Approximating k-Edge-Connected Spanning Subgraphs via a Near-Linear Time LP Solver

Parinya Chalermsook ✉
Aalto University, Espoo, Finland

Chien-Chung Huang ✉
École Normale Supérieure, Paris, France

Danupon Nanongkai ✉
University of Copenhagen, Denmark
KTH Royal Institute of Technology, Stockholm, Sweden

Thatchaphol Saranurak ✉
University of Michigan, Ann Arbor, MI, USA

Pattara Sukprasert ✉
Northwestern University, Evanston, IL, USA

Sorrachai Yingchareonthawornchai ✉
Aalto University, Espoo, Finland

──── **Abstract** ────

In the k-edge-connected spanning subgraph (kECSS) problem, our goal is to compute a minimum-cost sub-network that is resilient against up to k link failures: Given an n-node m-edge graph with a cost function on the edges, our goal is to compute a minimum-cost k-edge-connected spanning subgraph. This NP-hard problem generalizes the minimum spanning tree problem and is the "uniform case" of a much broader class of survival network design problems (SNDP). A factor of two has remained the best approximation ratio for polynomial-time algorithms for the whole class of SNDP, even for a special case of 2ECSS. The fastest 2-approximation algorithm is however rather slow, taking $O(mnk)$ time [Khuller, Vishkin, STOC'92]. A faster time complexity of $O(n^2)$ can be obtained, but with a higher approximation guarantee of $(2k-1)$ [Gabow, Goemans, Williamson, IPCO'93].

Our main contribution is an algorithm that $(1+\varepsilon)$-approximates the optimal fractional solution in $\tilde{O}(m/\varepsilon^2)$ time (independent of k), which can be turned into a $(2+\varepsilon)$ approximation algorithm that runs in time $\tilde{O}\left(\frac{m}{\varepsilon^2} + \frac{k^2 n^{1.5}}{\varepsilon^2}\right)$ for (integral) kECSS; this improves the running time of the aforementioned results while keeping the approximation ratio arbitrarily close to a factor of two.

1 Introduction

In the k-Edge-Connected Spanning Subgraph problem (kECSS), we are given an undirected n-node m-edge graph $G = (V, E)$ together with edge costs, and want to find a minimum-cost k-edge connected spanning subgraph.[1] For $k = 1$, this is simply the minimum spanning tree problem, and thus can be solved in $O(m)$ time [28]. For $k \geq 2$, the problem is a classical NP-hard problem whose first approximation algorithm was given almost four decades ago, where Frederickson and Jaja [19] gave a 3-approximation algorithm that runs in $O(n^2)$ time for the case of $k = 2$. The approximation ratio was later improved to 2 by an $\tilde{O}(mnk)$-time algorithm of Khuller and Vishkin [32].[2] This approximation factor of 2 has remained the best for more than 30 years, even for a special case of 2ECSS called the weighted tree augmentation problem. When the running time is of the main concern, the fastest known algorithm takes $O(n^2)$ time at the cost of a significantly higher $(2k - 1)$-approximation guarantee, due to Gabow, Goemans, and Williamson [22].

This above state-of-the-arts leave a big gap between algorithms achieving the best approximation ratio and the best time complexity. This gap exists even for $k = 2$. In this paper, we improve the running time of both aforementioned algorithms of [32, 22] while keeping the approximation ratio arbitrarily close to two. Our main contribution is a near-linear time algorithm that $(1 + \varepsilon)$-approximates the optimal *fractional* solution.

▶ **Theorem 1.** *For any $\varepsilon > 0$, there is a randomized $\tilde{O}(m/\varepsilon^2)$-time algorithm that outputs a $(1 + \varepsilon)$-approximate* fractional *solution for kECSS.*

Following, in the high-level, the arguments of Chekuri and Quanrud [7] (i.e. solving the minimum-weight k disjoint arborescences in the style of [32] on the support of the sparsified fractional solution), the above fractional solution can be turned into a fast $(2 + \varepsilon)$-approximation algorithm for the integral version of kECSS.

▶ **Corollary 2.** *For any $\varepsilon > 0$, there exist*
- *a randomized $\tilde{O}(m/\varepsilon^2)$-time algorithm that estimates the* value *of the optimal solution for kECSS to within a factor $(2 + \varepsilon)$, and*
- *a randomized $\tilde{O}\left(\frac{m}{\varepsilon^2} + \frac{k^2 n^{1.5}}{\varepsilon^2}\right)$-time algorithm that produces a feasible kECSS solution of cost at most $(2 + \varepsilon)$ times the optimal value.*

We remark that the term $\tilde{O}(k^2 n^{1.5})$ is in fact "tight" up to the state-of-the-art algorithm for finding minimum-weight k disjoint arborescences.[3]

Prior to our results, a sub-quadratic time algorithm was not known even for special cases of kECSS, called k-*Edge-Connected Augmentation* (kECA). In this problem, we are given a $(k - 1)$-edge-connected subgraph H of a graph G, and we want to minimize the total cost of adding edges in G to H so that H becomes k-edge connected. It is not hard to see that if we can α-approximates kECSS, then we can α-approximates kECA by assigning cost 0 to all

---

[1] Note that this problem should not be confused with a variant that allows to pick the same edge multiple time, which is sometimes also called kECSS (e.g., [6]). We follow the convention in [13] and call the latter variant minimum-cost k-edge connected spanning sub-multigraph (kECSSM) problem. (See also the work by Pritchard [39].)

[2] $\tilde{O}$ hides polylog(n) factor.

[3] More formally, if a minimum-weight union of k edge-disjoint arborescences can be found in time $T(k, m, n)$, then our algorithm would run in time $T(k, kn, n)$. The term $O(k^2 n^{1.5})$ came from Gabow's algorithm [20] that runs in time $O(km\sqrt{n}\log(nc_{\max}))$.

edges in H. This problem previously admits a $O(kn^2)$-time 2-approximation algorithm for any even integer k [31][4]. The approximation ratio of 2 remains the best even for 2ECA. Our result in Corollary 2 improves the previously best time complexity by a $\tilde{\Theta}(\sqrt{n})$ factor.

Perspective. The gap between algorithms with best approximation ratio and best time complexity in fact reflects a general lack of understanding on *fast approximation algorithms.* While polynomial-time algorithms were perceived by many as efficient, it is not a reality in the current era of large data, where it is nearly impossible to take $O(n^3)$ time to process a graph with millions or billions of nodes. Research along this line includes algorithms for sparsest cut [30, 29, 42, 35], multi-commodity flow [23, 17, 36], and travelling salesman problem [6, 7]. Some of these algorithms have led to exciting applications such as fast algorithms for max-flow [43], dynamic connectivity [38, 8, 40, 45, 37], vertex connectivity [34] and maximum matching [44].

The kECSS problem belongs to the class of survivable network design problems (SNDPs), where the goal is to find a subgraph ensuring that every pair of nodes (u, v) are $\kappa(u, v)$-edge-connected for a given function κ. (kECSS is the *uniform* version of SNDP where $\kappa(u, v) = k$ for every pair (u, v).) These problems typically focus on building a network that is resilient against device failures (e.g. links or nodes), and are arguably among the most fundamental problems in combinatorial optimization. Research in this area has generated a large number of beautiful algorithmic techniques during the 1990s, culminating in the result of Jain [26] which gives a 2-approximation algorithm for the whole class of SNDPs. Thus, achieving a *fast* 2-approximation algorithm for SNDPs is a very natural goal.

Towards this goal and towards developing fast approximation algorithms in general, there are two common difficulties:

1. Many approximation algorithms inherently rely on solving a *linear program (LP)* to find a fractional solution, before performing rounding steps. However, the state-of-the-art general-purpose linear program solvers are still quite slow, especially for kECSS and SNDP where the corresponding LPs are *implicit.*

   In the context of SNDP, the state-of-the-art (approximate) LP solvers still require at least quadratic time: Fleischer [18] designs an $\tilde{O}(mnk)$ for solving kECSS LP, and more generally for SNDP and its generalization [18, 14] with at least $\Theta(m \min\{n, k_{\max}\})$ iterations of minimum cost flow's computation are the best known running time where $k_{\max}$ is the maximum connectivity requirements.

2. Most existing techniques that round fractional solutions to integral ones are not "friendly" for the design of fast algorithms. For instance, Jain's celebrated iterative rounding [26] requires solving the LP $\Omega(m)$ times. Moreover, most LP-based network design algorithms are fine-tuned to optimize approximation factors, while designing near-linear time LP rounding algorithms requires limiting ourselves to a relatively small set of tools, about which we currently have very limited understanding.

This paper completely resolves the first challenge for kECSS and manages to identify a fundamental bottleneck of the second challenge.

---

[4] In Khuller and Vishkin [31], the kECA problem aims at augmenting the connectivity from k to $(k+1)$ (but for us it is from $(k-1)$ to k.)

Challenges for LP Solvers.   Our main challenge is handling the so-called *box constraints* in the LPs. To be concrete, below is the LP relaxation of kECSS on graph $G = (V, E)$.

$$\min\{\sum_{e \in E} c_e x_e : \sum_{e \in \delta_G(S)} x_e \geq k \; (\forall S \subseteq V), x \in [0, 1]^E\} \tag{1}$$

where $\delta_G(S)$ is the set of edges between nodes in S and $V \setminus S$. The box constraints refer to the constraints $x \in [0, 1]^E$. Without these constraints, we can select the same edge multiple times in the solution; this problem is called kECSSM in [13] (see Footnote 1). Removing the box constraints often make the problem significantly easier. For example, the min-cost st-flow problem without the box constraints become computing the shortest st-path, which admits a much faster algorithm.

For kECSS, it can be shown that solving (1) without the box constraints can be reduced to solving (1) *with $k = 1$* and multiplying all x_e with k. In other words, without the box constraints, fractional kECSS is equivalent to fractional 1ECSS. This *fractional* 1ECSS can be $(1 + \varepsilon)$-approximated in near-linear time by plugging in the dynamic minimum cut data structure of Chekuri and Quanrud [6] to the multiplicative weight update framework (MWU).

However, with the presence of box constraints, to use the MWU framework we would need a dynamic data structure for a much more complicated cut problem, that we call, the *minimum normalized free cut* problem (roughly, this is a certain normalization of the minimum cut problem where the costs of up to k heaviest edges in the cut are ignored.) For our problem, the best algorithm in the static setting we are aware of (prior to this work) is to use Zenklusen's $\tilde{O}(mn^4)$-time algorithm [47] for the *connectivity interdiction* problem.[5] This results in an $\tilde{O}(kmn^4)$-time static algorithm. Speeding up and dynamizing this algorithm seems very challenging. Our main technical contribution is an efficient dynamic data structure (in the MWU framework) for the $(1 + \varepsilon)$-approximate minimum normalized free cut problem. We explain the high-level overview of our techniques in Section 2.

Further Related Works.   The kECSS and its special cases have been studied extensively. For all $k \geq 2$, the kECSS problem is known to be APX-hard [15] even on bounded-degree graphs [9] and when the edge costs are 0 or 1 [39]. Although a factor 2 approximation for kECSS has not been improved for almost 3 decades, various special cases of kECSS admit better approximation ratios (see for instance [25, 16, 1]). For instance, the unit-cost kECSS ($c_e = 1$ for all $e \in E$) behaves very differently, admitting a $(1 + O(1/k))$ approximation algorithm [21, 33]. For the 2ECA problem, one can get a better than 2 approximation when the edge costs are bounded [1, 16]. Otherwise, for general edge costs, the factor of 2 has remained the best known approximation ratio even for the 2ECA problem.

The kECSS problem in special graph classes have also received a lot of attention. In Euclidean setting, a series of papers by Czumaj and Lingas led to a near-linear time approximation schemes for constant k [12, 11]. The problem is solvable in near-linear time when k and treewidth are constant [3, 5]. In planar graphs, 2ECSS, 2ECSSM and 3ECSSM admit a PTAS [10, 4].

Organization.   We provide a high-level overview of our proofs in Section 2. In Section 3, we explain the background on Multiplicative Weight Updates (MWU) for completeness (although this paper is written in a way that one can treat MWU as a black box). In Section 4, we

---

[5] In the connectivity interdiction problem, we are given $G = (V, E)$ and $k \in \mathbb{N}$, our goal is to compute $F \subseteq E$ to delete from G in order to minimize the minimum cut in the resulting graph.

prove our main technical component. In Section 5, we present our LP solver. In Section 6, we show how to round the fractional solution obtained from the LP solver. Due to space limitations, many proofs are deferred to Appendix.

2   Overview of Techniques

In this section, we give a high-level overview of our techniques in connection to the known results. Our work follows the standard Multiplicative Weight Update (MWU) framework together with the Knapsack Covering (KC) inequalities (see Section 3 for more background). Roughly, in this framework, in order to obtain a near-linear time LP solver for kECSS, it suffices to provide a fast dynamic algorithm for a certain optimization problem (often called the oracle problem in the MWU literature):

▶ **Definition 3** (Minimum Normalized Free Cuts). *We are given a graph $G = (V, E)$, weight function $\boldsymbol{w} : E \to \mathbb{R}_{\geq 0}$, integer k, and our goal is to compute a cut $S \subseteq V$ together with edges $F \subseteq \delta_G(S) : |F| \leq k - 1$ that minimizes the following objective[6]:*

$$\min_{S \subsetneq V, F \subseteq \delta_G(S):|F| \leq (k-1)} \frac{\boldsymbol{w}(\delta_G(S) \setminus F)}{k - |F|},$$

where $\delta_G(S)$ denotes the set of edges that has exactly one end point in S. We call the minimizer (S, F) the minimum normalized free cut.

This is similar to the minimum cut problem, except that we are allowed to "remove" up to $(k - 1)$ edges (called *free edges*) from each candidate cut $S \subseteq V$, and the cost would be "normalized" by a factor of $(k - |F|)$.[7] Notice that there are (apparently) two sources of complexity for this problem. First, we need to find the cut S and second, given S, to compute the optimal set $F \subseteq \delta_G(S)$ of free edges. To our best knowledge, a previously fastest algorithm for this problem takes $\tilde{O}(mn^4)$ time by reducing to the connectivity interdiction problem [47], while we require near-linear time. This is our first technical challenge.

Our second challenge is as follows. To actually speed up the whole MWU framework, in addition to solving the oracle problem statically efficiently, we further need to implement a dynamic version of the oracle with polylog(n) update time. In our case, the goal is to maintain a dynamic data structure on graph $G = (V, E)$, weight function $\mathbf{w}$, cost function c, that supports the following operation:

▶ **Definition 4.** *The PUNISHMIN operation computes a $(1 + O(\varepsilon))$-approximate normalized free cut and multiply the weight of each edge $e \in \delta_G(S) \setminus F$ by a factor of at most e^ε.*[8]

We remark that invoking the PUNISHMIN operation does not return the cut (S, F), and the only change is the weight function $\mathbf{w}$ being maintained by the data structure.

▶ **Proposition 5** (Informal). *Assume that we are given a dynamic algorithm that supports PUNISHMIN with amortized* polylog(n) *cost per operations, then the kECSS LP can be solved in time $\tilde{O}(m)$.*

---

[6] For any function f, for any subset S of its domain, we define $f(S) = \sum_{s \in S} f(s)$.

[7] This is in fact a special case of a similar objective considered by Feldmann, Könemann, Pashkovich and Sanità [14], who considered applying the MWU framework for the generalized SNDP

[8] The actual weight $w(e)$ is updated for all $e \in \delta_G(S) \setminus F$: $w(e) \leftarrow w(e) \cdot \exp(\frac{\varepsilon c_{\min}}{c_e})$ where $c_{\min}$ is the minimum edge capacity in $\delta_G(S) \setminus F$.

Let us call such a dynamic algorithm a fast **dynamic punisher**. The fact that a fast dynamic punisher implies a fast LP solver is an almost direct consequence of MWU [23].

Therefore, we focus on designing a fast dynamic algorithm for solving (and punishing) the minimum normalized free cut problem. Our key idea is an efficient and dynamic implementation of the **weight truncation** idea.

> **Weight truncation**: Let $G = (V, E)$ and $\rho \in \mathbb{R}_{\geq 0}$ be a threshold. For any weight function $\mathbf{w}$ of G, denote by $\mathbf{w}_\rho$ the truncated weight defined by $\mathbf{w}_\rho(e) = \min\{\mathbf{w}(e), \rho\}$ for each $e \in E$. Call an edge e with $\mathbf{w}(e) \geq \rho$ a ρ-heavy edge.

Our main contribution is to show that, when allowing $(1 + \varepsilon)$-approximation, we can use the weight truncation to reduce the minimum normalized free cut to minimum cut with $O(\mathrm{polylog}(n))$ extra factors in the running time. Moreover, this reduction can be implemented efficiently in the dynamic setting. We present the ideas in two steps, addressing our two technical challenges mentioned above respectively. First, we show how to solve the static version of minimum normalized free cut in near-linear time. Second, we sketch the key ideas to implement them efficiently in the dynamic setting, which can be used in the MWU framework.

We remark that weight truncation technique has been used in different context. For instance, Zenklusen [47] used it for reducing the connectivity interdiction problem to $O(|E|)$ instances of the minimum budgeted cut problem.

2.1   Step 1: Static Algorithm

We show that the minimum normalized free cut problem can be solved efficiently in the static setting. For convenience, we often use the term cut to refer to a set of edges instead of a set of vertices.

Define the objective function of our problem as, for any cut C,

$$\mathsf{val}_{\mathbf{w}}(C) = \min_{F \subseteq C : |F| \leq k-1} \frac{\mathbf{w}(C \setminus F)}{k - |F|}.$$

For any weight function $\mathbf{w}$, denote by $\mathsf{OPT}_{\mathbf{w}} = \min_C \mathsf{val}_{\mathbf{w}}(C)$. In this paper, the graph G is always fixed, while $\mathbf{w}$ is updated dynamically by the algorithm (so we omit the dependence on G from the notation val and OPT). When $\mathbf{w}$ is clear from context, we sometimes omit the subscript $\mathbf{w}$.

We show that the truncation technique can be used to establish a connection between our problem and minimum cut.

▶ **Lemma 6.** *We are given a graph $G = (V, E)$, weight function $\boldsymbol{w}$, integer k, and $\varepsilon > 0$. For any threshold $\rho \in (\mathsf{OPT}_{\boldsymbol{w}}, (1 + \varepsilon)\mathsf{OPT}_{\boldsymbol{w}}]$,*
- *any optimal normalized free cut in $(G, \boldsymbol{w})$ is a $(1 + \varepsilon)$-approximate minimum cut in $(G, \boldsymbol{w}_\rho)$, and*
- *any minimum cut C^* in $(G, \boldsymbol{w}_\rho)$ is a $(1 + \varepsilon)$-approximation for the minimum normalized free cut.*

Proof. First, consider any cut C with $\mathsf{val}(C) = \mathsf{OPT}$. Let $F \subseteq C$ be an optimal set of free edges for C, so we have $\mathbf{w}_\rho(C \setminus F) \leq \mathbf{w}(C \setminus F) = (k - |F|)\mathsf{OPT}$. Moreover, $\mathbf{w}_\rho(F) \leq |F|\rho$. This implies that

$$\mathbf{w}_\rho(C) = \mathbf{w}_\rho(C \setminus F) + \mathbf{w}_\rho(F) < k\rho \tag{2}$$

Next, we prove that any cut in $(G, \mathbf{w}_\rho)$ is of value at least $k\mathsf{OPT}$ (so the cut C is a $(1 + \varepsilon)$ approximate minimum cut). Assume for contradiction that there is a cut C' such that $\mathbf{w}_\rho(C') < k\mathsf{OPT}$. Let $F' \subseteq C'$ be the set of ρ-heavy edges. Observe that $|F'| \leq k - 1$ since otherwise the total weight $\mathbf{w}_\rho(C')$ would have already exceeded $k\mathsf{OPT}$. This implies that $\mathbf{w}(C' \setminus F') = \mathbf{w}_\rho(C' \setminus F') < (k - |F'|)\mathsf{OPT}$ and that

$$\mathsf{val}(C') \leq \frac{\mathbf{w}(C' \setminus F')}{(k - |F'|)} < \mathsf{OPT}$$

which is a contradiction. Altogether, we have proved the first part of the lemma.

To prove the second part of the lemma, consider a minimum cut C^* in $(G, \mathbf{w}_\rho)$, we have that $\mathbf{w}_\rho(C^*) < \mathbf{w}_\rho(C) < k\rho$ (from Equation (2)). Again, the set of heavy edges $F^* \subseteq C^*$ can contain at most $k-1$ edges, so we must have $\mathbf{w}(C^* \setminus F^*) < (k - |F^*|)\rho \leq (k - |F^*|)(1 + \varepsilon)\mathsf{OPT}$, implying that $\mathsf{val}(C^*) < (1 + \varepsilon)\mathsf{OPT}$.   ◄

We remark that this reduction from the minimum normalized free cut problem to the minimum cut problem does not give an exact correspondence, in the sense that a minimum cut in $(G, \mathbf{w}_\rho)$ cannot be turned into a minimum normalized free cut in $(G, \mathbf{w})$. In other words, the approximation factor of $(1 + \varepsilon)$ is unavoidable.

▶ **Theorem 7.** *Given a graph $G = (V, E)$ with weight function $\boldsymbol{w}$ and integer k, the minimum normalized free cut problem can be $(1 + \varepsilon)$ approximated by using $O(\frac{1}{\varepsilon} \cdot \log n)$ calls to the exact minimum cut algorithm.*

Proof. We assume that the minimum normalized free cut of G is upper bounded by some value M which is polynomial in $n = |V(G)|$ (we show how to remove this assumption in the full version of the paper). For each i such that $(1 + \varepsilon)^i \leq M$, we compute the minimum cut C_i in $(G, \mathbf{w}_{\rho_i})$ where $\rho_i = (1 + \varepsilon)^i$ and return one with minimum value $\mathsf{val}(C_i)$. Notice that there must be some i^* such that $\rho_{i^*} \in (\mathsf{OPT}_\mathbf{w}, (1 + \varepsilon)\mathsf{OPT}_\mathbf{w}]$ and by the lemma, we must have that C_{i^*} is a $(1 + \varepsilon)$-approximate solution for the normalized free cut problem.   ◄

By using any near-linear time minimum cut algorithm e.g., [27], the collorary follows.

▶ **Corollary 8.** *There exists a $(1 + \varepsilon)$ approximation algorithm for the minimum normalized free cut problem that runs in time $\tilde{O}(|E|/\varepsilon)$.*

2.2 Step 2: Dynamic Algorithm

The next idea we use is from Chekuri and Quanrud [6]. One of the key concepts there is that it is sufficient to solve a "range punishing" problem in near-linear time; for completeness we prove this sufficiency in Appendix. In particular, the following proposition is a consequence of their work:

▶ **Definition 9.** *A **range punisher**[9] is an algorithm that, on any input graph G, initial weight function $\boldsymbol{w} = \boldsymbol{w}^{\mathrm{init}}$, real numbers ε, and $\lambda \leq \mathsf{OPT}_{\boldsymbol{w}^{\mathrm{init}}}$, iteratively applies $\mathrm{PUNISHMIN}$ on $(G, \boldsymbol{w})$ until the optimal becomes at least $\mathsf{OPT}_{\boldsymbol{w}} \geq (1 + \varepsilon)\lambda$.*

The following proposition connects a fast range punisher to a fast LP solver.

---

[9] Our range punisher corresponds to an algorithm of Chekuri and Quanrud [7] in one epoch.

▶ **Proposition 10.** *If there exists a range punisher running in time*

$$\widetilde{O}\left(|E| + K + \sum_{e \in E} \log(\frac{\boldsymbol{w}(e)}{\boldsymbol{w}^{\mathrm{init}}(e)})\right)$$

where K is the number of cuts punished, then, there exists a fast dynamic punisher, and consequently the kECSS LP can be solved in near-linear time.

This proposition applies generally in the MWU framework independent of problems. That is, for our purpose of solving kECSS LP, we need a fast range punisher for the minimum normalized free cut problem. For Chekuri and Quanrud [6], they need such algorithm for the minimum cut problem (therefore a fast LP solver for the Held-Karp bound).

▶ **Theorem 11** ([6], informal). *There exists a fast range punisher for the minimum cut problem.*

Our key technical tool in this paper is a more robust reduction from the range punishing of normalized free cuts to the one for minimum cuts. This reduction works for all edge weights and is suitable for the dynamic setting. That is, it is a strengthened version of Lemma 6 and is summarized below (see its proof in Section 4).

▶ **Theorem 12** (Range Mapping Theorem). *Let $(G = (V, E), \boldsymbol{w})$ be a weighted graph. Let $\lambda > 0$ and $\rho = (1 + \gamma)\lambda$.*

1. *If the value of optimal normalized free cut is in $[\lambda, (1 + \gamma)\lambda)$, then the value of minimum cut in $(G, \boldsymbol{w}_\rho)$ lies in $[k\rho/(1 + \gamma), k\rho)$.*
2. *For any cut C where $\boldsymbol{w}_\rho(C) < k\rho$, then $\frac{w(C \setminus F)}{k - |F|} < (1 + \gamma)\lambda$ where F contains all ρ-heavy edges in C. In particular, $\mathsf{val}(C) < (1 + \gamma)\lambda$.*

Given the above reduction, we can implement range punisher fast. We present its full proof in Section 5 and sketch the argument below.

▶ **Theorem 13.** *There exists a fast range punisher for the minimum normalized free cut problem.*

Proof (sketch). We are given λ and weighted graph $(G, \mathbf{w}) : \mathbf{w} = \mathbf{w}^{\mathrm{init}}$ such that $\mathsf{OPT}_{\mathbf{w}^{\mathrm{init}}} \geq \lambda$. Our goal is to punish the normalized free cuts until the optimal value in $(G, \mathbf{w})$ becomes at least $(1 + \varepsilon)\lambda$. We first invoke Theorem 7 to get a $(1 + \varepsilon)$-approximate cut, and if the solution is already greater than $(1 + \varepsilon)^2\lambda$, we are immediately done (this means $\mathsf{OPT} > (1 + \varepsilon)\lambda$).

Now, we know that $\mathsf{OPT} \leq (1 + \varepsilon)^2\lambda \leq (1 + 3\varepsilon)\lambda$. We invoke Lemma 12(1) with $\gamma = 3\varepsilon$. The minimum cut in $(G, \mathbf{w}_\rho)$ has size in the range $[k\rho/(1 + 3\varepsilon), k\rho)$. We invoke (one iteration of) Theorem 11 with $\lambda' = k\rho(1 + 3/\varepsilon)$ to obtain a cut C whose size is less than $k\rho$ and therefore, by Lemma 12(1), $\mathsf{val}(C) < (1 + 3\varepsilon)\lambda$. This is a cut that our algorithm can punish (we ignore the detail of how we actually punish it – we would need to do that implicitly since the cut itself may contain up to m edges). We repeat this process until all cuts whose values are relevant have been punished, that is, we continue this process until the returned cut C has size at least $k\rho$.

The running time of this algorithm is

$$\widetilde{O}\left(|E| + K + \sum_{e \in E} \log(\frac{\mathbf{w}_\rho(e)}{\mathbf{w}_\rho^{\mathrm{init}}(e)})\right) \leq \widetilde{O}\left(|E| + K + \sum_{e \in E} \log(\frac{\mathbf{w}(e)}{\mathbf{w}^{\mathrm{init}}(e)})\right)$$

Notice that we rely crucially on the property of our reduction using truncated weights. ◀

We remark that in the actual proof of Theorem 13, there are quite a few technical complications (e.g., how to find optimal free edges for a returned cut C?), and we cannot invoke Theorem 11 in a blackbox manner. We refer to Section 5 for the details.

2.3 LP Rounding for kECSS

Most known techniques for kECSS (e.g. [22, 33]) rely on iterative LP rounding, which is computationally expensive. We achieve fast running time by making use of the 2-approximation algorithm of Khuller and Vishkin [31].

Roughly speaking, this algorithm creates a directed graph H from the original graph G and then compute on H the minimum-weight k disjoint arboresences. The latter can be found by Gabow's algorithms, in either $\tilde{O}(|E||V|k)$ or $\tilde{O}(k|E|\sqrt{|V|}\log c_{\max})$ time.

To use their algorithm, we will construct H based on the support of the fractional solution x computed by the LP solver. By the integrality of the arborescence polytope [41], an integral solution is as good as the fractional solution. However, the support of x can be potentially large, which causes Gabow's algorithm to take longer time. Here our idea is a sparsification of the support, by extending the celebrated sparsification theorem of Benzcur and Karger [2] to handle our problem, i.e., we prove the following (see Section 6 for the proofs):

▶ **Theorem 14.** *Let G be a graph and c_G its capacities. There exists a capacitated graph (H, c_H) on the same set of vertices that can be computed in $\tilde{O}(m)$ such that (i) $|E(H)| = \tilde{O}(nk)$, and (ii) for every cut S and $F \subseteq S : |F| \leq (k-1)$, we have $c_G(S \setminus F) = (1 \pm \varepsilon)\, c_H(S \setminus F)$.*

Benzcur and Karger's theorem corresponds to this theorem when $k = 1$. We believe that this theorem might have further applications, e.g., for providing a fast algorithm for the connectivity interdiction problem. Our result implies the following (see Section 6 for the proof):

▶ **Theorem 15.** *Assume that there exists an algorithm that finds a minimum-weight k-arborescences in an m-edge n-node graph in time $T_k(m, n)$. Then there exists a $(2 + \varepsilon)$ approximation algorithm for kECSS running in time $\tilde{O}(m/\varepsilon^2 + T_k(kn/\varepsilon^2, n))$*

Applying Theorem 15 with the Gabow's algorithm (see Theorem 35 in Section 6), we obtain Corollary 2.

3 Preliminaries

In this section, we review the multiplicative-weight update (MWU) framework for solving a (covering) LP relaxation of the form $\min\{c \cdot x : Ax \geq 1, x \geq 0\}$, where A is an m-by-n matrix with non-negative entries and $c \in \mathbb{R}^n_{\geq 0}$. Our presentation abstracts away the detail of MWU, so readers should feel free to skip this section.

Let $A_1, \ldots, A_m$ be the rows of matrix A. Here is a concrete example:

■ **Held-Karp Bound:** The Held-Karp bound on input (G, c) aims at solving the LP:[10]

$$\min\{ \sum_{e \in E(G)} c_e x_e : \sum_{e \in S} x_e \geq 2 \text{ for any cut } S \subseteq E \}$$

---

[10] We refer the readers to [6] for more discussion about this LP and Held-Karp bound.

Matrix $A = A_G$ is a cut-edge incidence matrix of graph G where each row A_j corresponds to a cut $F_j \subseteq E(G)$, so there are exponentially many rows. Each column corresponds to an edge $e \in E(G)$. There are exactly $|E(G)|$ columns. The matrix is implicitly given as an input graph G.

We explain the MWU framework in terms of matrices. Some readers may find it more illustrative to work with concrete problems in mind.

MWU Framework for Covering LPs

In the MWU framework for solving covering linear programs, we are given as input an m-by-n matrix A and cost vectors c associated with the columns.[11] Let $\varepsilon > 0$ be a parameter; that is, we aim at computing a solution x that is $(1 + \varepsilon)$ approximation of the optimal LP solution. Denote by $\text{MINROW}(A, w)$ the value $\min_{j \in [m]} A_j w$. We start with an initial weight vector $\mathbf{w}_i^{(0)} = 1/c_i$ for $i \in [n]$. On each day $t = 1, \ldots, T$, we compute an approximately "cheapest" row j^* such that $A_{j^*} \mathbf{w}^{(t-1)} \leq (1 + \varepsilon) \text{MINROW}(A, \mathbf{w}^{(t-1)})$, and update the weight $\mathbf{w}_i^{(t)} \leftarrow \mathbf{w}_i^{(t-1)} \exp\left(\frac{\varepsilon A_{j^*,i} c_{\min}}{c_i} \right)$ where $c_{\min} = \min_{i \in [n]} \frac{c_i}{A_{j^*,i}}$.[12] After $T = O(n \log n / \varepsilon^2)$ many days, the solution can be found by taking the best scaled vectors; in particular, observe that, for any day t, the scaled vector $\bar{\mathbf{w}}^{(t)} = \mathbf{w}^{(t)} / \left(\min_{j \in [m]} A_j \mathbf{w}^{(t)} \right)$ is always feasible for the LP. The algorithm returns $\bar{\mathbf{w}}^{(t)}$ which has minimum cost. The following theorem shows that at least one such solution is near-optimal.

▶ **Theorem 16.** *For $T = O(\frac{n \log n}{\varepsilon^2})$, one of the solutions $\bar{\mathbf{w}}^{(t)}$ for $t \in [T]$ is a $(1 + O(\varepsilon))$ approximation of the optimal solution $\min\{c \cdot x : Ax \geq 1, x \geq 0\}$.*

Since we use slightly different language than the existing proofs in the literature, we provide a proof in the appendix.

KC Inequalities

Our LP is hard to work with mainly because of the mixed packing/covering constraints $x \in [0, 1]^n$. There is a relatively standard way to get rid of the mixed packing/covering constraints by adding Knapsack covering (KC) inequalities into the LP. In particular, for each row (or constraint) $j \in [m]$, we introduce constraints:

$$(\forall F \subseteq \text{supp}(A_j), |F| \leq (k-1)) : \sum_{i \in [n] \setminus F} A_{j,i} x_i \geq k - |F|, \text{ or } \sum_{i \in [n] \setminus F} \frac{A_{j,i}}{(k - |F|)} x_i \geq 1$$

Let A^{kc} be the new matrix after adding KC inequalities, that is, imagine the row indices of A^{kc} as (j, F) where $j \in [m]$ and $F \subseteq \text{supp}(A_j)$; we define $A^{\text{kc}}_{(j,F),i} = A_{j,i}/(k - |F|)$. The actual number of rows in A^{kc} can be as high as $m \cdot n^{O(k)}$, but our algorithm will not be working with this matrix explicitly.

The following lemma shows that we can now remove the packing constraints. We defer the proof to Appendix.

▶ **Lemma 17.** *Any solution to $\{x \in \mathbb{R}^n : A^{kc} x \geq 1, x \geq 0\}$ is feasible for $\{x \in \mathbb{R}^n : Ax \geq k, x \in [0, 1]\}$. Conversely, for any point z in the latter polytope, there exists a point z' in the former such that $z' \leq z$.*

---

[11] There are several ways to explain such a framework. Chekuri and Quanrud [6] follow the continuous setting of Young [46]. We instead follow the combinatorial interpretation of Garg and Könemann [23].

[12] In the MWU literature, this is often referred to as an oracle problem.

▶ **Corollary 18.** *For any cost vector $c \in \mathbb{R}_{\geq 0}^n$,*

$$\min\{c^T x : A^{kc} x \geq 1, x \geq 0\} = \min\{c^T x : A x \geq k, x \in [0,1]\}$$

4 Range Mapping Theorem

The goal of this section is to prove Theorem 12, a cornerstone of this paper. We emphasize that it works for *any* weight function $\mathbf{w}$. First, we introduce more notations for convenience. For any cut $C \in \mathcal{C}$, and any subset of edges $F \subseteq E$, we define $\mathsf{val}_{\mathbf{w}}(C, F) = \frac{\mathbf{w}(C \setminus F)}{k - |F|}$ if $F \subseteq C$ and $|F| < k$; otherwise, $\mathsf{val}_{\mathbf{w}}(C, F) = \infty$. Also, denote $\mathsf{val}_{\mathbf{w}}(C) = \min_{F \subseteq E} \mathsf{val}_{\mathbf{w}}(C, F)$. By definition, we have $\mathsf{val}_{\mathbf{w}}(C) = \min_{i \leq k-1} \mathsf{val}_{\mathbf{w}}(C, F_i)$ where F_i is the set of heaviest i edges in C with respect to weight function $\mathbf{w}$. We let $\mathsf{mincut}_{\mathbf{w}_\rho}$ be the value of a minimum cut with respect with weight $\mathbf{w}_\rho$. When it is clear from context, we sometimes omit the subscript $\mathbf{w}$. For any positive number ρ, let $H_{\mathbf{w},\rho} = \{e \in E : \mathbf{w}(e) \geq \rho\}$ be the set of ρ-heavy edges.[13] Define the weight truncation $\mathbf{w}_\rho(e) = \min\{\mathbf{w}(e), \rho\}$.

▶ **Theorem 19** (Restatement of Theorem 12). *We are given a weighted graph $(G, \boldsymbol{w})$, $\lambda > 0$ be a parameter and $\rho = (1 + \gamma)\lambda$. Then we have the following:*
1. *If $\mathsf{OPT}_{\boldsymbol{w}} \in [\lambda, (1+\gamma)\lambda)$, then $\mathsf{mincut}_{\boldsymbol{w}_\rho} \in [k\rho/(1+\gamma), k\rho)$, and*
2. *if a cut C satisfies $\boldsymbol{w}_\rho(C) < k\rho$, then $\mathsf{val}_{\boldsymbol{w}}(C, H_{\boldsymbol{w},\rho} \cap C) < (1+\gamma)\lambda$.*

Notice that the above theorem not only gives a mapping between solutions of the two problems but also that the heavy edges can be used as a set of free edges. We say that a cut C is *interesting* if it contains at most $k-1$ heavy edges, i.e., $|H_{\mathbf{w},\rho} \cap C| < k$.

▶ **Proposition 20.** *If cut $C \subseteq E$ is not interesting (i.e., $|H_{\boldsymbol{w},\rho} \cap C| \geq k$), then $\mathsf{val}_{\boldsymbol{w}}(C) \geq \rho$ and $\boldsymbol{w}_\rho(C) \geq k\rho$.*

Proof. The fact that $\mathbf{w}_\rho(C) \geq k\rho$ follows immediately from the definition of heavy edges. Let F_i be the set heaviest i edges in C with respect to $\mathbf{w}$. Since C contains at least k heavy edges, we have that for all $i < k$, $C \setminus F_i$ contains at least $k - i$ heavy edges. Therefore, we have $\mathsf{val}_{\mathbf{w}}(C) = \min_{i \leq k-1} \frac{\mathbf{w}(C \setminus F_i)}{k-i} \geq \min_{i \leq k-1} \frac{(k-i)\rho}{k-i} = \rho$. ◀

Proposition 20 says that if a cut is not interesting it must be expensive as a normalized free cut (i.e., high $\mathsf{val}_{\mathbf{w}}(C)$) and as a graph cut (i.e., high $w_\rho(C)$). We next give a characterization that relates $\mathsf{val}_{\mathbf{w}}$ and the sizes of the cuts for interesting cuts.

▶ **Lemma 21.** *Let C be an interesting cut. Then $\mathsf{val}_{\boldsymbol{w}}(C) \leq \mathsf{val}_{\boldsymbol{w}}(C, H_{\boldsymbol{w},\rho} \cap C) < \rho$ if and only if $\boldsymbol{w}_\rho(C) < k\rho$.*

Proof. ($\rightarrow$) By definition of $\mathbf{w}_\rho$, we have

$$\mathbf{w}_\rho(C) = \mathbf{w}(C \setminus (H_{\mathbf{w},\rho} \cap C)) + \rho|H_{\mathbf{w},\rho} \cap C|. \tag{3}$$

If $\mathsf{val}_{\mathbf{w}}(C, H_{\mathbf{w},\rho} \cap C) < \rho$, then $\mathbf{w}(C \setminus H_{\mathbf{w},\rho} \cap C) < \rho(k - |H_{\mathbf{w},\rho} \cap C|)$. By Equation (3), we have $\mathbf{w}_\rho(C) < k\rho$.

($\leftarrow$) Denote $F = H_{\mathbf{w},\rho} \cap C$. By definition of val, we have

$$\mathsf{val}_{\mathbf{w}}(C) \leq \mathsf{val}_{\mathbf{w}}(C, F) = \frac{\mathbf{w}(C \setminus F)}{k - |F|} \overset{(3)}{=} \frac{\mathbf{w}_\rho(C) - \rho|F|}{k - |F|} < \frac{k\rho - \rho|F|}{k - |F|} = \rho. \quad ◀$$

---

[13] When it is clear from the context, for brevity, we might say that e is a *heavy edge* instead of ρ-*heavy edge*.

Proof of Theorem 19. For the first part, we begin by proving that $\mathsf{mincut}_{\mathbf{w}_\rho} < k\rho$. Let C^* be a cut such that $\mathsf{val}_{\mathbf{w}}(C^*) = \mathsf{OPT}_{\mathbf{w}}$. By Proposition 20, C^* must be interesting. Since $\mathsf{val}_{\mathbf{w}}(C^*) = \mathsf{OPT}_{\mathbf{w}} < (1+\gamma)\lambda = \rho$, Lemma 21 implies that we have $\mathbf{w}_\rho(C^*) < k\rho$. Therefore, $\mathsf{mincut}_{\mathbf{w}_\rho} < k\rho$.

Next, we prove that $\mathsf{mincut}_{\mathbf{w}_\rho} \geq k\rho/(1+\gamma)$. Let C be a cut, and denote $F = H_{\mathbf{w},\rho} \cap C$. If C is not interesting, then Proposition 20 implies that $\mathbf{w}_\rho(C) \geq k\rho \geq k\rho/(1+\gamma)$. If C is interesting, by definition of $\mathbf{w}_\rho$, we have

$$\mathbf{w}_\rho(C) = \mathbf{w}(C \setminus F) + \rho|F| \geq \mathsf{OPT}_{\mathbf{w}}(k - |F|) + \frac{\rho}{1+\gamma}|F| \geq \frac{\rho k}{1+\gamma}.$$

The last inequality follows since by assumption $\mathsf{OPT}_{\mathbf{w}} \geq \rho/(1+\gamma)$.

For the second part of the theorem, as $\mathbf{w}_\rho(C) < k\rho$, Proposition 20 implies that C is interesting. By Lemma 21, $\mathsf{val}_{\mathbf{w}}(C, H_{\mathbf{w},\rho} \cap C) < \rho = (1+\gamma)\lambda$. ◄

5   Fast Approximate LP Solver

In this section, we construct the fast range punisher for the normalized free cut problem. Our algorithm cannot afford to maintain the actual MWU weights, so it will instead keep track of *lazy weights*. From now on, we will use $\mathbf{w}^{\mathrm{mwu}}$ to denote the actual MWU weights and $\mathbf{w}$ the weights that our data structure maintains.

▶ **Theorem 22** (Fast Range Punisher). *Given graph G initial weight function $\boldsymbol{w}^{\mathrm{init}}$ and two real values $\lambda, \varepsilon > 0$ such that $\lambda \leq \mathsf{OPT}_{\boldsymbol{w}^{\mathrm{init}}}$, there is a randomized algorithm that iteratively applies* PunishMin *until the optimal with respect to the final weight function $\boldsymbol{w}^{\mathrm{mwu}}$ becomes at least $\mathsf{OPT}_{\boldsymbol{w}^{\mathrm{mwu}}} \geq (1+\varepsilon)\lambda$, in time $\widetilde{O}(|E| + K + \frac{1}{\varepsilon}\sum_{e \in E}\log(\cdot\frac{\boldsymbol{w}^{\mathrm{mwu}}(e)}{\boldsymbol{w}^{\mathrm{init}}(e)}))$, where K is the number of cuts punished.*

The following theorem is almost standard: the fast range punisher, together with a fast algorithm for approximating OPT_w for any weight $\mathbf{w}$, implies a fast approximate LP solver (e.g., see [6, 18]). For completeness, we provide the proof in the Appendix.

▶ **Theorem 23** (Fast LP Solver). *Given a fast range punisher as described in Theorem 22, and a near-linear time algorithm for approximating OPT_w for any weight function w, there is an algorithm that output $(1 + O(\varepsilon))$-approximate solution to kECSS LP in $\tilde{O}(m/\varepsilon^2)$ time.*

Notice that the above theorem implies our main result, Theorem 1. The rest of this section is devoted to proving Theorem 22. Following the high-level idea of [6], our data structure has two main components:

- **Range cut-listing data structure**: This data structure maintains dynamic (truncated) weighted graph $(G, \mathbf{w}_\rho)$ and is able to find a (short description of) $(1 + O(\varepsilon))$-approximate cut whenever one exists, that is, it returns a cut of size between λ and $(1 + O(\varepsilon))\lambda$ for some parameter λ. Since our weight function $\mathbf{w}$ changes over time, the data structure also has an interface that allows such changes to be implemented. The data structure can be taken and used directly in a blackbox manner, thanks to [6].

- **Lazy weight data structures**: Notice that a fast range punisher can only afford the running time of $\widetilde{O}\left(\sum_e \log \frac{\mathbf{w}^{\mathrm{mwu}}(e)}{\mathbf{w}^{\mathrm{init}}(e)}\right)$ for updating weights, while in the MWU framework, some edges would have to be updated much more often. We follow the idea of [6] to maintain approximate (lazy) weights that do not get updated too often but are still sufficiently close to the real weights. We remark that $\mathbf{w}^{\mathrm{mwu}}$ only depends on the sequence of cuts, that PunishMin actually punishes. This lazy weight data structure is responsible for maintaining $\mathbf{w}$ that satisfies the following invariant:

▶ **Invariant 24.** *We have* $(1 - \varepsilon)\boldsymbol{w}^{\mathrm{mwu}} \leq \boldsymbol{w} \leq \boldsymbol{w}^{\mathrm{mwu}}$.

That is, we allow **w** to underestimate weights, but they cannot deviate more than by a factor of $(1 - \varepsilon)$. In this way, our data structure only needs to update the weight implicitly and output necessary increments to the cut listing data structure whenever the invariant is violated.

In sum, our range punisher data structures deal with three weight functions **w** (lazy weights), $\mathbf{w}_\rho$ (truncated lazy weights, used by the range cut listing data structure) and $\mathbf{w}^{\mathrm{mwu}}$ (actual MWU weights, maintained implicitly).

The rest of this section is organized as follows. In Section 5.1–Section 5.3, we explain the components that will be used in our data structure, and in Section 5.4, we prove Theorem 22 using these components.

5.1 Compact representation of cuts

This part serves as a "communication language" for various components in our data structure. Since a cut can have up to $\Omega(m)$ edges, the data structure cannot afford to describe it explicitly. We will use a compact representation of cuts [6], which allows us to describe any $(1 + \varepsilon)$-approximate solution in a given weighted graph using $\tilde{O}(1)$ bits; notice that, in the MWU framework, we only care about (punishing) near-optimal solutions, so it is sufficient for us that we are able to concisely describe such cuts.

Formally, we say that a family $\mathcal{F}$ of subsets of edges is ε-canonical for $(G, \mathbf{w})$ if (i) $|\mathcal{F}| \leq \tilde{O}(|E|)$, (ii) any $(1 + \varepsilon)$-approximate minimum cut of $(G, \mathbf{w})$ is a disjoint union of at most $\tilde{O}(1)$ sets in $\mathcal{F}$, (iii) any set $S \in \mathcal{F}$ can be described concisely by $\tilde{O}(1)$ bits, and (iv) every edge in the graph belongs to $\tilde{O}(1)$ sets in $\mathcal{F}$. It follows that any $(1 + \varepsilon)$-approximate cut admits a short description. Denote by $[[S]]$ a short description of cut $S \in \mathcal{F}$, and for each $(1 + \varepsilon)$ approximate cut C, $[[C]]$ a short description of C.

▶ **Lemma 25** (implicit in [6]). *There exists a randomized data structure that, on input $(G, \boldsymbol{w})$, can be initialized in near-linear time, (w.h.p) constructs an ε-canonical family $\mathcal{F} \subseteq 2^{E(G)}$, and handles the following queries:*

- *Given a description $[[C]]$ of a $(1 + \varepsilon)$-approximate cut, output a list of $\tilde{O}(1)$ subsets in $\mathcal{F}$ such that C is a disjoint union of those subsets in $\tilde{O}(1)$ time.*
- *Given a description of $[[S]]$, $S \in \mathcal{F}$, output a list of edges in S in $\tilde{O}(|S|)$ time.*

5.2 Range Cut-listing Data Structure

The cut listing data structure is encapsulated in the following theorem.

▶ **Theorem 26** (Range Cut-listing Data Structure [6]). *The cut-listing data structure, denoted by $\mathcal{D}$, maintains dynamically changing weighted graph $(G, \widehat{\boldsymbol{w}})$ and supports the following operations.*

- $\mathcal{D}.\text{INIT}(G, \boldsymbol{w}^{\mathrm{init}}, \lambda, \varepsilon)$ *where G is a graph, $\widehat{\boldsymbol{w}}$ is an initial weight function, and $\mathsf{mincut}_{\widehat{\boldsymbol{w}}} \geq \lambda$: initialize the data structure and the weight $\widehat{\boldsymbol{w}} \leftarrow \boldsymbol{w}^{\mathrm{init}}$ in $\tilde{O}(m)$ time.*
- $\mathcal{D}.\text{FINDCUT}()$: *output either a short description of a $(1 + O(\varepsilon))$-approximate mincut $[[C]]$ or $\emptyset$ (when $\mathsf{mincut}_{\widehat{\boldsymbol{w}}} > (1 + \varepsilon)\lambda$). The operation takes amortized $\tilde{O}(1)$ time.*
- $\mathcal{D}.\text{INCREMENT}(\Delta)$ *where $\Delta = \{(e, \delta_e)\}$ is the set of increments (defined by a pair of an edge $e \in E$ and a value $\delta_e \in \mathbb{R}_{\geq 0}$): For each $(e, \delta_e) \in \Delta$, $\widehat{\boldsymbol{w}}(e) \leftarrow \widehat{\boldsymbol{w}}(e) + \delta_e$. The operation takes $\tilde{O}(|\Delta|)$ time (note that $|\Delta|$ corresponds to the number of increments).*

As outlined earlier, the cut listing data structure will be invoked with $\widehat{\mathbf{w}} = \mathbf{w}_\rho$.

5.3   Truncated Lazy MWU Increment

The data structure is formally summarized by the definition below.

▶ **Definition 27** (Truncated Lazy MWU Increment). *A truncated lazy MWU increment denoted by $\mathcal{L}$ maintains the approximate weight function $\boldsymbol{w}$ explicitly, and exact weight $\boldsymbol{w}^{\mathrm{mwu}}$ implicitly and supports the following operations:*[14]

- $\mathcal{L}.\text{INIT}(G, \boldsymbol{w}^{\mathrm{init}}, \rho)$ *where G is a graph, $\boldsymbol{w}^{\mathrm{init}}$ is the initial weight function, $\rho \in \mathbb{R}_{>0}$: Intialize the data structure, and set $\boldsymbol{w} \leftarrow \boldsymbol{w}^{\mathrm{init}}$.*
- $\mathcal{L}.\text{PUNISH}([[C]])$ *where C is a cut: Internally punish the free cut (C, F) for some F (to be made precise later) and output a list of increment $\Delta = \{(e, \delta_e)\}$ so that for each $e \in E$, $\boldsymbol{w}^{\mathrm{init}}(e)$ plus the total increment over e is $\boldsymbol{w}_\rho(e)$.*
- $\mathcal{L}.\text{FLUSH}()$*: Return the exact weight $\boldsymbol{w}^{\mathrm{mwu}}$.*

Remark that the output list of increments returned by PUNISH is mainly for the purpose of syncing with the cut listing data structure (so it aims at maintaining $\mathbf{w}_\rho$ instead of $\mathbf{w}$). Also, in the PUNISH operation, the data structure must compute the set $F \subseteq C$ of free edges efficiently (these are the edges whose weights would not be increased). This is one of the reasons for which we cannot use the lazy update data structure in [6] as a blackbox.

▶ **Theorem 28.** *There exists a lazy MWU increment with the following time complexity: (i) init operation takes $\tilde{O}(m)$ time, (ii) PUNISH takes $\tilde{O}(K) + \widetilde{O}\left(\sum_e \log \frac{\boldsymbol{w}^{\mathrm{mwu}}(e)}{\boldsymbol{w}^{\mathrm{init}}(e)}\right)$ time in total where K is the number of calls to PUNISH and outputs at most $\widetilde{O}\left(\sum_e \log \frac{\boldsymbol{w}^{\mathrm{mwu}}(e)}{\boldsymbol{w}^{\mathrm{init}}(e)}\right)$ increments, and (iii) flush takes $\tilde{O}(m)$ time. Moreover, the Invariant 24 is maintained throughout the execution.*

For space reason, we prove Theorem 28 in the full version of the paper.

5.4   A Fast Range Punisher for Normalized Free Cut Problem

Now we have all necessary ingredients to prove Theorem 22. The algorithm is very simple and described in Algorithm 1. We initialize the cut-listing data structure $\mathcal{D}$ so that it maintains the truncated weight $\mathbf{w}_\rho$ and the lazy weight data structure $\mathcal{L}$. We iteratively use $\mathcal{D}$ to find a cheap cut in $(G, \mathbf{w}_\rho)$ until no such cut exists. Due to our mapping theorem, such a cut found can be used for our problem, and the data structure $\mathcal{L}$ is responsible for punishing the weights (Line 8) and returns the list of edges to be updated (this is for the cut-listing $\mathcal{D}$ to maintain its weight function $\mathbf{w}_\rho$).

---

[14] This is implicit in the sense that w is divided into parts and they are internally stored in different memory segments. Whenever needed, the real weight can be constructed from the memory content in near-linear time.

Algorithm

🟧 **Algorithm 1** FASTRANGEPUNISHER$(G, \mathbf{w}, \lambda)$.

Input: $G, \mathbf{w}^{\mathrm{init}}, \lambda, \varepsilon$ such that $\mathsf{OPT}_{\mathbf{w}^{\mathrm{init}}} \geq \lambda$.
Output: a correct weight function $\mathbf{w} = \mathbf{w}^{\mathrm{mwu}}$ such that $\mathsf{OPT}_{\mathbf{w}} \geq (1 + \varepsilon)\lambda$.

1 $\mathbf{w} \leftarrow \mathbf{w}^{\mathrm{init}}$ and $\rho \leftarrow (1 + \varepsilon)\lambda$
2 Let $\mathbf{w}_\rho$ be the truncated weight function of $\mathbf{w}$.
3 if $\mathsf{mincut}_{w_\rho} \geq k\rho$ **then return** w.
4 Let $\mathcal{D}$ and $\mathcal{L}$ be cut listing data structure, and truncated lazy MWU increment.
5 $\mathcal{D}.\mathrm{INIT}(G, \mathbf{w}_\rho, k\rho/(1+\varepsilon), \varepsilon)$
6 $\mathcal{L}.\mathrm{INIT}(G, \mathbf{w}, \rho, \varepsilon)$
7 while $\mathcal{D}.FINDCUT()$ returns $[[C]]$ **do**
8 $\quad\quad \Delta \leftarrow \mathcal{L}.\mathrm{PUNISH}([[C]])$
9 $\quad\quad \mathcal{D}.\mathrm{INCREMENT}(\Delta)$
10 $\mathbf{w} \leftarrow \mathcal{L}.\mathrm{FLUSH}()$
11 return w.

Analysis

By input assumption, we have $\mathsf{OPT}_w \geq \lambda$. If $\mathbf{w}$ is returned at line 3, then $\mathsf{mincut}_{\mathbf{w}_\rho} \geq k\rho$. By Theorem 19(1), $\mathsf{OPT}_{\mathbf{w}^{\mathrm{mwu}}} \geq \mathsf{OPT}_{\mathbf{w}} \geq \rho = (1 + \varepsilon)\lambda$, and we are done (since minimum cut can be computed in near-linear time). Now, we assume that $\mathbf{w}$ is returned at the last line. The following three claims imply Theorem 22.

▷ **Claim 29.** For every cut $[[C]]$ returned by the range cut listing data structure during the execution of Algorithm 1, we have that $(C, H_{\mathbf{w},\rho} \cap C)$ is a $(1 + O(\varepsilon))$-approximation to $\mathsf{OPT}_{\mathbf{w}^{\mathrm{mwu}}}$ at the time $[[C]]$ is returned.

We remark that it is important that our cut punished must be approximately optimal w.r.t. the actual MWU weight.

Proof. By definition of $\mathcal{L}.\mathrm{FLUSH}()$ operation, we always have that the exact weight function and approximate weight function are identical at the beginning of the loop. By definition of $\mathcal{L}.\mathrm{PUNISH}([[C]])$, the total increment plus the initial weight at the beginning of the loop for every edge e is $\mathbf{w}_\rho(e)$ and Invariant 24 holds. Therefore, by definition of $\mathcal{D}.\mathrm{INCREMENT}(\Delta)$, the range cut-listing data structure maintains the weight function $\mathbf{w}_\rho$ internally. We now bound the approximation of each cut $[[C]]$ that $\mathcal{D}.\mathrm{FINDCUT}()$ returned. Let $F = H_{\mathbf{w},\rho} \cap C$. By definition of $\mathrm{FINDCUT}()$, we have that $\mathbf{w}_\rho(C) < k\rho$. By Theorem 19(2), $\mathsf{val}_{\mathbf{w}}(C, F) < (1+\varepsilon)\lambda$. By Invariant 24, we have that $\mathsf{val}_{\mathbf{w}^{\mathrm{mwu}}}(C, \tilde{F}) < (1 + O(\varepsilon))\lambda$. Since $\mathsf{OPT}_{\mathbf{w}^{\mathrm{mwu}}} \geq \mathsf{OPT}_{w^{\mathrm{init}}} \geq \lambda$, we have (C, F) is a $(1 + O(\varepsilon))$-approximation to OPT_w. ◁

▷ **Claim 30.** At the end of Algorithm 1, we have $\mathsf{OPT}_{\mathbf{w}^{\mathrm{mwu}}} \geq (1 + \varepsilon)\lambda$.

Proof. Consider the time when $\mathcal{D}.\mathrm{FINDCUT}()$ outputs $\emptyset$. The fact that this procedure terminates means that $\mathsf{mincut}_{\mathbf{w}_\rho} \geq k\rho$. Therefore, Theorem 19(1) implies that $\mathsf{OPT}_{\mathbf{w}} \geq (1 + \varepsilon)\lambda$. Let (C^*, F^*) be an optimal normalized free cut with respect to $\mathbf{w}^{\mathrm{mwu}}$. We have

$$
\begin{aligned}
\mathsf{OPT}_{\mathbf{w}^{\mathrm{mwu}}} &= \mathsf{val}_{\mathbf{w}^{\mathrm{mwu}}}(C^*, F^*) \\
&\geq \mathsf{val}_{\mathbf{w}}(C^*, F^*) \\
&\geq \mathsf{OPT}_{\mathbf{w}} \\
&\geq (1 + \varepsilon)\lambda
\end{aligned}
$$

where the first inequality follows from Invariant 24. ◁

▷ **Claim 31.** Algorithm 1 terminates in $\tilde{O}(m + K + \frac{1}{\varepsilon} \cdot \sum_{e \in E} \log(\frac{\mathbf{w}(e)}{\mathbf{w}^{\text{init}}(e)}))$ time where K is the number of PUNISH operations.

Proof. We first bound the running time due to truncated lazy MWU increment. By Theorem 28, the total running time due to $\mathcal{L}$ (i.e., $\mathcal{L}.\text{INIT}, \mathcal{L}.\text{PUNISH}, \mathcal{L}.\text{FLUSH}$) is $\tilde{O}(m + K + \frac{1}{\varepsilon} \cdot \sum_{e \in E} \log(\frac{\mathbf{w}(e)}{\mathbf{w}^{\text{init}}(e)}))$ time where K is the number of PUNISH operations. We bound the running time due to cut-listing data structure. Observe that the number of cuts listed equals the number of calls of PUNISH operations, and the total number of edge increments in $\mathcal{D}$ is $\widetilde{O}\left(\frac{1}{\varepsilon} \cdot \sum_{e \in E} \log(\frac{\mathbf{w}(e)}{\mathbf{w}^{\text{init}}(e)})\right)$. By Theorem 26, the total running time due to $\mathcal{D}$ (i.e, $\mathcal{D}.\text{INIT}, \mathcal{D}.\text{FINDCUT}(), \mathcal{D}.\text{INCREMENT}(\Delta)$) is as desired. ◁

6 LP Rounding for kECSS (Proof of Theorem 15)

In this section, we show how to round the LP solution x found by invoking Theorem 1. The main idea is use a sampling technique to sparsify the support of x. On the subgraph $G' \subseteq G$ based on this sparsified support, we apply the 2-approximation algorithm of Khuller and Vishkin [32] to obtain a $(2 + \varepsilon)$-approximation solution.

Let G be a graph with capacities c (we omit capacities whenever it is clear from the context). Our algorithm performs the following steps.

Step 1: Sparsification

We will be dealing with the following LP relaxation for kECSS.

$$\min\{\sum_{e \in E(G)} c(e)x_e : \sum_{e \in C \setminus S} x_e \geq k - |S|, \quad \forall C \in \mathcal{C} \; \forall S \in \{F : |F| \leq k - 1 \wedge F \subseteq C\}, x \geq 0\}$$

Denote by $\mathsf{LP}_{kECSS}(G)$ the optimal LP value on input G. We prove the following lemma in the full version of the paper that will allow us to sparsify our graph without changing the optimal fractional value by too much:

▶ **Lemma 32.** *Given an instance (G, c), and in $\tilde{O}(m/\varepsilon^2)$ time, we can compute a subgraph G' having at most $\widetilde{O}(nk/\varepsilon^2)$ edges such that $\mathsf{LP}_{kECSS}(G') = (1 \pm O(\varepsilon))\mathsf{LP}_{kECSS}(G)$.*

The first step is simply to apply this lemma to obtain G' from G.

Step 2: Reduction to k-arborescences

Next, we reduce the kECSS problem to the minimum-cost k-arborescence problem which, on capacitated directed graph (H, c_H), can be described as the following IP:

$$\min\{\sum_{e \in E(H)} c_H(e)z_e : \sum_{e \in \delta^+(C)} z_e \geq k \text{ for } C \in \mathcal{C}; z \in \{0, 1\}^{E(H)}\}$$

where $\mathcal{C}$ is the set of all cuts C such that $\{r\} \subseteq C \subsetneq V(G)$. Denote by $\mathsf{OPT}_{ar}(H)$ and $\mathsf{LP}_{ar}(H)$ the optimal integral and fractional values[15] of the minimum-cost k-arborescence problem respectively. We use the following integrality of its polytope:

▶ **Theorem 33** ([41], Corollary 53.6a). *The minimum-cost k-arborescence's polytope is integral, so we have that $\mathsf{OPT}_{ar}(H) = \mathsf{LP}_{ar}(H)$ for every capacitated input graph H.*

---

[15] The relaxation is simply $\min\{\sum_{e \in E(H)} c_H(e)z_e : \sum_{e \in \delta^+(C)} z_e \geq k \text{ for } C \in \mathcal{C}; z \in [0, 1]^{E(H)}\}$.

For any undirected graph G, denote by $D[G]$ the directed graph obtained by creating, for each (undirected) edge uv in G, two edges $(u \to v)$ and $(v \to u)$ in $D[G]$ whose capacities are just $c(uv)$. We will use the following theorem by Khuller and Vishkin (slightly modified) that relates the optimal values of the two optimization problems.

▶ **Theorem 34.** *For any graph (H, c), the following properties hold:*

- $\mathsf{LP}_{ar}(D[H]) \leq 2\mathsf{LP}_{kECSS}(H)$, *and*
- *Any feasible solution for k-arborescences in $D[H]$ induces a feasible kECSS solution in H of at most the same cost.*

Note that Theorem 33 and the algorithm by Khuller and Vishkin imply that the integrality gap of the kECSS LP is at most 2. While this result is immediate, it seems to be a folklore. To the best of our knowledge, it was not explicitly stated anywhere in the literature. This integrality gap allows us to obtain the first part of Corollary 2. We defer the proof of Theorem 34 to the full version of the paper. Our final tool to obtain Theorem 15 (and the second part of Corollary 2) is Gabow's algorithm:

▶ **Theorem 35** ([20]). *Given a graph $G = (V, E, c)$ with positive cost function c, a fixed root $r \in V$, and let $c_{\max}$ be the maximum cost on edges, there exists an algorithm that in $\tilde{O}(km\sqrt{n}\log(nc_{\max}))$ time outputs the integral minimum-cost k-arborescence.*

Algorithm of Theorem 15

Now, using the graph G' created in the first step, we create $D[G']$, and invoke Gabow's algorithm to compute an optimal k-arborescence in $D[G']$. Let $S \subseteq E(G)$ be the induced kECSS solution.

The cost of S is at most:

$$
\begin{aligned}
\mathsf{OPT}_{ar}(D[G']) &\leq \mathsf{LP}_{ar}(D[G']) \\
&\leq 2\mathsf{LP}_{kECSS}(G') \\
&\leq 2(1 + O(\varepsilon))\mathsf{LP}_{kECSS}(G) \\
&\leq 2(1 + O(\varepsilon))\mathsf{OPT}_{kECSS}(G)
\end{aligned}
$$

The first inequality is due to Theorem 33. The second one is due to Theorem 34 (first bullet). The third one is due to Lemma 32.

Analysis

Step 1 takes $\tilde{O}(m/\varepsilon^2)$ time, by Lemma 32. As the sparsified G' has $m' = \tilde{O}(\frac{nk}{\varepsilon^2})$ edges, for Step 2, by Theorem 35, we can compute the arborescence in $O(km'\sqrt{n}\log(nc_{\max})) = \tilde{O}(\frac{k^2 n^{1.5}}{\varepsilon^2}\log c_{\max})$ time. We show in the full version how to remove the term $\log c_{\max}$ in our case. In summary, the total running time is $\tilde{O}\left(\frac{m}{\varepsilon^2} + \frac{k^2 n^{1.5}}{\varepsilon^2}\right)$. Notice that the running time can be $\tilde{O}(\frac{m}{\varepsilon^2} + T_k(kn/\varepsilon^2, n))$ if we let the running time of Theorem 35 be $T_k(m, n)$, this complete the proof for Theorem 15.

References

1 David Adjiashvili. Beating approximation factor two for weighted tree augmentation with bounded costs. *ACM Transactions on Algorithms (TALG)*, 15(2):1–26, 2018.
2 András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015.

3   André Berger and Michelangelo Grigni. Minimum weight 2-edge-connected spanning subgraphs in planar graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 90–101. Springer, 2007.

4   Glencora Borradaile, Erik D Demaine, and Siamak Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68(2):287–311, 2014.

5   Parinya Chalermsook, Syamantak Das, Guy Even, Bundit Laekhanukit, and Daniel Vaz. Survivable network design for group connectivity in low-treewidth graphs. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2018.

6   Chandra Chekuri and Kent Quanrud. Approximating the held-karp bound for metric TSP in nearly-linear time. *CoRR*, abs/1702.04307, 2017. `arXiv:1702.04307`.

7   Chandra Chekuri and Kent Quanrud. Fast approximations for metric-tsp via linear programming. *CoRR*, abs/1802.01242, 2018. `arXiv:1802.01242`.

8   Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. *CoRR*, abs/1910.08025, 2019. `arXiv:1910.08025`.

9   Béla Csaba, Marek Karpinski, and Piotr Krysta. Approximability of dense and sparse instances of minimum 2-connectivity, tsp and path problems. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 74–83. Society for Industrial and Applied Mathematics, 2002.

10   Artur Czumaj, Michelangelo Grigni, Papa Sissokho, and Hairong Zhao. Approximation schemes for minimum 2-edge-connected and biconnected subgraphs in planar graphs. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 496–505. Society for Industrial and Applied Mathematics, 2004.

11   Artur Czumaj and Andrzej Lingas. On approximability of the minimum-cost k-connected spanning subgraph problem. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 281–290. Citeseer, 1999.

12   Artur Czumaj and Andrzej Lingas. Fast approximation schemes for euclidean multi-connectivity problems. In *International Colloquium on Automata, Languages, and Programming*, pages 856–868. Springer, 2000.

13   Artur Czumaj and Andrzej Lingas. Approximation schemes for minimum-cost k-connectivity problems in geometric graphs. In *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007.

14   Andreas Emil Feldmann, Jochen Könemann, Kanstantsin Pashkovich, and Laura Sanità. Fast approximation algorithms for the generalized survivable network design problem. In *ISAAC*, volume 64 of *LIPIcs*, pages 33:1–33:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

15   Cristina G Fernandes. A better approximation ratio for the minimum sizek-edge-connected spanning subgraph problem. *Journal of Algorithms*, 28(1):105–124, 1998.

16   Samuel Fiorini, Martin Groß, Jochen Könemann, and Laura Sanità. Approximating weighted tree augmentation via chvátal-gomory cuts. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 817–831. SIAM, 2018.

17   Lisa Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000.

18   Lisa Fleischer. A fast approximation scheme for fractional covering problems with variable upper bounds. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1001–1010, 2004.

19   Greg N. Frederickson and Joseph JáJá. Approximation algorithms for several graph augmentation problems. *SIAM J. Comput.*, 10(2):270–283, 1981.

20   Harold N Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, 1995.

21 Harold N Gabow, Michel X Goemans, Éva Tardos, and David P Williamson. Approximating the smallest k-edge connected spanning subgraph by lp-rounding. *Networks: An International Journal*, 53(4):345–357, 2009.

22 Harold N. Gabow, Michel X. Goemans, and David P. Williamson. An efficient approximation algorithm for the survivable network design problem. *Math. Program.*, 82:13–40, 1998. announced at IPCO'93.

23 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, 2007.

24 Michel X. Goemans, Andrew V. Goldberg, Serge A. Plotkin, David B. Shmoys, Éva Tardos, and David P. Williamson. Improved approximation algorithms for network design problems. In *SODA*, pages 223–232. ACM/SIAM, 1994.

25 Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. Improved approximation for tree augmentation: saving by rewiring. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 632–645, 2018.

26 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001. `doi:10.1007/s004930170004`.

27 David R Karger. Minimum cuts in near-linear time. *Journal of the ACM (JACM)*, 47(1):46–76, 2000. announced at STOC'96.

28 David R Karger, Philip N Klein, and Robert E Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM (JACM)*, 42(2):321–328, 1995.

29 Rohit Khandekar, Subhash Khot, Lorenzo Orecchia, and Nisheeth K Vishnoi. On a cut-matching game for the sparsest cut problem. *Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2007-177*, 2007.

30 Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. Graph partitioning using single commodity flows. *J. ACM*, 56(4):19:1–19:15, 2009. `doi:10.1145/1538902.1538903`.

31 Samir Khuller and Ramakrishna Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14(2):214–225, 1993.

32 Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *J. ACM*, 41(2):214–235, 1994. announced at STOC'92.

33 Bundit Laekhanukit, Shayan Oveis Gharan, and Mohit Singh. A rounding by sampling approach to the minimum size k-arc connected subgraph problem. In *International Colloquium on Automata, Languages, and Programming*, pages 606–616. Springer, 2012.

34 Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. In *STOC*, pages 317–329. ACM, 2021.

35 Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *FOCS*, pages 245–254. IEEE Computer Society, 2010.

36 Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 121–130, 2010. `doi:10.1145/1806689.1806708`.

37 Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, las vegas, and o (n1/2-ε)-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1122–1129, 2017.

38 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *FOCS*, pages 950–961. IEEE Computer Society, 2017.

39 David Pritchard. k-edge-connectivity: Approximation and LP relaxation. In *WAOA*, volume 6534 of *Lecture Notes in Computer Science*, pages 225–236. Springer, 2010.

40 Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *SODA*, pages 2616–2635. SIAM, 2019.

41   Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

42   Jonah Sherman. Breaking the multicommodity flow barrier for o(vlog n)-approximations to sparsest cut. In *FOCS*, pages 363–372. IEEE Computer Society, 2009.

43   Jonah Sherman. Nearly maximum flows in nearly linear time. In *FOCS*, pages 263–269. IEEE Computer Society, 2013.

44   Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *FOCS*, pages 919–930. IEEE, 2020.

45   Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1130–1143, 2017. `doi:10.1145/3055399.3055415`.

46   Neal E. Young. Nearly linear-time approximation schemes for mixed packing/covering and facility-location linear programs. *CoRR*, abs/1407.3015, 2014. `arXiv:1407.3015`.

47   Rico Zenklusen. Connectivity interdiction. *Oper. Res. Lett.*, 42(6-7):450–454, 2014.