$ obtained by deleting the edges in C from $\mathsf{proj}(\mathsf{T}, \chi(\mathsf{t}))$. Observe that $\tilde{P}'_{V_T^{\chi(t)}}$ is a refinement of $\tilde{P}_{V_T^{\chi(t)}}$. We remark that $\tilde{P}'_{V_T^{\chi(t)}}$ and $\tilde{P}_{V_T^{\chi(t)}}$ are different objects, and that this difference is crucial to our arguments (we apologize for the notational similarity!).

Since (τ, χ) is an $((s+1)^5, s)$-unbreakable tree decomposition, each part in $\tilde{P}_{\chi(t)}$ is of size $\mathcal{O}((s+1)^5)$ except for at most one part. Let $S_{\chi(t)}$ denote the set of all vertices in parts having size $\mathcal{O}((s+1)^5)$ in $\tilde{P}_{\chi(t)}$. The set of vertices from $\chi(t)$ in each part in $\tilde{P}_{V_T^{\chi(t)}}$ are either all from $S_{\chi(t)}$ or from $\chi(t) \setminus S_{\chi(t)}$ as $\tilde{P}_{\chi(t)}$ is the projection of $\tilde{P}_{V_T^{\chi(t)}}$ on $\chi(t)$. Let the union of the vertices in parts in $\tilde{P}_{V_T^{\chi(t)}}$ that contain vertices from $S_{\chi(t)}$ be denoted by $S_{V_T^{\chi(t)}}$, that is $S_{V_T^{\chi(t)}} = \bigcup_{P \in \tilde{P}_{V_T^{\chi(t)}}, P \cap \chi(t) \subseteq S_{\chi(t)}} P$. Observe that $S_{\chi(t)} \subseteq S_{V_T^{\chi(t)}}$. Let the set of edges in $E_T^{\chi(t)} \setminus C$ having both end points in $S_{V_T^{\chi(t)}}$ be denoted by $S_{E_T^{\chi(t)}}$. We now state a lemma that will help bound the sizes of the sets $S_{E_T^{\chi(t)}}$, and $S_{V_T^{\chi(t)}}$.

Lemma III.3. *The size of each part P in the partition $\tilde{P}'_{V_T^{\chi(t)}}$ is at most $2(|P \cap \chi(t)| + 2k - 2)$. Furthermore the sets $S_{E_T^{\chi(t)}}$ and $S_{V_T^{\chi(t)}}$ are of size at most $2 \cdot (2k-1)(|S_{\chi(t)}| + 2k - 2)$.*

Proof. Since $\tilde{P}'_{V_T^{\chi(t)}}$ is the partition of $V_T^{\chi(t)}$ obtained by removing the edges in C from $\mathsf{proj}(\mathsf{T}, \chi(\mathsf{t}))$, each part P in it can be associated with a tree T_P in the

forest obtained by removing C from $\mathsf{proj}(\mathsf{T}, \chi(\mathsf{t}))$. The number of vertices from $V_T^{\chi(t)} \setminus \chi(t)$ that are a leaf or a degree two vertex in T_P is at most $2k-2$ because from Observation III.1 they had degree more than two in $\mathsf{proj}(\mathsf{T}, \chi(\mathsf{t}))$ and thus must have been adjacent to an edge in C. The number of vertices in P having degree greater than two in T_P is bounded by the number of leaves in T_P which is bounded by $|P \cap \chi(t)| + 2k - 2$ because every leaf of T_P is either in $\chi(t)$ or not in $\chi(t)$. Thus, the number of vertices from $V_T^{\chi(t)} \setminus \chi(t)$ in P is bounded by $|P \cap \chi(t)| + 2 \cdot (2k-2)$ and hence the size of P is at most $2 \cdot (|P \cap \chi(t)| + 2k - 2)$.

Since the size of C is at most $2k-2$, there are at most $2k-1$ parts in $\tilde{P}'_{V_T^{\chi(t)}}$ and thus the sizes of $S_{E_T^{\chi(t)}}$ and $S_{V_T^{\chi(t)}}$ are at most $2 \cdot (2k-1)(|S_{\chi(t)}| + 2k - 2)$. □

Since $S_{\chi(t)}$ is of size $\mathcal{O}(k(s+1)^5)$, the following observation directly follows from Lemma III.3

Observation III.2. *The set C is of size at most $2k-2$ and the sets $S_{E_T^{\chi(t)}}$ and $S_{V_T^{\chi(t)}}$ are of size $\mathcal{O}(k^2(s+1)^5)$.*

In later parts of this section, we develop an algorithm that takes as input a subset C' of edges of $\mathsf{proj}(T, \chi(t))$ and outputs a positive integer v greater than or equal to $f_t(\tilde{P}_{A_t}, i)$. Further if $C \subseteq C'$ and $S_{E_T^{\chi(t)}} \cap C' = \emptyset$, then v will be equal to $f_t(\tilde{P}_{A_t}, i)$. We state this as a result below in Lemma III.4 and will be using it to prove Lemma III.2.

Lemma III.4. *There exists an algorithm that takes as input (τ, χ), a node $t \in V(\tau)$, a T-feasible partition $\tilde{P}_{A_t}$ of A_t, a positive integer $i \leq k$, $\mathsf{proj}(\mathsf{T}, \chi(\mathsf{t}))$, a set of edges $C' \subseteq E_T^{\chi(t)}$, together with the value of $f_{t'}(\tilde{P}'_{A_{t'}}, i')$ for every child t' of t, T-feasible partition $\tilde{P}'_{A_{t'}}$ of $A_{t'}$, and positive integer $i' \leq i$, and outputs an integer $v \geq f_t(\tilde{P}_{A_t}, i)$ in time $s^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. Further if $f_t(\tilde{P}_{A_t}, i) \neq \infty$, $C \subseteq C'$, and $S_{E_T^{\chi(t)}} \cap C' = \emptyset$, then $v \leq f_t(\tilde{P}_{A_t}, i)$.*

Since the proof of Lemma III.2 is easy assuming Lemma III.4, we sketch it here. In the proof, we apply the well-known technique of splitters to obtain a family $\mathcal{C}$ of subsets of edges in $\mathsf{proj}(\mathsf{T}, \mathsf{X})$ such that if $f_t(\tilde{P}_{A_t}, i) \neq \infty$, there is a set C' in the family such that $C \subseteq C'$, and $S_{E_T^{\chi(t)}} \cap C' = \emptyset$. Then, we apply Lemma III.4 on each set in this family to obtain $f_t(\tilde{P}_{A_t}, i)$. In the rest of this section, we will prove Lemma III.4, the crux of our result.

A. Nice Decompositions of $\chi(t)$

We first begin by defining *nice decompositions* of $\chi(t)$, a structure that will be helpful for proving Lemma III.4.

Definition III.2. *Let $\tilde{P}'_{\chi(t)}$ be a partition of $\chi(t)$, $\tilde{Q}_{\chi(t)}$ be a refinement of $\tilde{P}'_{\chi(t)}$ and O be a part in $\tilde{P}'_{\chi(t)}$ or an empty set. The triple $(\tilde{P}'_{\chi(t)}, \tilde{Q}_{\chi(t)}, O)$ is called a nice decomposition of $\chi(t)$ if it satisfies the following properties:*

- *(i) If $O \neq \emptyset$, then the projection of $\tilde{Q}_{\chi(t)}$ on O is O, that is O is not refined by $\tilde{Q}_{\chi(t)}$ and if $O = \emptyset$, then $\tilde{P}'_{\chi(t)}$ has exactly one part.*
- *(ii) The size of the projection of $\tilde{Q}_{\chi(t)}$ on each part $P \in \tilde{P}'_{\chi(t)} \setminus O$ has at most $2k-1$ parts.*
- *(iii) There are no cross edges between the parts in the partition $\tilde{P}'_{\chi(t)} \setminus O$.*
- *(iv) There are no adhesions in the set $\{A_{t'} : t' \in \mathsf{children}(t) \text{ or } t' = t\}$ having vertices from more than one part in $\tilde{P}'_{\chi(t)} \setminus O$.*

We refer to O as the center of the nice decomposition.

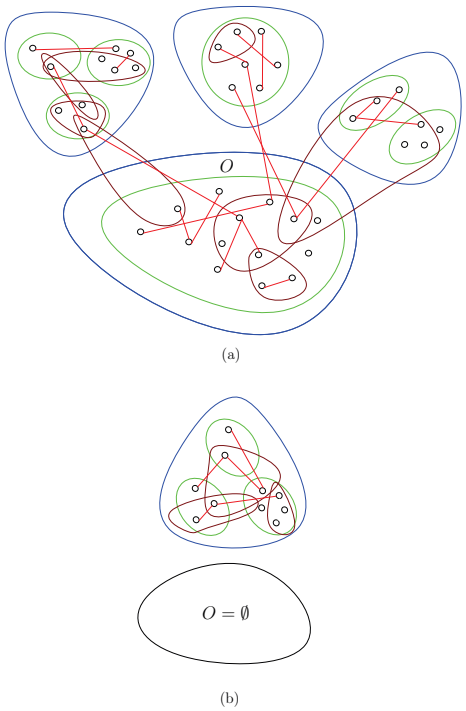

Fig. 4. Examples of nice decomposition $(\tilde{P}'_{\chi(t)}, \tilde{Q}_{\chi(t)}, O)$ of $\chi(t)$. (a) $O \neq \emptyset$ (b) $O = \emptyset$. The blue parts depict the parts in $\tilde{P}'_{\chi(t)}$ and the green parts depict those in $\tilde{Q}_{\chi(t)}$. Possible edges and adhesions among the vertices of $\chi(t)$ are depicted in red and dark red respectively. Note that there are no adhesions and edges among different parts in $\tilde{P}'_{\chi(t)} \setminus O$.

The structure of a nice decomposition of $\chi(t)$ is depicted by examples in Figure 4.

In order to prove Lemma III.4, we first state and prove some results related to nice decompositions of $\chi(t)$ that will help us with the proof. Firstly, we develop an algorithm that computes a *small* family of nice decompositions of $\chi(t)$ containing a special nice decomposition in desirable cases. Then, we design another algorithm that computes a value greater than $f_t(\tilde{P}_{A_t}, i)$ when input any nice decomposition and

computes $f_t(\tilde{P}_{A_t}, i)$ exactly if the input nice decomposition is special.

Lemma III.5. *There exists an algorithm that takes as input (τ, χ), a node $t \in V(\tau)$, $\mathsf{proj}(\mathsf{T}, \chi(\mathsf{t}))$, a set of edges $C' \subseteq E_T^{\chi(t)}$ and returns a set $\mathcal{D}$ of nice decompositions of $\chi(t)$ of size $\mathcal{O}(s^{\mathcal{O}(k)} \log n)$ in time $\mathcal{O}(s^{\mathcal{O}(k)} n^{\mathcal{O}(1)})$. Furthermore, if $f_t(\tilde{P}_{A_t}, i) \neq \infty$, $C \subseteq C'$, and $S_{E_T^{\chi(t)}} \cap C' = \emptyset$, then set $\mathcal{D}$ contains a nice decomposition $(\tilde{P}'_{\chi(t)}, \tilde{Q}_{\chi(t)}, O)$ such that $\tilde{Q}_{\chi(t)}$ is a refinement of $\tilde{P}_{\chi(t)}$.*

Proof. Given G, k, (τ, χ), $t \in V(\tau)$, $\mathrm{proj}(T, \chi(t))$, and $C' \subseteq E_T^{\chi(t)}$ we design an algorithm that returns a set of nice decompositions by carrying out the following steps :

(i) Set $\tilde{R}$ to be the partition of $V_T^{\chi(t)}$ obtained by removing the edges in C' from $proj(T, \chi(t))$.

(ii) If $\chi(t)$ is of size greater than $\mathcal{O}((s+1)^5)$, go to step (iii). Else, let $\tilde{P}'_{\chi(t)}$ be the partition of $\chi(t)$ having all vertices in $\chi(t)$ in one part and let $\tilde{Q}_{\chi(t)}$ be the projection of $\tilde{R}$ on $\chi(t)$. If the number of parts in $\tilde{R}$ is less than or equal to $2k-1$, then return the set $\mathcal{D}$ of nice decompositions as $\{(\tilde{P}'_{\chi(t)}, \tilde{Q}_{\chi(t)}, \emptyset)\}$, else return $\mathcal{D} = \emptyset$.

(iii) Let S be the set of all parts in $\tilde{R}$. Use Lemma II.1 on S, $s1 = 2k-1$ and $s_2 = (2k-1)(s^2+2s+k)$ and obtain a set $\mathcal{S}$ of subsets of S.

(iv) Construct a set $\mathcal{D}$ of nice decompositions using the following procedure: Initialize $\mathcal{D} = \emptyset$ and for each $X \in \mathcal{S}$, repeat the following steps:

a) Initialize partition $\tilde{Q} = \tilde{R}$.
b) Combine all parts in $\tilde{Q}$ that are not in X. Let this part be Q_1.
c) Form a graph G' with the parts in $\tilde{Q}$ except Q_1 as vertices. Add an edge between two parts in G' if they have a cross edge between them or both parts have non empty intersection with some adhesion $A_{t'}$, $t' \in \mathsf{children}(t) \cup t$.
d) Initialize $\tilde{Q}' = \tilde{Q}$, combine all parts in connected components in G' having more than $2k-1$ parts with Q_1 in $\tilde{Q}'$. Call this part O'.
e) Initialize $\tilde{P}' = \tilde{Q}'$. For each connected component Y in G' having at most $2k-1$ parts, combine all parts in Y in $\tilde{P}'$.
f) Let $\tilde{Q}_\chi(t)$ be the projection of $\tilde{Q}'$ on $\chi(t)$ and let $\tilde{P}'_{\chi(t)}$ be the projection of $\tilde{P}'$ on $\chi(t)$. Let O be the projection of O' on $\chi(t)$.
g) Add the pair $(\tilde{P}'_{\chi(t)}, \tilde{Q}_{\chi(t)}, O)$ to $\mathcal{D}$ if $O \neq \emptyset$.

(v) Return $\mathcal{D}$.

Now, we state an important claim that talks about $\mathcal{D}$. The key ingredients of the proof includes splitters, the properties of nice decompositions stated in Definition III.2, and the fact that the tree decomposition we are using is unbreakable and compact. For the full proof, we refer the reader to the full version.

Claim III.1*. *Each triple $(\tilde{P}'_{\chi(t)}, \tilde{Q}_{\chi(t)}, O) \in \mathcal{D}$ is a nice decomposition of $\chi(t)$. Further if $f_t(\tilde{P}_{A_t}, i) \neq \infty$, $C \subseteq C'$, and $S_{E_T^{\chi(t)}} \cap C' = \emptyset$, then $\mathcal{D}$ contains a nice decomposition $(\tilde{P}'_{\chi(t)}, \tilde{Q}_{\chi(t)}, O)$ such that $\tilde{Q}_{\chi(t)}$ is a refinement of $\tilde{P}_{\chi(t)}$.*

Claim III.2. *$\mathcal{D}$ is of size $\mathcal{O}(s^{\mathcal{O}(k)} \log n)$ and is computed in time $\mathcal{O}(s^{\mathcal{O}(k)} n^{\mathcal{O}(1)})$.*

Proof. In the case when the size of $\chi(t)$ is of $\mathcal{O}((s+1)^5)$, $\mathcal{D}$ is of size at most one and is computed in polynomial time. In the other case, $\mathcal{S}$ is of size $\mathcal{O}(s^{\mathcal{O}(k)} \log n)$ from Lemma II.1. Since for each $X \in \mathcal{S}$ we add at most one nice decomposition to set $\mathcal{D}$ in step (vi), $\mathcal{D}$ is of size $\mathcal{O}(ks)^{\mathcal{O}(k)} \log n)$. Time taken to compute $\mathcal{S}$ in step (v) is $\mathcal{O}(s^{\mathcal{O}(k)} n^{\mathcal{O}(1)})$ from Lemma II.1. All other steps take polynomial time, in particular, for each set $X \in \mathcal{S}$ it takes polynomial time to compute $\tilde{P}'_{\chi(t)}$, $\tilde{Q}_{\chi(t)}$ and O and add it to $\mathcal{D}$. Thus, the time taken to compute $\mathcal{D}$ in this case is $\mathcal{O}(s^{\mathcal{O}(k)} n^{\mathcal{O}(1)})$. □

Claims III.1 and III.2 complete the proof of Lemma III.5. □

With this, we start the second part of our section, an algorithm that takes as input a nice decomposition of $\chi(t)$ along with other inputs and outputs a value greater than or equal to $f_t(\tilde{P}_{A_t}, i)$. Further if the partition $\tilde{Q}_{\chi(t)}$ in the nice decomposition satisfies a special property, then the algorithm will output $f_t(\tilde{P}_{A_t}, i)$. We now state this as a Lemma.

Lemma III.6. *There exists an algorithm that takes as input (τ, χ), a node $t \in V(\tau)$, a T-feasible partition $\tilde{P}_{A_t}$ of A_t, a positive integer $i \leq k$, a nice decomposition $(\tilde{P}'_{\chi(t)}, \tilde{Q}_{\chi(t)}, O)$ of $\chi(t)$, together with the value of $f_{t'}(\tilde{P}'_{A_{t'}}, i')$ for every child t' of t, T-feasible partition $\tilde{P}'_{A_{t'}}$ of $A_{t'}$, and positive integer $i' \leq i$, and returns a positive integer v such that $f_t(\tilde{P}_{A_t}, i) \leq v$ or $v = \infty$ in time $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. Furthermore, if $\tilde{Q}_{\chi(t)}$ is a refinement of $\tilde{P}_{\chi(t)}$, then $v \leq f_t(\tilde{P}_{A_t}, i)$.*

To prove Lemma III.6, we design a dynamic programming algorithm on a given nice decomposition $(\tilde{P}'_{\chi(t)}, \tilde{Q}_{\chi(t)}, O)$ of $\chi(t)$. Let p denote the number of parts in $\tilde{P}'_{\chi(t)} \backslash O$, it is easy to see that by the definition of a nice decomposition, if $O = \emptyset$, then $p = 1$ and if not then $p = |\tilde{P}'_{\chi(t)}| - 1$. We arbitrarily order all the parts in $\tilde{P}'_{\chi(t)}$ except O and denote by P_l the l^{th} part in this order, where $l \geq 1$ and denote by $P_{\leq l} = \bigcup_{x \leq l} P_x$, the union of all parts P_x, $x \leq l$.

Given a non negative integer l that is less than or equal to p, we define the set $\mathcal{A}(l)$ to be the set of all adhesions in the set $\{A_{t'} : t' \in \mathsf{children}(t)\}$ that have vertices only from $P_l \cup O$ and have non empty intersection with P_l. Also, we denote by $\mathcal{A}_{\leq}(l) = \bigcup_{0 \leq l' \leq l} \mathcal{A}(l')$,

the union of all sets $\mathcal{A}(l')$ where $0 \leq l' \leq l$. Further, we define the graph $G(l) = G[O \cup P_l] \cup \bigcup_{A_{t'} \in \mathcal{A}(l)} G_{t'}$, to be the subgraph of G induced by all the vertices in $O \cup P_l \cup \bigcup_{A_{t'} \in \mathcal{A}(l)} V(G_{t'})$ and define the graph $G_{\leq}(l) = G[O \cup P_{\leq l}] \cup \bigcup_{A_{t'} \in \mathcal{A}_{\leq}(l)} G_{t'}$, to be the subgraph of G induced by all the vertices in $O \cup P_{\leq l} \cup \bigcup_{A_{t'} \in \mathcal{A}_{\leq}(l)} V(G_{t'})$.

We define two function $h, g : \{0, \ldots, p\} \times \{1, \ldots, i\} \longrightarrow \{0, \ldots, s\} \cup \{\infty\}$ that our algorithm will compute. The domain of h and g consists of all pairs (l, j) where l is a non negative integer less than or equal to p and j is a positive integer less than or equal to the input integer i. On input (l, j), h returns the smallest possible weight of a j-partition $\tilde{P}$ of $G(l)$ such that the projection of $\tilde{P}$ on $O \cup P_l$ is refined by the projection of $\tilde{Q}_{\chi(t)}$ on $O \cup P_l$ and if $A_t \subseteq O \cup P_l$ then $\tilde{P}_{A_t}$ is the projection of $\tilde{P}$ on A_t and g returns the smallest possible weight of a j-partition $\tilde{P}$ of $G_{\leq}(l)$ such that the projection of $\tilde{P}$ on $O \cup P_{\leq l}$ is refined by the projection of $\tilde{Q}_{\chi(t)}$ on $O \cup P_{\leq l}$ and if $A_t \subseteq O \cup P_{\leq l}$ then $\tilde{P}_{A_t}$ is the projection of $\tilde{P}$ on A_t. However, if this weight is greater than s, or no such partition exists then both $h(l, j)$ and $g(l, j)$ return ∞.

The main step of an algorithm for Lemma III.6 is an algorithm that computes $g(l, j)$ for every pair $(l, j) \in \{0, \ldots, p\} \times \{1, \ldots, i\}$ assuming that the value of $g(l', j')$ has been computed for every pair $(l', j') \in \{0, \ldots, l-1\} \times \{1, \ldots, i\}$. We now state that this step can be carried out.

Lemma III.7*. *There exists an algorithm that takes all the inputs of Lemma III.6, along with a non negative integer $l \leq p$, a positive integer $j \leq i$ and if $l \geq 1$ the value of $g(l-1, j')$ for every positive integer $j' \leq j$ and returns a positive integer $v = g(l, j)$ in time $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.*

Observe that $g(p, i)$ is the smallest possible weight of an i-partition $\tilde{P}$ of G_t such that $\tilde{Q}_{\chi(t)}$ is a refinement of the projection of $\tilde{P}$ on $\chi(t)$ and the projection of $\tilde{P}$ on A_t is $\tilde{P}_{A_t}$. By definition, $f_t(\tilde{P}_{A_t}, i)$ is the weight of the smallest possible i-partition whose projection on A_t is $\tilde{P}_{A_t}$, thus it follows that $g(p, i) \geq f_t(\tilde{P}_{A_t}, i)$. If $f_t(\tilde{P}_{A_t}, i) \neq \infty$ and $\tilde{Q}_{\chi(t)}$ is a refinement of $\tilde{P}_{\chi(t)}$ then $g(p, i) \leq f_t(\tilde{P}_{A_t}, i)$ since by definition, $\tilde{P}(\tilde{P}_{A_t}, i)$ is a i-partition of G_t that has weight $f_t(\tilde{P}_{A_t}, i)$, whose projection on $\chi(t)$ is $\tilde{P}_{\chi(t)}$, and whose projection on A_t is $\tilde{P}_{A_t}$. We state these relations between $g(p, i)$ and $f_t(\tilde{P}_{A_t}, i)$ in the following observation.

Observation III.3. *$g(p, i)$ is always greater than or equal to $f_t(\tilde{P}_{A_t}, i)$ and if $f_t(\tilde{P}_{A_t}, i) \neq \infty$ and $\tilde{Q}_{\chi(t)}$ is a refinement of $\tilde{P}_{\chi(t)}$ then $g(p, i)$ is less than or equal to $f_t(\tilde{P}_{A_t}, i)$.*

We are now ready to prove Lemma III.6 using Lemma III.7 and Observation III.3.

Proof of Lemma III.6 assuming Lemma III.7. For proving Lemma III.6 we propose the following algorithm. The algorithm will take inputs as specified in the Lemma, compute the function g using Lemma III.7 and finally returns an integer $v = g(p, i)$. From Observation III.3, it follows that the algorithm outputs a value $v \geq f_t(\tilde{P}_{A_t}, i)$. Further if $f_t(\tilde{P}_{A_t}, i) \neq \infty$ and $\tilde{Q}_{\chi(t)}$ is a refinement of $\tilde{P}_{\chi(t)}$ then $v \leq f_t(\tilde{P}_{A_t}, i)$. Since the domain of g has $p \cdot i$ values, $p \leq n$ and $i \leq k$, from Lemma III.7 it is easy to see that the algorithm runs in time $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. □

Since, $G_{\leq}(l) = G_{\leq}(l-1) \cup G(l)$ and all common edges of $G_{\leq}(l-1)$ and $G(l)$ are in $G(0)$, it follows that for any pair $(l, j) \in \{0, \ldots, p\} \times \{1, \ldots, i\}$ the following equation holds if $O \neq \emptyset$.

$$g(l, j) = \begin{cases} \min\limits_{1 \leq j' \leq j} g(l-1, j') + h(l, j - j' + 1), \; l \geq 1 \\ h(l, j), \; l = 0 \end{cases} \tag{1}$$

If $O = \emptyset$, then there is only one part in $\tilde{P}'_{\chi(t)}$ of the nice decomposition and thus we can compute $g(1, j) = h(1, j)$, for all $j \leq i$ directly.

The proof of Lemma III.7 has a knapsack style dynamic programming algorithm that computes $h(l, j')$ efficiently for all $j' \leq j$, we refer the reader to the full version for the proof. We now are ready to complete the proof of Lemma III.4 for which we obtained results throughout this section.

Proof of Lemma III.4. For the proof, we propose the following algorithm. Firstly, the algorithm will take inputs as specified in Lemma III.4 and obtain a set of nice decompositions $\mathcal{D}$ from Lemma III.5. Then for each nice decomposition $D \in \mathcal{D}$, the algorithm obtains a value v_D from Lemma III.6. Finally it returns the positive integer $v = \min\limits_{D \in \mathcal{D}} v_D$. If $\mathcal{D} = \emptyset$, then return $v = \infty$. Combining Lemma III.5 and Lemma III.6, it is easy to see that the algorithm will return $v \geq f_t(\tilde{P}_{A_t}, i)$. Further if $f_t(\tilde{P}_{A_t}, i) \neq \infty$, $C \subseteq C'$, and $S_{E_T^{\chi(t)}} \cap C' = \emptyset$, then it will return $v = f_t(\tilde{P}_{A_t}, i)$. The time taken by the algorithm is $\mathcal{O}((s)^{\mathcal{O}(k)} n^{\mathcal{O}(1)})$ since $\mathcal{D}$ has at most $\mathcal{O}((s)^{\mathcal{O}(k)} \log n)$ sets and each set can be computed in $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time from Lemma III.5 and Lemma III.6. This proves Lemma III.4 □

IV. Combining all the Pieces: Proof of Theorem I.1

In this section we conjure all the pieces we obtained so far and give a proof of our main result (Theorem I.1).

Proof of Theorem I.1. Let G, k, ϵ be the input to Min k-Cut. Also let $|V(G)| = n$. We will output a partition $\tilde{P}$ with weight v such that $v \leq (1+\epsilon)\mathsf{OPT}(G, k)$

with probability at least $1 - \frac{1}{n^{26}}$. We first check if $\mathsf{cc}(G) \geq k$. If this is true then we know that G has optimum k-cut of value 0 and we can return G itself. If $\epsilon < \frac{1}{n}$ then we run the best known exact algorithm on (G, k) running in time $n^{\mathcal{O}(k)}$ and return the exact answer [2], [3], [16]. Clearly, $n^{\mathcal{O}(k)}$ is $(k/\epsilon)^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ in this case. So from now onwards we assume that $\mathsf{cc}(G) \leq k$ and $\epsilon \geq \frac{1}{n}$. Also, we set $\epsilon' = \frac{\epsilon}{10}$.

Now we apply a standard rounding procedure (similar to the well-known Knapsack PTAS [20]) that reduces the problem in a $(1+\epsilon')$-approximation preserving manner to an unweighted multi-graph with at most m^2/ϵ edges. This implies that now the graph has at most n^5 edges (counting multiplicities). Let this graph be $G^\star$. Next we apply Lemma I.1 and obtain a subgraph G_1 of $G^\star$ with $V(G') = V(G^\star)$, such that $q = |E(G^\star)| - |E(G_1)| \leq 2\epsilon' \cdot \mathsf{OPT}(G^\star, k)$. Further, if $\mathsf{cc}(G_1) < k$, then each non-trivial 2-cut of G_1 has weight at least $\frac{\epsilon' \cdot \mathsf{OPT}(G^\star, k)}{k-1}$. If $\mathsf{cc}(G_1) = k$, then we return the connected components as a partition $\tilde{P}$. The cost of returned solution is

$$\begin{aligned} v &\leq 2\epsilon' \cdot \mathsf{OPT}(G^\star, k) \leq 2\epsilon' \cdot (1+\epsilon')\mathsf{OPT}(G, k) \\ &\leq (1+\epsilon)\mathsf{OPT}(G, k) \end{aligned}$$

Thus, we assume that $\mathsf{cc}(G_1) < k$ and each non-trivial 2-cut of G_1 has weight at least $\frac{\epsilon' \cdot \mathsf{OPT}(G^\star, k)}{k-1}$. Now we apply the sparsification procedure described in Lemma I.2 and obtain a subgraph G_2 with $V(G_2) = V(G_1)$, and a real number r such that with probability at least $1 - \frac{1}{n^{26}}$, for all k-cuts $\tilde{P}$ in G_1, $(1-\epsilon') \cdot w(G_1, \tilde{P}) \leq w(G_2, \tilde{P}) \cdot r \leq (1+\epsilon') \cdot w(G_1, \tilde{P})$. Here, $r = 1/p$, where $p = \frac{100 \cdot \log n}{\epsilon'^2 \cdot \mathsf{OPT}(G_1, 2)}$ However, in G_2 we know that

$$\begin{aligned} \mathsf{OPT}(G_2, k) &\leq (1+\epsilon') \cdot p \cdot \mathsf{OPT}(G_1, k) \\ &= \frac{(1+\epsilon')100 \log n}{\mathsf{OPT}(G_1, 2) \cdot \epsilon'^2} \cdot \mathsf{OPT}(G_1, k) \\ &\leq \frac{(1+\epsilon')100 \log n}{\mathsf{OPT}(G_1, 2) \cdot \epsilon'^2} \cdot \mathsf{OPT}(G^\star, k) \\ &\leq \frac{(1+\epsilon') \cdot k \cdot 100 \log n}{\epsilon'^3} \\ &\leq \frac{\beta \cdot k \cdot 100 \log n}{\epsilon^3}. \end{aligned}$$

Here, β is a fixed constant. Further, in the second last transition we used the assumption that $\mathsf{OPT}(G_1, 2) \geq \epsilon' \cdot \mathsf{OPT}(G^\star, k)/k - 1$. Now we apply the Theorem I.2 on G_2 and for each $s \in \left\{1, \ldots, \frac{\beta \cdot k \cdot 100 \log n}{\epsilon^3}\right\}$ and solve the problem exactly in time $s^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. Thus the running time of the algorithm is upper bounded by

$$\begin{aligned} &\left(\frac{\beta \cdot k \cdot 100 \log n}{\epsilon^3}\right)^{\mathcal{O}(k)} n^{\mathcal{O}(1)} \\ &= (k/\epsilon)^{\mathcal{O}(k)} (\log n)^{\mathcal{O}(k)} n^{\mathcal{O}(1)} \\ &\leq (k/\epsilon)^{\mathcal{O}(k)} (k^{\mathcal{O}(k)} + n) n^{\mathcal{O}(1)} \\ &\leq 2^{\mathcal{O}(k \log(\frac{k}{\epsilon}))} n^{\mathcal{O}(1)} \end{aligned}$$

This completes the running time analysis. All that remains is to show that what we get is an $(1+\epsilon)$-approximation algorithm. Let s be the minimum value for which the algorithm described in Theorem I.2 returns yes. This implies that $s = \mathsf{OPT}(G_2, k)$. Let $\tilde{P}$ be the corresponding partition. We return $s \cdot r + q$ as value of the cut and $\tilde{P}$ as a solution. Here, the value is the sum of edges deleted when we obtained G_1 from $G^\star$ and the value returned by Theorem I.2 when ran on (G_2, k, s). Now we have that

$$\begin{aligned} &\mathsf{OPT}(G_2, k) \cdot r + q \\ &\leq (1+\epsilon') \cdot \mathsf{OPT}(G_1, k) + 2\epsilon' \cdot \mathsf{OPT}(G^\star, k) \\ &\leq (1+\epsilon') \cdot \mathsf{OPT}(G^\star, k) + 2\epsilon' \cdot \mathsf{OPT}(G^\star, k) \\ &\leq (1+3\epsilon') \cdot \mathsf{OPT}(G^\star, k) \\ &\leq (1+3\epsilon') \cdot (1+\epsilon') \cdot \mathsf{OPT}(G, k) \\ &\leq (1+\epsilon) \cdot \mathsf{OPT}(G, k) \end{aligned}$$

The correctness of the algorithm follows from Lemmas I.1, I.2 and Theorem I.2. This concludes the proof. □

V. Conclusion

In this paper we gave a parameterized approximation algorithm with best possible approximation guarantee, and best possible running time dependence on said guarantee (upto ETH and constants in the exponent) for Min k-Cut. In particular, for every $\epsilon > 0$, the algorithm computes a $(1+\epsilon)$-approximate solution in time $(k/\epsilon)^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. Along the way we also obtained a new exact algorithm with running time $s^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ on unweighted (multi-) graphs, where s denotes the number of edges in a minimum k-cut. For an even more complete understanding of the parameterized apperoximation complexity of Min k-Cut one could explore the possibility of lossy kernels of polynomial size in s or k, and ratios between $2-\epsilon$ and $1+\epsilon$. Finally, an intriguing open problem about the parameterized complexity of Min k-Cut is whether the problem admits an algorithm with running time $2^{\mathcal{O}(s)} n^{\mathcal{O}(1)}$.

Acknowledgment

Daniel Lokshtanov: United States - Israel Binational Science Foundation grant no. 2018302.
Saket Saurabh: European Research Council (ERC) under the European Union's Horizon 2020 research and

innovation programme (grant no. 819416), and Swarnajayanti Fellowship grant DST/SJF/MSA01/2017-18.

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: https://doi.org/10.1287/moor.19.1.24 1

[2] D. R. Karger and C. Stein, "A new approach to the minimum cut problem," *J. ACM*, vol. 43, no. 4, pp. 601–640, 1996. [Online]. Available: https://doi.org/10.1145/234533.234534 1, 2, 3, 11

[3] M. Thorup, "Minimum k-way cuts via deterministic greedy tree packing," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, C. Dwork, Ed. ACM, 2008, pp. 159–166. [Online]. Available: https://doi.org/10.1145/1374376.1374402 1, 4, 5, 6, 11

[4] C. Chekuri, K. Quanrud, and C. Xu, "LP relaxation and tree packing for minimum k-cut," *SIAM J. Discret. Math.*, vol. 34, no. 2, pp. 1334–1353, 2020. [Online]. Available: https://doi.org/10.1137/19M1299359 1

[5] J. Naor and Y. Rabani, "Tree packing and approximating k-cuts," in *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*, S. R. Kosaraju, Ed. ACM/SIAM, 2001, pp. 26–27. [Online]. Available: http://dl.acm.org/citation.cfm?id=365411.365415 1, 2

[6] R. Ravi and A. Sinha, "Approximating k-cuts using network strength as a lagrangean relaxation," *European Journal of Operational Research*, vol. 186, no. 1, pp. 77–90, 2008. 1, 2

[7] H. Saran and V. V. Vazirani, "Finding k cuts within twice the optimal," *SIAM J. Comput.*, vol. 24, no. 1, pp. 101–108, 1995. [Online]. Available: https://doi.org/10.1137/S0097539792251730 1, 2

[8] P. Manurangsi, "Inapproximability of maximum biclique problems, minimum k-cut and densest at-least-k-subgraph from the small set expansion hypothesis," *Algorithms*, vol. 11, no. 1, p. 10, 2018. [Online]. Available: https://doi.org/10.3390/a11010010 1, 2

[9] R. G. Downey, V. Estivill-Castro, M. R. Fellows, E. Prieto-Rodriguez, and F. A. Rosamond, "Cutting up is hard to do: the parameterized complexity of k-cut and related problems," *Electron. Notes Theor. Comput. Sci.*, vol. 78, pp. 209–222, 2003. [Online]. Available: https://doi.org/10.1016/S1571-0661(04)81014-4 1, 2

[10] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*. Springer, 2015. [Online]. Available: https://doi.org/10.1007/978-3-319-21275-3 1, 2

[11] M. Cygan, P. Komosa, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, S. Saurabh, and M. Wahlstrom, "Randomized contractions meet lean decompositions," *CoRR*, vol. abs/1810.06864, 2018. [Online]. Available: http://arxiv.org/abs/1810.06864 2, 3, 7

[12] A. Gupta, E. Lee, and J. Li, "Faster exact and approximate algorithms for k-cut," in *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, M. Thorup, Ed. IEEE Computer Society, 2018, pp. 113–123. [Online]. Available: https://doi.org/10.1109/FOCS.2018.00020 2

[13] 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*, A. Czumaj, Ed. SIAM, 2018, pp. 2821–2837. [Online]. Available: https://doi.org/10.1137/1.9781611975031.179 2

[14] A. Gupta, E. Lee, and J. Li, "The number of minimum k-cuts: improving the karger-stein bound," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, M. Charikar and E. Cohen, Eds. ACM, 2019, pp. 229–240. [Online]. Available: https://doi.org/10.1145/3313276.3316395 2

[15] J. Li, "Faster minimum k-cut of a simple graph," in *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, D. Zuckerman, Ed. IEEE Computer Society, 2019, pp. 1056–1077. [Online]. Available: https://doi.org/10.1109/FOCS.2019.00068 2

[16] A. Gupta, E. Lee, and J. Li, "The number of minimum k-cuts: Improving the karger-stein bound," *To appear in STOC 2020*, vol. abs/1906.00417, 2020. [Online]. Available: http://arxiv.org/abs/1906.00417 2, 11

[17] K. Kawarabayashi and B. Lin, "A nearly 5/3-approximation FPT algorithm for min-k-cut," in *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, S. Chawla, Ed. SIAM, 2020, pp. 990–999. [Online]. Available: https://doi.org/10.1137/1.9781611975994.59 2

[18] A. Abboud, V. V. Williams, and O. Weimann, "Consequences of faster alignment of sequences," in *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, ser. Lecture Notes in Computer Science, J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, Eds., vol. 8572. Springer, 2014, pp. 39–51. [Online]. Available: https://doi.org/10.1007/978-3-662-43948-7_4 2

[19] A. Backurs and C. Tzamos, "Improving viterbi is hard: Better runtimes imply faster clique algorithms," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 2017, pp. 311–321. [Online]. Available: http://proceedings.mlr.press/v70/backurs17a.html 2

[20] J. M. Kleinberg and É. Tardos, *Algorithm design*. Addison-Wesley, 2006. 2, 11

[21] A. A. Benczúr and D. R. Karger, "Randomized approximation schemes for cuts and flows in capacitated graphs," *SIAM J. Comput.*, vol. 44, no. 2, pp. 290–319, 2015. [Online]. Available: https://doi.org/10.1137/070705970 2

[22] D. R. Karger, "Using randomized sparsification to approximate minimum cuts," in *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA*, D. D. Sleator, Ed. ACM/SIAM, 1994, pp. 424–432. [Online]. Available: http://dl.acm.org/citation.cfm?id=314464.314582 2

[23] D. Lokshtanov, S. Saurabh, and V. Surianarayanan, "A parameterized approximation scheme for min k-cut," *CoRR*, vol. abs/2005.00134, 2020. [Online]. Available: https://arxiv.org/abs/2005.00134 3

[24] D. Marx, "Parameterized graph separation problems," *Theor. Comput. Sci.*, vol. 351, no. 3, pp. 394–406, 2006. [Online]. Available: https://doi.org/10.1016/j.tcs.2005.10.007 3

[25] K. Kawarabayashi and M. Thorup, "The minimum k-way cut of bounded size is fixed-parameter tractable," in *52nd Annual IEEE Symposium on Foundations of Computer Science, FOCS, 2011, Palm Springs, CA, USA, October 22-25, 2011*. IEEE Computer Society, 2011, pp. 160–169. [Online]. Available: https://doi.org/10.1109/FOCS.2011.53 3

[26] 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: https://doi.org/10.1137/15M1032077 3, 7

[27] R. Diestel, *Graph Theory, 4th Edition*, ser. Graduate texts in mathematics. Springer, 2012, vol. 173. 4

[28] M. Naor, L. J. Schulman, and A. Srinivasan, "Splitters and near-optimal derandomization," in *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*. IEEE Computer Society, 1995, pp. 182–191. [Online]. Available: https://doi.org/10.1109/SFCS.1995.492475 5