THEORY OF COMPUTING, Volume 18 (7), 2022, pp. 1–24
www.theoryofcomputing.org

SPECIAL ISSUE: APPROX-RANDOM 2019

Fast and Deterministic Approximations for k-Cut

Kent Quanrud*

Received May 14, 2020; *Revised* March 27, 2022; *Published* April 19, 2022

Abstract. In an undirected graph, a k-cut is a set of edges whose removal breaks the graph into at least k connected components. The minimum-weight k-cut can be computed in $n^{O(k)}$ time, but when k is treated as part of the input, computing the minimum-weight k-cut is NP-hard [Goldschmidt and Hochbaum 1994]. For $\mathrm{poly}(m,n,k)$-time algorithms, the best possible approximation factor is essentially 2 under the Small-Set Expansion Hypothesis [Manurangsi 2017]. Saran and Vazirani [1995] showed that a $(2-2/k)$-approximately minimum-weight k-cut can be computed via $O(k)$ minimum cuts, which implies a $\tilde{O}(km)$ randomized running time via the nearly linear-time randomized min-cut algorithm of Karger [2000]. Nagamochi and Kamidoi [2007] showed that a $(2-2/k)$-approximately minimum-weight k-cut can be computed deterministically in $O(mn+n^2\log n)$ time. These results prompt two basic questions. The first concerns the role of randomization. Is there a deterministic algorithm for 2-approximate minimum k-cut, matching the randomized running time of $\tilde{O}(km)$? The second question qualitatively compares minimum cut to 2-approximate minimum k-cut. Can 2-approximate minimum k-cut be computed as fast as the minimum cut, in $\tilde{O}(m)$ randomized time?

An extended abstract of this paper, [43], appeared in the Proceedings of the 22nd International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'19).

*Work on this paper was supported in part by NSF grant CCF-1526799.

ACM Classification: F.2.2,G.1.6

AMS Classification: 68W25

Key words and phrases: k-cut, multiplicative weight updates

 DOI: 10.4086/toc.2022.v018a007

We give a deterministic approximation algorithm that computes $(2+\varepsilon)$-approximate minimum k-cut in $O(m\log^3 n/\varepsilon^2)$ time, via a $(1+\varepsilon)$-approximation for an LP relaxation of k-cut.

1 Introduction

Let $G=(V,E)$ be an undirected graph with m edges and n vertices, with positive edge capacities given by $c: E \to \mathbb{R}_{>0}$. A *cut* is a set of edges $C \subseteq E$ whose removal leaves G disconnected. For $k \in \mathbb{N}$, a *k-cut* is a set of edges $C \subseteq E$ whose removal leaves G disconnected into at least k components. The capacity of a cut C is the sum capacity $\overline{c}(C) = \sum_{e\in C} c_e$ of edges in the cut. The *minimum k-cut* problem is to find a k-cut C of minimum capacity $\overline{c}(C)$.

The special case $k=2$, which is to find the minimum cut, is particularly well-studied. The minimum cut can be computed in polynomial time by fixing a source s and computing the minimum s-t cut (via s-t max-flow) for all choices of t. Nagamochi and Ibaraki [38, 39] and Hao and Orlin [22] improved the running time to $\tilde{O}(mn)$ which, at the time, was as fast as computing a single maximum flow.[1] A randomized edge contraction algorithm by Karger and Stein [28] finds the minimum cut with high probability in $\tilde{O}\big(n^2\big)$ time; this algorithm is now a staple of graduate level courses on randomized algorithms. Karger [27] gave a randomized algorithm based on the Tutte–Nash-Williams theorem [42, 50] that computes the minimum weight cut with high probability in $\tilde{O}(m)$ time. The best deterministic running time for minimum cut is currently $O\big(mn+n^2\log n\big)$, by Stoer and Wagner [48]. Computing the minimum capacity cut deterministically in nearly linear time is a major open problem. Recently, Kawarabayashi and Thorup [30] made substantial progress on this problem with a deterministic nearly linear-time algorithm for computing the minimum *cardinality* cut in an unweighted simple graph. This algorithm was simplified by Lo, Schmidt and Thorup [34], and a faster algorithm was obtained by Henzinger, Rao, and Wang [24]. For capacitated graphs, a $(2+\varepsilon)$-approximate minimum cut can be computed in $O\big(\big(m\log n + n\log^2 n\big)/\varepsilon\big)$ deterministic time by an algorithm of Matula [37] (as observed by Karger [26]).

The general case $k>2$ is more peculiar. Goldschmidt and Hochbaum [18] showed that for any fixed k, finding the minimum k-cut is polynomial time polynomial-time solvable, but when k is part of the input, the problem is NP-hard. The aforementioned randomized contraction algorithm of Karger and Stein [28] computes a minimum k-cut with high probability in $\tilde{O}\big(n^{2(k-1)}\big)$ time. Thorup [49] gave a deterministic algorithm that also leverages the Tutte–Nash-Williams theorem [42, 50] and runs in $\tilde{O}\big(mn^{2k-2}\big)$ time; this approach was recently refined to improve the running time to $\tilde{O}\big(mn^{2k-3}\big)$ deterministic time [11] and $O\big(n^{(1.981+o(1))k}\big)$ randomized time [21] (where the $o(1)$ goes to zero as k increases). There are slightly faster algorithms for particularly small values of k [33] and when the graph is unweighted [19]. As far as algorithms with running times that are polynomial in k are concerned, Saran and Vazirani [46] showed that a $\big(2-\frac{2}{k}\big)$-approximate minimum k-cut can be obtained by $O(k)$ minimum cut computations. [2]

By the aforementioned min-cut algorithms, this approach can be implemented in $\tilde{O}(km)$ randomized time, $\tilde{O}(kmn)$ time deterministically, and $\tilde{O}(km)$ time deterministically in unweighted graphs. Alternatively, Saran and Vazirani [46] showed that the same approximation factor can be obtained by computing a

[1] $\tilde{O}(\cdots)$ hides polylogarithmic factors.

[2] The term "factor" always refers to a multiplicative factor. An "α-approximation" refers to an approximation by a factor α.

Gomory–Hu tree and taking the k lightest cuts. The Gomory–Hu tree can be computed in n maximum flow computations, and the maximum flow can be computed deterministically in $\tilde{O}\big(m\min\{m^{1/2}, n^{2/3}\}\log U\big)$ time for integer capacities between 1 and U [17]. This gives a $\tilde{O}\big(mn\min\{m^{1/2}, n^{2/3}\}\log U\big)$ deterministic time $(2-2/k)$-approximation for minimum k-cut, which is faster than $\tilde{O}(kmn)$ for sufficiently large k. (There are faster randomized algorithms for maximum flow [32, 35], but these still lead to slower randomized running times than $\tilde{O}(km)$ for k-cut.) An LP-based 2-approximation was derived by Naor and Rabani [41], and a combinatorial 2-approximation was given by Ravi and Sinha [45] (see also [2]), but the running times are worse than those implied by Saran and Vazirani [46]. The best deterministic algorithm in the poly(m,n,k) regime is due to Nagamochi and Kamidoi [40], who compute $\big(2-\frac{2}{k}\big)$-approximately minimum k-cut in $O\big(mn+n^2\log n\big)$ deterministic time. The constant factor of 2 is believed to be essentially the best possible. Manurangsi [36] showed that under the Small-Set Expansion Hypothesis [44], for any fixed $\varepsilon > 0$, one cannot compute a $(2-\varepsilon)$-approximation for the minimum k-cut in poly(k,m,n) time unless $P = NP$.

The state of affairs for computing 2-approximate minimum k-cut in poly(m,n,k)-time parallels the status of minimum cut. The fastest randomized algorithm is an order of magnitude faster than the fastest deterministic algorithm, while in the unweighted case the running times are essentially equal. A basic question is whether there exists a deterministic algorithm that computes a 2-approximation in $\tilde{O}(km)$ time, matching the randomized running time. As Saran and Vazirani's algorithm reduces k-cut to k minimum cuts, the gap between the deterministic and randomized running times for k-cut is not only similar to, but a reflection of, the gap between the deterministic and randomized running times for minimum cut. An $\tilde{O}(m)$ deterministic algorithm for minimum cut would close the gap for 2-approximate minimum k-cut as well.

A second question asks if computing a 2-approximate minimum k-cut is qualitatively harder than computing the minimum cut. There is currently a large gap between the fastest algorithm for minimum cut and the fastest algorithm for 2-approximate minimum k-cut. Can one compute 2-approximate minimum k-cuts as fast as minimum cuts, in $\tilde{O}(m)$ randomized time? Removing the linear dependence on k would show that computing 2-approximate minimum k-cuts is as easy as computing a minimum cut.

1.1 The main result

We make progress on both of these questions with a deterministic and nearly linear-time $(2+\varepsilon)$-approximation scheme for minimum k-cuts. To state the result formally, we first introduce an LP relaxation for the minimum k-cut due to Naor and Rabani [41].

$$\begin{aligned} \min \sum_e c_e x_e \text{ over } x : E \to \mathbb{R} \\ \text{s.t.} \sum_{e\in T} x_e \geq k-1 \text{ for all spanning trees } T, \\ 0 \leq x_e \leq 1 \text{ for all edges } e. \end{aligned} \tag{L}$$

The feasible integral solutions of the LP (L) are precisely the k-cuts in G. Our main contribution is a nearly linear-time approximation scheme for (L).

Theorem 1.1. *In $O\big(m\log^3(n)/\varepsilon^2\big)$ deterministic time, one can compute a $(1+\varepsilon)$-approximation to (L).*

The integrality gap of (L) is known to be $(2-2/n)$ [5, 41]. Upon inspection, the rounding algorithm can be implemented in $O(m\log n)$ time, giving the following nearly linear-time $(2+\varepsilon)$-approximation scheme for minimum k-cut.

Theorem 1.2. *For sufficiently small $\varepsilon > 0$, there is a deterministic algorithm that computes a k-cut with total capacity at most $(2+\varepsilon)$ times the optimum value to (L) in $O\big(m\log^3(n)/\varepsilon^2\big)$ time.*

The algorithm should be compared with the aforementioned algorithms of Saran and Vazirani [46], which computes a $\big(2-\frac{2}{k}\big)$-approximation to the minimum k-cut in $\tilde{O}(km)$ randomized time (with high probability), and of Nagamochi and Kamidoi [40], which computes a $\big(2-\frac{2}{k}\big)$-approximate minimum k-cut in $O\big(mn+n^2\log n\big)$ deterministic time. At the cost of a $(1+\varepsilon)$ factor, we obtain a deterministic algorithm with nearly linear running time for *all values of k*. The approximation factor converges to 2, and we cannot expect to beat 2 under the Small-Set Expansion Hypothesis [36]. Thus, Theorem 1.2 gives a tight, deterministic, and nearly linear-time approximation scheme for minimum k-cut. Theorem 1.2 leaves a little bit of room for improvement: the hope is for a deterministic algorithm that computes $(2-o(1))$-approximate minimum k-cuts in $\tilde{O}(m)$ time. Based on Theorem 1.2, we conjecture that such an algorithm exists.

1.2 Overview of the algorithm

We give a brief sketch of the algorithm, for the sake of informing subsequent discussion on related work in Section 1.3. A more complete description of the algorithm begins in Section 2.

The algorithm consists of a nearly linear-time approximation scheme for the LP (L), and a nearly linear-time rounding scheme. The approximation scheme for solving the LP extends techniques from recent work [7], applied to the dual of an *indirect reformulation* of (L). The rounding scheme is a simplification of the rounding scheme by Chekuri et al. [5] for the more general Steiner k-cut problem, which builds on the primal-dual framework of Goemans and Williamson [15].

The first step is to obtain a $(1+\varepsilon)$-approximation to the LP (L). Here a $(1+\varepsilon)$-approximation to the LP (L) is a feasible vector x of cost at most a $(1+\varepsilon)$ factor greater than the optimum value.

The LP (L) can be solved exactly by the ellipsoid method, with the separation oracle supplied by a minimum spanning tree (MST) computation, but the running time is a larger polynomial than desired. From the perspective of fast approximations, the LP (L) is difficult to handle because it is an exponentially large mixed packing and covering problem, with exponentially many covering constraints alongside upper bounds on each edge. Fast approximation algorithms for mixed packing and covering problems (e.g. [9, 53]) give bicriteria approximations that meet either the covering constraints or the packing constraints but not both. Even without consideration of the objective function, it is not known how to find feasible points to general mixed packing and covering problems in time faster than via exact LP solvers. Alternatively, one may consider the dual of (L), as follows. Let $\mathcal{T}$ denote the family of spanning trees in G.

$$\begin{aligned} &\max\ (k-1)\sum_{T} y_T - \sum_{e\in E} z_e \text{ over } y:\mathcal{T}\to\mathbb{R} \text{ and } z:E\to\mathbb{R} \\ &\text{s.t. } \sum_{T\ni e} y_T \le c_e + z_e \text{ for all edges } e, \end{aligned}$$

$$y_T \geq 0 \text{ for all spanning trees } T \in \mathcal{T},$$
$$z_e \geq 0 \text{ for all edges } e.$$

This program is not a positive linear program, and the edge potentials $z \in \mathbb{R}^E_{\geq 0}$ are difficult to handle by techniques such as [53, 7, 9]. Prior to this work it was not known how to obtain any approximation to (L) (better than its integrality gap) with running time faster than the ellipsoid algorithm.

Critically, we consider the following larger LP instead of (L). Let $\mathcal{F}$ denote the family of all forests in G.

$$\begin{aligned} \min \sum_e c_e x_e &\text{ over } x : E \to \mathbb{R} \\ \text{s.t. } \sum_{e \in F} x_e &\geq |F| + k - n \text{ for all forests } F \in \mathcal{F}, \\ x_e &\geq 0 \text{ for all edges } e \in E. \end{aligned} \tag{C}$$

(C) is also an LP relaxation for k-cut. In fact, (C) is equivalent to (L), as one can verify directly (see Lemma 2.1 below). (C) is obtained from (L) by adding all the knapsack covering constraints [3], which makes the packing constraints ($x_e \leq 1$ for each edge e) redundant.

Although the LP (C) adds exponentially many constraints to the original LP (L), (C) has the advantage of being a pure covering problem. Its dual is a pure packing problem, as follows.

$$\begin{aligned} \text{maximize } \sum_{F \in \mathcal{F}} (|F| + k - n) y_F &\text{ over } y : \mathcal{F} \to \mathbb{R} \\ \text{s.t. } \sum_{F \ni e} y_F &\leq c_e \text{ for all edges } e \in E, \\ y_F &\geq 0 \text{ for all forests } F \in \mathcal{F}. \end{aligned} \tag{P}$$

The above LP packs forests into the capacitated graph G where the value of a forest F depends on the number of edges it contains, $|F|$. The objective value $|F| + k - n$ of a forest F is a lower bound on the number of edges F contributes to any k-cut. Clearly, we need only consider forests with at least $n - k + 1$ edges.

To approximate the optimum solution of the LP (C), we apply the multiplicative weight update (MWU) framework to the above LP (P), which generates $(1 \pm \varepsilon)$-approximations to both (P) and its dual, (C). Implementing the MWU framework in nearly linear time is not immediate, despite precedent for similar problems. In the special case where $k = 2$, the above LP (P) fractionally packs spanning trees into G. A nearly linear-time approximation scheme for $k = 2$ is given in previous work [7]. The general case with $k > 2$ is more difficult for two reasons. First, the family of forests that we pack is larger than the family of spanning trees. Second, (P) is a weighted packing problem, where the coefficients in the objective depends on the number of edges in the forest. When $k = 2$, we need only consider spanning trees with $n - 1$ edges, so all the coefficients are 1 and the packing problem is unweighted. The heterogeneous coefficients in the objective create technical complications in the MWU framework, as the Lagrangian relaxation generated by the framework is no longer solved by a MST. In Section 2, we give an overview of the MWU framework and discuss the algorithmic complications in greater depth. In Section 3 and Section 4, we show how to extend the techniques of [7] with some new observations to overcome these challenges and approximately solve the LP (P) in nearly the same time as one can approximately pack spanning trees. Ultimately, we obtain the following deterministic algorithm for approximately solving the LP (P).

Theorem 1.3. *In $O\left(m\log^3(n)/\varepsilon^2\right)$ deterministic time, one can compute $(1\pm\varepsilon)$-approximations to (P), (C) and (L).*

The second step, after computing a fractional solution x to (L) with Theorem 1.3, is to round x to a discrete k-cut. The rounding step is essentially that of Chekuri et al. [5] for Steiner k-cuts. Their case is more general than ours; we simplify their rounding scheme, and pay greater attention to the running time. The rounding scheme is based on the elegant primal-dual MST algorithm of Goemans and Williamson [15].

Theorem 1.4. *Given a feasible solution x to (C), one can compute a k-cut C with cost at most $2(1-1/n)$ times the cost of x in $O(m\log n)$ time.*

Applying Theorem 1.4 to the output of Theorem 1.3 gives Theorem 1.2.

Computing the minimum k-cut via the LP (L) has additional (informal) benefits. First, computing a minimum k-cut with an approximation factor relative to the LP may be stronger than the same approximation factor relative to the original problem, as the LP can have optimum value lower than the capacity of the minimum (discrete) k-cut. Second, the solution to the LP gives a certificate of approximation ratio, as we can compare the rounded k-cut to the LP solution to infer an upper bound on the approximation ratio that may be smaller than 2.

Lastly, we note that some data structures can be simplified at the cost of randomization by using a randomized MWU framework instead [9]. These modifications are discussed at the end of Section 4.

1.3 Further related results and discussion

Fixed parameter tractability. The k-cut results reviewed above were focused on either exact polynomial-time algorithms for constant k or $(2-o(1))$-approximations in $\mathrm{poly}(k,m,n)$ time, with a particular emphasis on the fastest algorithms in the $\mathrm{poly}(k,m,n)$ time regime. There is also a body of literature concerning fixed parameter tractable algorithms for k-cut. Downey et al. [13] showed that k-cut is $W[1]$-hard in k even for simple unweighted graphs; $W[1]$-hardness implies that it is unlikely to obtain a running time of the form $f(k)\,\mathrm{poly}(m,n)$ for any function f. On the other hand, Kawarabayashi and Thorup [29] showed that k-cut is fixed parameter tractable in the number of edges in the cut. More precisely, [29] gave a deterministic algorithm that, for a given cardinality $s\in\mathbb{N}$, time, either finds a k-cut with at most s edges or reports that no such cut exists in $O\left(s^{s^{O(s)}}n^2\right)$ time. The running time was improved to $O\left(2^{O\left(s^2\log s\right)}n^4\log n\right)$ deterministic time and $\tilde{O}\left(2^{O(s)\log k}n^2\right)$ randomized time by Chitnis et al. [12].

Besides exact parameterized algorithms for k-cut, there is interest in approximation ratios between 1 and 2. Xiao et al. [51] showed that by adjusting the reduction of Saran and Vazirani [46] to use (exact) minimum ℓ-cuts, instead of minimum (2-)cuts, for any choice of $\ell\in\{2,\dots,k-1\}$, one can obtain $\left(2-\frac{\ell}{k}+O\left(\frac{\ell^2}{k^2}\right)\right)$-approximate minimum k-cuts in $n^{O(\ell)}$ time. For sufficiently small $\varepsilon>0$, by setting $\ell\approx\varepsilon k$, this gives a $n^{O(\varepsilon k)}$ time algorithm for $(2-\varepsilon)$-approximate minimum cuts.

The hardness results of Downey et al. [13] and Manurangsi [36] do not rule out approximation algorithms with approximation ratio better than 2 and running times of the form $f(k)\,\mathrm{poly}(m,n)$ (for any function f). Recently, Gupta et al. [20] gave a FPT algorithm that, for a particular constant $c \in (0,1)$, computes a $(2-c)$-approximate minimum k-cut in $2^{O\left(k^6\right)}\tilde{O}\left(n^4\right)$ time. This improves the running time of [51] for $\varepsilon = c$ and k greater than some constant. Further improvements by Gupta et al. [19] achieved a deterministic 1.81 approximation in $2^{O\left(k^2\right)}n^{O(1)}$ time, and a randomized $(1+\varepsilon)$-approximation (for any $\varepsilon > 0$) in $(k/\varepsilon)^{O(k)}n^{k+O(1)}$ time.

Knapsack covering constraints. We were surprised to discover that adding the knapsack covering constraints allowed for *faster* approximation algorithms. Knapsack covering constraints, proposed by Carr et al. [3] in the context of capacitated network design problems, generate a stronger LP whose solutions can be rounded to obtain better approximation factors [3, 4, 31]. However, the larger LP can be much more complicated and is usually more difficult to solve. Recent work obtained a faster approximation scheme for approximately solving covering integer programs via knapsack covering constraints, but the dependency on ε is much worse, and the algorithm has a "weakly nearly linear" running time that suffers from a logarithmic dependency on the multiplicative range of input coefficients [10].

Fast approximations via LPs. Linear programs have long been used to obtain more accurate approximations to NP-hard problems. Recently, we have explored the use of fast LP solvers to obtain *faster* approximations, including situations where polynomial time algorithms are known. In recent work [8], we used a linear-time approximation to an LP relaxation for Metric TSP (obtained in [6]) to effectively sparsify the input and accelerate Christofides's algorithm. While nearly linear-time approximations for complicated LPs are surprising in and of themselves, perhaps the application to obtain faster approximations for combinatorial optimization problems is more compelling. We think this result is an interesting data point for this approach.

2 Reviewing the MWU framework and identifying bottlenecks

In this section, let $\varepsilon > 0$ be fixed. It suffices to assume that $\varepsilon \geq 1/\mathrm{poly}(n)$, since below this point one can use the ellipsoid algorithm instead and still meet the desired running time. For ease of exposition, we seek only a $(1+O(\varepsilon))$-approximation; a $(1+\varepsilon)$-approximation with the same asymptotic running time follows by decreasing ε by a constant factor.

2.1 k-cuts as a (pure) covering problem

As discussed above, the first (and most decisive) step towards a fast, fractional approximation to k-cut is identifying the right LP. The standard LP (L) is difficult because it is a mixed packing and covering problem, and fast approximation algorithms for mixed packing and covering problems lead to bicriteria approximations that we do not know how to round. On the other hand, extending (L) with all the knapsack cover constraints makes the packing constraints $x_e \leq 1$ redundant (as shown below), leaving the pure covering problem, (C). The LPs (L) and (C) have essentially equivalent solutions in the following sense.

Lemma 2.1. *Any feasible solution $x \in \mathbb{R}^E_{\geq 0}$ to (L) is a feasible solution to (C). For any feasible solution x to (C), the truncation $x' \in \mathbb{R}^n_{\geq 0}$ defined by $x'_e = \min\{x_e, 1\}$ is a feasible solution to both (L) and (C).*

Proof. Let x be a feasible solution to (L). We claim that x is feasible in (C). Indeed, let F be a forest, and extend F to a tree T. Then

$$\sum_{e\in F} x_e = \sum_{e\in T} x_e - \sum_{e\in T\setminus F} x_e \geq k-1-|T\setminus F| = k-1-(n-1-|F|) = |F|+k-n,$$

as desired.

Conversely, let $x \in \mathbb{R}^E_{\geq 0}$ be a feasible solution to (C), and let x' be the coordinatewise minimum of x and $\mathbb{1}$. Since $x' \leq \mathbb{1}$, and the covering constraints in (L) are a subset of the covering constraints in (C), if x' is feasible in (C) then it is also feasible in (L). To show that x' is feasible in (C), let F be a forest. Let $F' = \{e \in F : x_e > 1\}$ be the edges in F truncated by x' and let $F'' = F \setminus F'$ be the remaining edges. Since (a) $x'_e = 1$ for all $e \in F'$ and $x'_e = x_e$ for all $e \in F''$, and (b) x covers F'' in (C), we have

$$\sum_{e\in F} x'_e = \sum_{e\in F'} x'_e + \sum_{e\in F''} x'_e \overset{\text{(a)}}{=} \left|F'\right| + \sum_{e\in F''} x_e \overset{\text{(b)}}{\geq} \left|F'\right| + \left|F''\right| + k - n = |F| + k - n,$$

as desired. ∎

While having many more constraints than (L), (C) is a pure covering problem, for which finding a feasible point (faster than an exact LP solver) is at least plausible. The dual of (C) is the LP (P), which packs forests in the graph and weights each forest by the number of edges minus $(n-k)$. The coefficient of a forest in the objective of (P) can be interpreted as the number of edges that forest must contribute to any k-cut.

2.2 A brief sketch of width-independent MWU

We apply a *width-independent* version of the MWU framework to the packing LP (P), developed by Garg and Könemann [14] for multicommodity flow problems and generalized by Young [52]. We restrict ourselves to a sketch of the framework and refer to previous work for further details. We also refer to [1] for additional connections and variations of the MWU framework.

The width-independent MWU framework is a monotonic and iterative algorithm that starts with an empty solution $y = \mathbb{0}$ to the LP (P) and increases y along forests that solve certain Lagrangian relaxations to (P). Each Lagrangian relaxation is designed to steer y away from packing forests that have edges that are already tightly packed. For each edge e, the framework maintains a weight w_e that (approximately) exponentiates the load of edge e with respect to the current forest packing y, as follows:

$$\ln(c_e w_e) \approx \frac{\log n}{\varepsilon} \cdot \frac{\sum_{F\ni e} y_F}{c_e}. \tag{2.1}$$

The weight can be interpreted as follows. For an edge e, the value $\frac{\sum_{F\ni e} y_F}{c_e}$ is the amount of capacity used by the current packing y relative to the capacity of the edge e. We call $\frac{\sum_{F\ni e} y_F}{c_e}$ the (relative) *load* on edge e and is at most 1 if y is a feasible packing. The weight w_e is exponential in the load on the edge, where the exponential is amplified by the leading coefficient $\frac{\log n}{\varepsilon}$. Initially, the empty solution $y = \emptyset$ induces zero load on any edge and we have $w_e = \frac{1}{c_e}$ for each edge e.

On each iteration, the framework solves the following Lagrangian relaxation of (P):

$$\text{maximize } \sum_{F\in\mathcal{F}} (|F|+k-n)z_F \text{ over } z : \mathcal{F} \to \mathbb{R}_{\geq 0} \text{ s.t. } \sum_{e\in E} w_e \sum_{F\ni e} z_F \leq \sum_{e\in E} w_e c_e. \tag{R}$$

Given a $(1+O(\varepsilon))$-approximate solution z to the above, the framework adds δz to y for a carefully chosen value $\delta > 0$ (discussed in greater detail below). The next iteration encounters a different relaxation, where the edge weights w_e are increased to account for the loads increased by adding δz. Note that the edge weights w_e are monotonically increasing over the course of the algorithm.

At the end of the algorithm, standard analysis shows that the fractional forest packing y has objective value $(1-O(\varepsilon))\,\mathrm{OPT}$, and that $(1-O(\varepsilon))y$ satisfies all of the packing constraints. The error can be made one-sided by scaling y up or down. Moreover, it can be shown that at some point in the algorithm, an easily computable rescaling of w is a $(1+O(\varepsilon))$-approximation for the optimum solution to the LP (C) (see for example [14, 6]). Thus, although we may appear more interested in solving the dual LP (P), we are approximately solving the LP (C) as well.

The choice of δ differentiates this "width-independent" MWU from other MWU-type algorithms in the literature. The step size δ is chosen small enough that no weight increases by more than an $\exp(\varepsilon)$ factor, and large enough that some weight increases by (about) an $\exp(\varepsilon)$ factor. The analysis of the MWU framework reveals that $\langle w, c\rangle \leq n^{O(1/\varepsilon)}$ at all times. In particular, each weight can increase by an $\exp(\varepsilon)$ factor at most $O\big(\frac{\ln n}{\varepsilon^2}\big)$ times, so there are at most $O\big(\frac{n \ln n}{\varepsilon^2}\big)$ iterations total.

2.3 Two bottlenecks

The MWU framework alternates between (a) solving the relaxation (R) induced by edge weights w_e and (b) updating the weights w_e for each edge in response to the solution to the relaxation. As the framework requires $O\left(\frac{m \log n}{\varepsilon^2}\right)$ iterations, both parts must be implemented in polylogarithmic amortized time to reach the desired running time. A sublinear per-iteration running time seems unlikely by the following simple observations.

Consider first the complexity of simply expressing a solution. Any solution z to (R) is indexed by forests in G. A forest can have $\Omega(n)$ edges and requires $\Omega(n \log n)$ bits to specify. Writing down the index of just one forest in each of $O\left(\frac{m \log n}{\varepsilon^2}\right)$ iterations takes $O\left(\frac{mn \log^2 n}{\varepsilon^2}\right)$ time. The difficulty of even writing down a solution to (P) is not just a feature of the MWU framework. In general, there exists an optimal solution to (P) that is supported by at most m forests, as m is the rank of the implicit packing matrix. Writing down m forests also requires $\Omega(mn \log n)$ bits. Thus, either on a per-iteration basis in the MWU framework or with respect to to the entire LP, the complexity of the output suggests a quadratic lower bound on the running time.

A second type of bottleneck arises from updating the weights. The weights w_e for each edge reflect the load induced by the packing y, per the formula (2.1). After computing a solution z to the relaxation (R), and updating $y \leftarrow y + \delta z$, we need to update the weights w_e to reflect the increased load from δz. In the worst case, δz packs into every edge, requiring us to update $O(m)$ individual weights. At the very least, δz should pack into the edges of at least one forest, and thus effect $\Omega(n)$ edges. Updating n edge weights in each iteration requires $O\left(\frac{mn\log n}{\varepsilon^2}\right)$ time.

Even in hindsight, implementing either part—solving the relaxation or updating the weights—in isolation in sublinear time remains difficult. Our algorithm carefully plays both parts off each other as co-routines, and amortizes against invariants revealed by the analysis of the MWU framework. The seemingly necessary dependence between parts is an important theme here and an ongoing theme from previous work [6, 7, 10].

3 Greedily finding forests to pack in $O\left(\log^2 n\right)$ amortized time

The MWU framework reduces (P) to a sequence of problems of the form (R). An important aspect of the Lagrangian approach is that satisfying the single packing constraint in (R) is much simpler than simultaneously satisfying all of the packing constraints in (P). With only 1 packing constraint, it suffices to (approximately) identify the best bang-for-buck forest F and taking as much as can fit in the packing constraint. The "bang-for-buck" ratio of a forest F is the ratio

$$\frac{|F|+k-n}{\sum_{e\in F} w_e}, \tag{3.1}$$

where $|F|$ is the number of edges in F. Given a forest F (approximately) maximizing the above ratio, we set $z=\gamma e_F$ for γ as large as possible as fits in the single packing constraint. Note that, when $k=2$, the optimal forest is the minimum weight spanning tree with respect to w.

We first consider the simpler problem of maximizing the above ratio over forests F with exactly $|F|=\ell$ edges, for some $\ell > n-k$. Recall that the MST can be computed greedily by repeatedly adding the minimum weight edge that does not induce a cycle. Optimality of the greedy algorithm follows from the fact that spanning trees are the bases of a matroid called the *graphic matroid*. The forests of exactly ℓ edges are also the bases of a matroid; namely, the restriction of the graphic matroid to forests of at most ℓ edges. In particular, the same greedy procedure computes the minimum weight forest of ℓ edges. Repeating the greedy algorithm for each choice of ℓ, one can solve (R) in $O(km\log n)$ time for each of $O\left(\frac{m\log n}{\varepsilon^2}\right)$ iterations.

Stepping back, we want to compute the minimum weight forest with ℓ edges for a range of $k-1$ values of ℓ, and we can run the greedy algorithm for each choice of ℓ. We observe that the greedy algorithm is oblivious to the parameter ℓ, except for deciding when to stop. We can run the greedy algorithm *once* to build the MST, and then simulate the greedy algorithm for any value of ℓ by taking the first ℓ edges added to the MST.

Lemma 3.1. *Let T be the minimum weight spanning tree with respect to w. For any $\ell \in [n-1]$, the minimum weight forest with respect to w with ℓ edges consists of the first ℓ minimum weight edges of T.*

Lemma 3.1 effectively reduces (R) to one MST computation, which takes $O(m\log n)$ time. Repeated over $O\left(\frac{m\log n}{\varepsilon^2}\right)$ iterations, this leads to a $O\left(\frac{m^2\log^2 n}{\varepsilon^2}\right)$ running time. As observed previously [49, 7], the minimum weight spanning tree does not have to be rebuilt from scratch from one iteration to another, but rather adjusted dynamically as the weights change.

Lemma 3.2 (Holm, de Lichtenberg, and Thorup [25])**.** *In $O(\log^2 n)$ amortized time per increment to w, one can maintain the MST with respect to w.*

The running time of Lemma 3.2 depends on the number of times the edge weights change, so we want to limit the number of weight updates exposed to Lemma 3.2. It is easy to see that solving (R) with respect to a second set of weights $\tilde{w}$ that is a coordinatewise $(1 \pm \varepsilon)$-approximation of w gives a solution that is a $(1 \pm \varepsilon)$-approximation to (R) with respect to w. We maintain the MST with respect to an approximation $\tilde{w}$ of w, and only propagate changes from w to $\tilde{w}$ when w is greater than $\tilde{w}$ by at least a $(1+\varepsilon)$ factor. As mentioned in Section 2, a weight w_e increases by a $(1+\varepsilon)$ factor at most $O\left(\frac{\log n}{\varepsilon^2}\right)$ times. Applying Lemma 3.2 to the discretized weights $\tilde{w}$ and amortizing against the total growth of weights in the system gives us the following.

Lemma 3.3. *In $O\left(\frac{m \log^3 n}{\varepsilon^2}\right)$ total time, one can maintain the MST with respect to a set of weights $\tilde{w}$ such that, for all $e \in E$, we have $\tilde{w}_e \in (1 \pm \varepsilon) w_e$. Moreover, the MST makes at most $O\left(\frac{m \log n}{\varepsilon^2}\right)$ edge updates total.*

Given such an MST T as above, and $\ell \in \{n-k+1, \ldots, n-1\}$ we need the ℓ minimum ($\tilde{w}$-)weight edges to form an (approximately) minimum weight forest F of ℓ edges. However, the data structure of Lemma 3.2 does not provide a list of edges in increasing order of weight. We maintain the edges in sorted order separately, where each time the dynamic MST replaces one edge with another, we make the same update in the sorted list. Clearly, such a list can be maintained in $O(\log n)$ time per update by self-adjusting binary search trees. Our setting is simpler because the range of possible values of any weight $\tilde{w}_e$ is known in advance as follows. For $e \in E$, define

$$\mathcal{W}_e = \left\{ \frac{(1+\varepsilon)^i}{c_e} : i \in \left\{0, 1, \ldots, O\left(\frac{\log n}{\varepsilon^2}\right)\right\}\right\}.$$

Then $\tilde{w}_e \in \mathcal{W}_e$ for all $e \in E$ at all times. Define $\mathcal{L} = \{(e, \alpha) : \alpha \in \mathcal{W}_e\}$. The set $\mathcal{L}$ represents the set of all possible assignments of weights to edges that may arise. As mentioned above, $|\mathcal{L}| = O\left(\frac{m \log n}{\varepsilon^2}\right)$. Let B be a balanced binary tree over $\mathcal{L}$, where $\mathcal{L}$ is sorted by increasing order of the second coordinate $\tilde{w}_e$ (and ties are broken arbitrarily). The tree B has height $\log|\mathcal{L}| = O(\log m)$ and can be built in $O(|\mathcal{L}|) = O\left(\frac{m \log n}{\varepsilon^2}\right)$ time[3].

We mark the leaves based on the edges in T. Every time the MST T adds an edge e of weight $\tilde{w}_e$, we mark the corresponding leaf $(e, \tilde{w}_e)$ as marked. When an edge e is deleted, we unmark the corresponding leaf. Lastly, when an edge $e \in T$ has its weight increased from α to α', we unmark the leaf (e, α) and mark the leaf (e, α'). Note that only edges in T have their weight changed.

For each subtree of B, we track aggregate information and maintain data structures over the set of all marked leaves in the subtree. For a node b in B, let $\mathcal{L}_b$ be the set of marked leaves in the subtree rooted at b. For each $b \in B$ we maintain two quantities: (a) the number of leaves marked in the subtree rooted at b, $|\mathcal{L}_b|$; and (b) the sum of edges weights of leaves marked in the subtree rooted at b, $W_b = \sum_{(e,\alpha) \in \mathcal{L}_b} \alpha$.

[3]or even less time if we build B lazily, but constructing B is not a bottleneck.

Since the height of B is $O(\log n)$, both of these quantities can be maintained in $O(\log n)$ time per weight update.

Lemma 3.4. *In $O\big(m\log(n)/\varepsilon^2\big)$ time initially and $O(\log n)$ time per weight update, one can maintain a data structure that, given $\ell \in [n-1]$, returns in $O(\log n)$ time (a) the ℓ minimum weight edges of the MST (implicitly), and (b) the total weight of the first ℓ edges of the MST.*

With Lemma 3.2 and Lemma 3.4, we can now compute, for any $\ell \in [n-1]$, a $(1+\varepsilon)$-approximation to the minimum weight forest of ℓ edges, along with the sum weight of the forest, both in logarithmic time. To find the best forest, then, we need only query the data structure for each of the $k-1$ integer values from $n-k+1$ to $n-1$. That is, excluding the time to maintain the data structures, we can now solve the relaxation (R) in $O(k\log n)$ time per iteration.

At this point, we still require $O(km\log n)$ time just to solve the Lagrangian relaxations (R) generated by the MWU framework. (There are other bottlenecks, such as updating the weights at each iteration, that we have not yet addressed.) To remove the factor of k in solving (R), we require one final observation.

Lemma 3.5. *Let the edges of T be listed in nondecreasing order of $\tilde{w}_e$. Then the subforest of T maximizing the ratio (3.1) with respect to $\tilde{w}$ is attained by a prefix of this list of edges. This prefix can be identified by a binary search probing the ratio of at most $O(\log k)$ such forests.*

Proof. Enumerate the MST edges $e_1,\dots,e_{n-1} \in T$ in increasing order of weight. For ease of notation, we denote $\tilde{w}_i = \tilde{w}_{e_i}$ for $i \in [n-1]$. We define a function $f : [k-1] \to \mathbb{R}_{>0}$ by

$$f(i) = \frac{i}{\sum_{j=1}^{n-k+i} \tilde{w}_j}.$$

For each i, $f(i)$ is the ratio achieved by the first $n-k+i$ edges of the MST. Our goal is to maximize $f(i)$ over $i \in [k-1]$. For any $i \in [k-2]$, we have

$$\begin{aligned} f(i+1) - f(i) &= \frac{i+1}{\sum_{j=1}^{n-k+i+1} \tilde{w}_j} - \frac{i}{\sum_{j=1}^{n-k+i} \tilde{w}_j} \\ &= \frac{(i+1)\sum_{j=1}^{n-k+i} \tilde{w}_j - i\sum_{j=1}^{n-k+i+1} \tilde{w}_j}{\left(\sum_{j=1}^{n-k+i+1} \tilde{w}_j\right)\left(\sum_{j=1}^{n-k+i} \tilde{w}_j\right)} = \frac{\sum_{j=1}^{n-k+i} \tilde{w}_j - i\tilde{w}_{n-k+i+1}}{\left(\sum_{j=1}^{n-k+i+1} \tilde{w}_j\right)\left(\sum_{j=1}^{n-k+i} \tilde{w}_j\right)}. \end{aligned}$$

Since the denominator $\left(\sum_{j=1}^{n-k+i+1} \tilde{w}_j\right)\left(\sum_{j=1}^{n-k+i} \tilde{w}_j\right)$ is positive, we have $f(i+1) \le f(i) \iff i\tilde{w}_{n-k+i+1} \ge \sum_{j=1}^{n-k+i} \tilde{w}_j$. If $i\tilde{w}_{n-k+i+1} < \sum_{j=1}^{n-k+i} \tilde{w}_j$ for all $i \in [k-2]$, then $f(1) < f(2) < \cdots < f(k-1)$, so $f(i)$ is maximized by $i = k-1$. Otherwise, let $i_0 \in [k-2]$ be the first value of i such that $f(i+1) \le f(i)$. For $i_1 \ge i_0$, as (a) $\tilde{w}_{e_i}$ is increasing in i, and (b) $f(i_0+1) \le f(i_0)$, we have

$$\begin{aligned} \sum_{j=1}^{n-k+i_1} \tilde{w}_j - i_1 w_{n-k+i_1+1} &= \sum_{j=1}^{n-k+i_0} \tilde{w}_j - i_0\tilde{w}_{n-k+i_1+1} + \sum_{j=n-k+i_0+1}^{n-k+i_1} \tilde{w}_j - (i_1 - i_0)\tilde{w}_{n-k+i_1+1} \\ &\overset{\text{(a)}}{\le} \sum_{j=1}^{n-k+i_0} \tilde{w}_j - i_0\tilde{w}_{n-k+i_0+1} \overset{\text{(b)}}{\le} 0. \end{aligned}$$

That is, $f(i_1+1) \le f(i_1)$ for all $i_1 \ge i_0$. Thus f consists of one increasing subsequence followed by a decreasing subsequence, and its global maximum is the unique local maximum. ■

By Lemma 3.5, the choice of ℓ can be found by calculating the ratio of $O(\log k)$ candidate forests. By Lemma 3.4, the ratio of a candidate forest can be computed in $O(\log n)$ time.

Lemma 3.6. *Given the data structure of Lemma 3.4, one can compute a $(1+O(\varepsilon))$-approximation to (R) in $O(\log n \log k)$ time.*

The polylogarithmic running time in Lemma 3.6 may seem surprising when considering that solutions to (R) should require at least a linear number of bits, as discussed earlier in Section 2.3. In hindsight, a combination of additional structure provided by the MWU framework and the LP (P) allows us to apply data structures that effectively compress the forests and output each forest in polylogarithmic amortized time. Implicit compression of this sort also appears in previous work [7, 6, 10].

4 Packing greedy forests in $O(\log^2 n)$ amortized time

In Section 3, we showed how to solve (R) in polylogarithmic time per iteration. In this section, we address the second main bottleneck: updating the weights w after increasing y to $y+\delta z$ per the formula (2.1), where z is an approximate solution to the relaxation (R) and $\delta > 0$ is the largest possible value such that no weight increases by more than a $(1+\varepsilon)$ factor. As discussed in Section 2.3, this may be hard to do in polylogarithmic time when many of the edges $e \in E$ require updating.

A sublinear-time weight update must depend heavily on the structure of the solutions generated to (R). In our case, each solution z to a relaxation (R) is of the form γe_F, where e_F is the indicator vector of a forest F and $\gamma > 0$ is a scalar as large as possible subject to the packing constraint in (R). We need to update the weights to reflect the loads induced by $\delta z = \delta\gamma e_F$, where δ is chosen large as possible so that no weight increases by more than an $\exp(\varepsilon)$ factor. With this choice of δ, the weight update simplifies to the following formula. Let w denote the set of weights before the updates and w' denote the set of weights after the updates. For a solution $z = \gamma e_F$, we have

$$w'_e = \begin{cases} w_e & \text{if } e \notin F, \\ \exp\left(\frac{\varepsilon \min_{f\in F} c_f}{c_e}\right) & \text{if } e \in F. \end{cases} \tag{4.1}$$

The weight update formula above can be interpreted as follows. Because our solution is supported along a single forest F, the only edges whose loads are affected are those in the forest F. As load is relative to the capacity of an edge e, the increase of the logarithm of the weight w_e of an edge $e \in F$ is inversely proportional to its capacity. By choice of δ, the minimum capacity edge $\arg\min_{f\in F} c_f$ has its weight increased by an $\exp(\varepsilon)$ factor. The remaining edges with larger capacity each have the logarithm of their weight increased in proportion to the ratio of the bottleneck capacity to its own capacity.

Simplifying the weight update formula does not address the basic problem of updating the weights of every edge in a forest F, *without visiting every edge in F*. Here we require substantially more structure as to how the edges in F are selected. We observe that although there may be $\Omega(n)$ edges in F, we can always decompose F into a logarithmic number of "canonical subforests", as follows.

Lemma 4.1. *One can maintain, in $O(\log n)$ time per update to the MST T, a collection of subforests $\mathcal{C}_T \subseteq \mathcal{F}$ such that:*

(i) $|\mathcal{C}_T| = O(n \log n)$.

(ii) Each edge $e \in T$ is contained in $O(\log n)$ forests.

(iii) For each $\ell \in [n-1]$, the forest F consisting of the ℓ minimum weight edges in T decomposes uniquely into the disjoint union of $O(\log n)$ forests in $\mathcal{C}_T$. The decomposition can be computed in $O(\log n)$ time.

In fact, the collection of subforests is already maintained implicitly in Lemma 3.4. Recall, from Section 3, the balanced binary tree B over the leaf set $\mathcal{L}$, which consists of all possible discretized weight-to-edge assignments and is ordered in increasing order of weight. Leaves are marked according to the edges in the MST T, and each node is identified with the forest consisting of all marked leaves in the subtree rooted at the node. For each $\ell \in [n-1]$, the forest F_ℓ induced by the ℓ minimum weight edges in T is the set of marked leaves over an interval of $\mathcal{L}$. The interval decomposes into the disjoint union of leaves of $O(\log n)$ subtrees, which corresponds to decomposing F_ℓ into the disjoint union of marked leaves of $O(\log n)$ subtrees of B. That is, the forests of marked leaves induced by subtrees of B gives the "canonical forests" $\mathcal{C}_T$ that we seek.

The following technique of decomposing weight updates is critical to previous work [7, 6, 10]; we briefly discuss the high-level ideas and refer to previous work for complete details.

Decomposing the solution into a small number of known static sets is important because weight updates can be simulated over a *fixed set* efficiently. The data structure `lazy-inc`, defined in [7] and inspired by techniques by Young [53], simulates a weight update over a fixed set of weights in such a way that the time can be amortized against the logarithm of the increase in each of the weights. As discussed above, the total logarithmic increase in each of the weights is bounded from above. The data structure `lazy-inc` is dynamic, allowing insertion and deletion into the underlying set, in $O(\log n)$ time per insertion or deletion [6].

We define an instance of `lazy-inc` at each node in the balanced binary tree B. Whenever a leaf is marked as occupied, the corresponding edge is inserted into each of $O(\log n)$ instances of `lazy-inc` at the ancestors of the leaf; when a leaf is marked as unoccupied, it is removed from each of these instances as well. Each instance of `lazy-inc` can then simulate a weight update over the marked leaves at its nodes in $O(1)$ constant time per instance, plus a total $O(\log n)$ amortized time. More precisely, the additional time is amortized against the sum of increases in the logarithms of the weights, which (as discussed earlier) is bounded above by $O\big(m \log(n)/\varepsilon^2\big)$.

We also track, for each canonical forest, the minimum capacity of any edge in the forest. The minimum capacity ultimately controls the rate at which all the other edges increase, per (4.1).

Given a forest F induced by the ℓ minimum weight edges of T, we decompose F into the disjoint union of $O(\log n)$ canonical subforests of T. For each subforest we have precomputed the minimum capacity, and an instance of `lazy-inc` that simulates weight updates on all edges in the subforest. The minimum capacity over edges in F determines the rate of increase, and the increase is made to each instance of `lazy-inc` in $O(1)$ time per instance plus $O(\log n)$ amortized time over all instances.

Lemma 4.2. *Given a forest F generated by Lemma 3.6, one can update the edge weights per (4.1) in $O(\log^2 n)$ amortized time per iteration.*

Note that Lemma 4.2 holds only for the forests output by Lemma 3.6. We can not decompose other forests in G, or even other subforests of T, into the disjoint union $O(\log n)$ subforests. Lemma 4.1 holds specifically for the forests induced by the ℓ minimum weight edges of T, for varying values of ℓ. This limitation highlights the importance of coupling the oracle and the weight update: the running time in Lemma 3.6 for solving (R) is amortized against the growth of the weights, and the weight updates in Lemma 4.2 leverage the specific structure by which solutions to (R) are generated.

We note that the `lazy-inc` data structures can be replaced by random sampling in the randomized MWU framework [9]. Here one still requires the decomposition into canonical subforests; an efficient threshold-based sampling is then conducted at each subforest.

5 Putting things together

In this section, we summarize the main points of the algorithm and account for the running time claimed in Theorem 1.3.

Proof of Theorem 1.3. By standard analysis (e.g., [7, Theorem 2.1]), the MWU framework returns a $(1-O(\varepsilon))$-approximation to the packing LP (P) as long as we can approximately solve the relaxation (R) to within a $(1-O(\varepsilon))$ factor. This slack allows us to maintain the weight w_e to within a $(1 \pm O(\varepsilon))$ factor of the "true weights" given (up to a leading constant) by (2.1). In particular, we only propagate a change to w_e when it has increased by a $(1+\varepsilon)$ factor. Each weight w_e is monotonically increasing and its growth is bounded by a $m^{O(1/\varepsilon)}$ factor, so each weight w_e increases by a $(1+\varepsilon)$ factor $O\left(\frac{\log m}{\varepsilon^2}\right)$ times.

By Lemma 3.6, each instance of (R) can be solved in $O(\log^2 n)$ amortized time. Here the running time is amortized against the number of weight updates, as the solution can be updated dynamically in $O(\log^2 n)$ amortized time. By Lemma 4.2, the weight update with respect to a solution generated by Lemma 3.6 can be implemented in $O(\log^2 n)$ amortized time. Here again the running time is amortized against the growth of the edge weights. Since there are $O\left(\frac{m \log n}{\varepsilon^2}\right)$ total edge updates, this gives a total running time of $O\left(\frac{m \log^3 n}{\varepsilon^2}\right)$. ∎

6 Rounding fractional forest packings to k-cuts

In this section, we show how to round a fractional solution x to (L) to a k-cut of cost at most twice the cost of x. The rounding scheme is due to Chekuri et al. [5] for the more general problem of Steiner k-cuts. The rounding scheme extends the primal-dual framework of Goemans and Williamson [15, 16]. In hindsight, we realized the primal-dual framework is only required for the analysis, and that the algorithm itself is very simple.

We first give a conceptual description of the algorithm, called `greedy-cuts`. The conceptual description suffices for the sake of analyzing the approximation guarantee. Later, we give implementation details and demonstrate that it can be executed in $O(m \log n)$ time.

To describe the algorithm, we first introduce the following definitions.

```
greedy-cuts(G = (V,E), c, x)

// conceptual sketch

1. let E' = { e ∈ E : x_e ≥ n/(2(n−1)) },  E ← E \ E'

2. if E' is a k-cut then return E'

3. let F be a minimum weight spanning forest in E w/r/t x w/ ℓ components

4. return the union of E' and the k − ℓ minimum weight greedy cuts of F
```

Figure 1: A conceptual sketch of a deterministic rounding algorithm for k-cut.

Definition 6.1. Let F be a minimum weight spanning forest in a weighted, undirected graph, and order the edges of F in increasing order of weight (breaking ties arbitrarily). A *greedy component* of F is a connected component induced by a prefix of F. A *greedy cut* is a cut induced by a greedy component of F.

The rounding algorithm is conceptually very simple and a pseudocode sketch is given in Figure 1. We first take all the edges with $x_e > 1/2 + o(1)$, which we denote E'. If this is already a k-cut, then it is a 2-approximation because the corresponding indicator vector is $\leq (2 - o(1))x$. Otherwise, we compute the minimum spanning forest F in the remaining graph, where the weight of an edge is given by x. Letting ℓ be number of components of F, we compute the $k - \ell$ minimum weight greedy cuts with respect to F. We output the union of E' and the $k - \ell$ greedy cuts.

Chekuri et al. [5] implicitly showed that this algorithm has an approximation factor of $2(1 - 1/n)$. Their analysis is for the more general Steiner k-cut problem, where we are given a set of terminal vertices T, and want to find the minimum weight set of edges whose removal divides the graph into at least k components each containing a terminal vertex $t \in T$. The algorithm and analysis is based on the primal-dual framework of Goemans and Williamson [15, 16]. For the minimum weight Steiner tree problem, the primal-dual framework returns a Steiner tree and a feasible fractional cut packing in the dual LP. The cost of the Steiner cut packing is within a $2(1 - o(1))$ factor of the corresponding Steiner tree. Via LP duality, the Steiner tree and the cut packing mutually certify an approximation ratio of $2(1 - o(1))$. The cut packing certificate has other nice properties, and Chekuri et al. [5] show that the $k - 1$ minimum cuts in the support of the fractional cut packing give a 2-approximate minimum Steiner k-cut.

For the (non-Steiner) k-cut problem, we want minimum cuts in the support of the fractional cut packing returned by the primal-dual framework applied to minimum spanning forests. To shorten the algorithm, we observe that (a) the primal-dual framework returns the minimum spanning forest, and (b) the cuts supported by the corresponding dual certificate are precisely the greedy cuts of the minimum spanning forest. Thus `greedy-cuts` essentially refactors the algorithm analyzed by Chekuri et al. [5].

Lemma 6.2 ([5])**.** `greedy-cuts` *returns a k-cut of total cost at most* $2\left(1 - \frac{1}{n}\right)\langle c, x\rangle$.

```
greedy-cuts(G = (V,E), c, x)

1. let E' = {e ∈ E : x_e ≥ n/(2(n−1))}, E ← E \ E'
2. if E' is a k-cut then return E'
3. let F be a minimum weight spanning forest in E w/r/t x w/ ℓ components
// Arrange the greedily induced components as subtrees of a dynamic forest
4. for each v ∈ V
   A. make a singleton tree labeled by v
5. for each edge f = {u,v} ∈ F in increasing order of x_f
   A. let T_u and T_v be the rooted trees containing u and v, respectively.
   B. make T_u and T_v children of a new vertex labeled by f
// Compute the weight of each cut induced by a greedy component.
6. let each node in the dynamic forest have value 0
7. for each edge e = {u,v} ∈ E
   A. add x_e to the value of every node on the u-to-root and v-to-root paths
   B. let w be the least common ancestor u and v
   C. subtract 2x_e from the value of every node on the w to root paths
8. let v_1, v_2, ..., v_{k−ℓ} be the k−ℓ minimum value nodes in the dynamic forest.
   For i ∈ [k−ℓ], let C_i be the components induced by the leaves in the
   subtree rooted by v_i.
9. return E' ∪ ∂(C_1) ∪ ··· ∪ ∂(C_{k−ℓ})
```

Figure 2: A detailed implementation of a deterministic rounding algorithm for k-cut.

The connection to Chekuri et al. [5] is not explicitly clear because [5] rounded a slightly more complicated LP. The complication arises from the difficulty of solving (L) directly for Steiner k-cut (which can be simplified by knapsack covering constraints, in hindsight). Morally, however, their proof extends to our setting here. For the sake of completeness, a proof of Lemma 6.2 is included in Section 6.1.

It remains to implement `greedy-cuts` in $O(m \log n)$ time. With the help of dynamic trees [47], this can be done in a straightforward fashion. We briefly describe the full implementation; pseudocode is given in Figure 2. Recall from the conceptual sketch above that `greedy-cuts` requires up to $k-1$ minimum greedy cuts of a minimum spanning forest with respect to x. To compute the value of these cuts, `greedy-cuts` first simulates the greedy algorithm by processing the edges in the spanning forest in increasing order of x. The greedy algorithm repeatedly adds an edge that bridges two greedy components. We assemble a auxiliary forest of dynamic trees where each leaf is a vertex, and each subtree corresponds to a greedy component induced by the vertices at the leaves of the subtree.

After building this dynamic forest, we compute the weight of edges in each cut. We associate each node in the dynamic forest with the greedy component induced by its leaves, and give each node an initial value of 0. We process edges one at a time and add its weight to the value of every node corresponding to a greedy component cutting that edge. Now, an edge in the original graph is cut by a greedy component iff the corresponding subtree in the dynamic forest does not contain both its end points as leaves. We compute the least common ancestor of the endpoints in the dynamic forest in $O(\log n)$ time [23], and add the weight of the edge to every node between the leaves and the common ancestor, excluding the common ancestor. Adding the weight to every node on a node-to-root path takes $O(\log n)$ time [47] in dynamic trees. After processing every edge, we simply read off the value of each greedy cut as the value of the corresponding node in the forest. Thus we have the following.

Lemma 6.3. `greedy-cuts` *can be implemented in $O(m \log n)$ time.*

Together, Lemma 6.2 and Lemma 6.3 imply Theorem 1.4.

6.1 Analysis of the rounding algorithm

Chekuri et al. [5] gave a rounding scheme for the more general problem of Steiner k-cut and the analysis extends to the rounding schemes presented here. We provide a brief sketch for the sake of completeness as there are some slight technical gaps. The proof is simpler and more direct in our setting because we have a direct fractional solution to (L), while Chekuri et al. [5] dealt with a solution to a slightly more complicated LP. We note that our analysis also extends to Steiner cuts. We take as a starting point the existence of a dual certificate from the primal-dual framework.

Lemma 6.4 ([15, 16])**.** *Let F be a minimum spanning forest in a undirected graph $G = (V, E)$ weighted by $x \in \mathbb{R}^E_{\geq 0}$. Let $\mathcal{C}$ be the family of greedy cuts induced by F. Then there exists $y \in \mathbb{R}^{\mathcal{C}}_{\geq 0}$ satisfying the following properties.*[4]

(i) *For each edge $e \in E$,* $\displaystyle\sum_{C \in \mathcal{C} : C \ni e} y_C \leq x_e$.

(ii) *For each edge $e \in F$,* $\displaystyle\sum_{C \in \mathcal{C} : C \ni e} y_C = x_e$.

(iii) $\displaystyle 2\left(1 - \frac{1}{n}\right)\langle y, \mathbb{1}\rangle \geq \sum_{e \in F} x_e$.

[4] Here we do not require property (ii), but we mention it anyway because it is important in other applications.

We note that the dual variables y can be computed in $O(n)$ time (after computing the minimum spanning forest).

Lemma 6.2 ([5]). `greedy-cuts` *returns a k-cut of total cost at most* $2\left(1-\frac{1}{n}\right)\langle c,x\rangle$.

Proof sketch. Let y be as in Lemma 6.4 (applied to $E \setminus E'$). We first make two observations about y. First, since $x_e \leq \frac{n}{2(n-1)}$ for all $e \in E \setminus E'$, we have (by property (i) of Lemma 6.4) that $y_C \leq \frac{n}{2(n-1)}$ for all greedy cuts C. Second, by (a) property (iii) of Lemma 6.4 and (b) the feasibility of x with respect to the k-cut LP (L), we have

$$2\left(1-\frac{1}{n}\right)\langle y,\mathbb{1}\rangle \overset{\text{(a)}}{\geq} \sum_{e\in F} x_e \overset{\text{(b)}}{\geq} k-\ell.$$

Let $C_1,\ldots,C_{k-\ell}$ be the $k-\ell$ minimum greedy cuts. Now, by (c) rewriting the sum of the $k-\ell$ minimum greedy cuts as the solution of a minimization problem, (d) observing that $2\left(1-\frac{1}{n}\right)y$ is a feasible solution to the minimization problem, (e) interchanging sums, and (f) property (i) of Lemma 6.4, we have

$$\begin{aligned}
\sum_{i=1}^{k-\ell}\bar{c}(C_i) &\overset{\text{(c)}}{=} \min\left\{\left\langle y',\bar{c}\right\rangle : \mathbb{0} \leq y' \leq \mathbb{1},\ \text{support}\left(y'\right) \subseteq \text{support}(y),\ \text{and}\ \left\langle \mathbb{1},y'\right\rangle \geq k-\ell\right\} \\
&\overset{\text{(d)}}{\leq} 2\left(1-\frac{1}{n}\right)\langle y,\bar{c}\rangle = 2\left(1-\frac{1}{n}\right)\sum_{C} y_C \sum_{e\in C} c_e \\
&\overset{\text{(e)}}{=} 2\left(1-\frac{1}{n}\right)\sum_{e\in E\setminus E'} c_e \sum_{C\ni e} y_C \overset{\text{(f)}}{\leq} 2\left(1-\frac{1}{n}\right)\sum_{e\in E\setminus E'} c_e x_e,
\end{aligned}$$

as desired. ∎

Acknowledgements

The author thanks Chandra Chekuri for introducing him to the problem and providing helpful feedback, including pointers to the literature for rounding the LP. The author thanks Chao Xu for pointers to references. The author thanks the anonymous reviewers for their helpful comments.

References

[1] SANJEEV ARORA, ELAD HAZAN, AND SATYEN KALE: The Multiplicative Weights Update method: A meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012. [doi:10.4086/toc.2012.v008a006] 8

[2] FRANCISCO BARAHONA: On the k-cut problem. *Oper. Res. Lett.*, 26(3):99–105, 2000. [doi:10.1016/S0167-6377(99)00071-1] 3

[3] ROBERT D. CARR, LISA FLEISCHER, VITUS J. LEUNG, AND CYNTHIA A. PHILLIPS: Strengthening integrality gaps for capacitated network design and covering problems. In *Proc. 11th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA'00)*, pp. 106–115. SIAM, 2000. ACM DL. 5, 7

[4] DEEPARNAB CHAKRABARTY, CHANDRA CHEKURI, SANJEEV KHANNA, AND NITISH KORULA: Approximability of capacitated network design. *Algorithmica*, 72(2):493–514, 2015. Preliminary version in IPCO'11. [doi:10.1007/s00453-013-9862-4, arXiv:1009.5734] 7

[5] CHANDRA CHEKURI, SUDIPTO GUHA, AND JOSEPH NAOR: The Steiner k-cut problem. *SIAM J. Comput.*, 20(1):261–271, 2006. Preliminary version in ICALP'03. [doi:10.1137/S0895480104445095] 4, 6, 15, 16, 17, 18, 19

[6] CHANDRA CHEKURI AND KENT QUANRUD: Approximating the Held–Karp bound for metric TSP in nearly-linear time. In *Proc. 58th FOCS*, pp. 789–800. IEEE Comp. Soc., 2017. [doi:10.1109/FOCS.2017.78, arXiv:1702.04307] 7, 9, 10, 13, 14

[7] CHANDRA CHEKURI AND KENT QUANRUD: Near-linear time approximation schemes for some implicit fractional packing problems. In *Proc. 28th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA'17)*, pp. 801–820. SIAM, 2017. [doi:10.1137/1.9781611974782.51] 4, 5, 10, 13, 14, 15

[8] CHANDRA CHEKURI AND KENT QUANRUD: Fast approximations for metric-TSP via linear programming, 2018. [arXiv:1802.01242] 7

[9] CHANDRA CHEKURI AND KENT QUANRUD: Randomized MWU for positive LPs. In *Proc. 29th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA'18)*, pp. 358–377. SIAM, 2018. [doi:10.1137/1.9781611975031.25] 4, 5, 6, 15

[10] CHANDRA CHEKURI AND KENT QUANRUD: On approximating (sparse) covering integer programs. In *Proc. 30th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA'19)*, pp. 1596–1615. SIAM, 2019. [doi:10.1137/1.9781611975482.97, arXiv:1807.11538] 7, 10, 13, 14

[11] CHANDRA CHEKURI, KENT QUANRUD, AND CHAO XU: LP relaxation and tree packing for minimum k-cuts. *SIAM J. Discr. Math.*, 34(2), 2020. Preliminary version in SOSA'19. [doi:10.1137/19M1299359, arXiv:1808.05765] 2

[12] RAJESH CHITNIS, MAREK CYGAN, MOHAMMADTAGHI HAJIAGHAYI, MARCIN PILIPCZUK, AND MICHAL PILIPCZUK: Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016. Preliminary version in FOCS'12. [doi:10.1137/15M1032077, arXiv:1207.4079] 6

[13] RODNEY G. DOWNEY, VLADIMIR ESTIVILL-CASTRO, MICHAEL R. FELLOWS, ELENA PRIETO-RODRIGUEZ, AND FRANCES A. ROSAMOND: Cutting up is hard to do: The parameterized complexity of k-cut and elated problems. *Electr. Notes Theoret. Comp. Sci.*, 78:209–222, 2003. [doi:10.1016/S1571-0661(04)81014-4] 6, 7

[14] 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. Preliminary version in FOCS'98. [doi:10.1137/S0097539704446232] 8, 9

[15] MICHEL X. GOEMANS AND DAVID P. WILLIAMSON: A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995. Preliminary version in SODA'92. [doi:10.1137/S0097539793242618] 4, 6, 15, 16, 18

[16] MICHEL X. GOEMANS AND DAVID P. WILLIAMSON: The primal-dual method for approximation algorithms and its applications to network design problems. In DORIT S. HOCHBAUM, editor, *Approximation Algorithms for NP-Hard Problems*, pp. 144–191. PWS Publishing Company, Boston, MA, 1996. ACM DL. 15, 16, 18

[17] ANDREW V. GOLDBERG AND SATISH RAO: Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998. Preliminary version in FOCS'97. [doi:10.1145/290179.290181] 3

[18] OLIVIER GOLDSCHMIDT AND DORIT S. HOCHBAUM: A polynomial algorithm for the k-cut problem for fixed k. *Math. Oper. Res.*, 19(1):24–37, 1994. Preliminary version in FOCS'88. [doi:10.1287/moor.19.1.24] 2

[19] ANUPAM GUPTA, EUIWOONG LEE, AND JASON LI: Faster exact and approximate algorithms for k-cut. In *Proc. 59th FOCS*. IEEE Comp. Soc., 2018. [doi:10.1109/FOCS.2018.00020, arXiv:1807.08144] 2, 7

[20] ANUPAM GUPTA, EUIWOONG LEE, AND JASON LI: An FPT algorithm beating 2-approximation for k-cut. In *Proc. 29th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA'18)*, pp. 2821–2837. SIAM, 2018. [doi:10.1137/1.9781611975031.179, arXiv:1710.08488] 7

[21] ANUPAM GUPTA, EUIWOONG LEE, AND JASON LI: The number of minimum k-cuts: improving the Karger–Stein bound. In *Proc. 51st STOC*, pp. 229–240. ACM Press, 2019. [doi:10.1145/3313276.3316395, arXiv:1906.00417] 2

[22] JIANXIU HAO AND JAMES B. ORLIN: A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424–446, 1994. Preliminary version in SODA'92. [doi:10.1006/jagm.1994.1043] 2

[23] DOV HAREL AND ROBERT ENDRE TARJAN: Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. [doi:10.1137/0213024] 18

[24] MONIKA HENZINGER, SATISH RAO, AND DI WANG: Local flow partitioning for faster edge connectivity. *SIAM J. Comput.*, 49(1):1–36, 2020. Preliminary version in SODA'17. [doi:10.1137/18M1180335, arXiv:1704.01254] 2

[25] JACOB HOLM, KRISTIAN DE LICHTENBERG, AND MIKKEL THORUP: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. Preliminary version in STOC'98. [doi:10.1145/502090.502095] 11

[26] DAVID R. KARGER: *Random Sampling in Graph Optimization Problems*. Ph. D. thesis, Stanford University, 1994. Stanford CS TR-95-1541. 2

[27] DAVID R. KARGER: Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. Preliminary version in STOC'96. [doi:10.1145/331605.331608, arXiv:cs/9812007] 2

[28] DAVID R. KARGER AND CLIFFORD STEIN: A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996. Preliminary version in STOC'93. [doi:10.1145/234533.234534] 2

[29] KEN-ICHI KAWARABAYASHI AND MIKKEL THORUP: The minimum k-way cut of bounded size is fixed-parameter tractable. In *Proc. 52nd FOCS*, pp. 160–169. IEEE Comp. Soc., 2011. [doi:10.1109/FOCS.2011.53] 6

[30] KEN-ICHI KAWARABAYASHI AND MIKKEL THORUP: Deterministic global minimum cut of a simple graph in near-linear time. In *Proc. 47th STOC*, pp. 665–674. ACM Press, 2015. [doi:10.1145/2746539.2746588, arXiv:1411.5123] 2

[31] STAVROS G. KOLLIOPOULOS AND NEAL E. YOUNG: Approximation algorithms for covering/packing integer programs. *J. Comput. System Sci.*, 71(4):495–505, 2005. Preliminary version in FOCS'01. [doi:10.1016/j.jcss.2005.05.002, arXiv:cs/0205030] 7

[32] YIN TAT LEE AND AARON SIDFORD: Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\sqrt{\text{rank}})$ iterations and faster algorithms for maximum flow. In *Proc. 55th FOCS*, pp. 424–433. IEEE Comp. Soc., 2014. [doi:10.1109/FOCS.2014.52] 3

[33] MATTHEW S. LEVINE: Fast randomized algorithms for computing minimum $\{3,4,5,6\}$-way cuts. In *Proc. 11th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA'00)*, pp. 735–742. SIAM, 2000. ACM DL. 2

[34] ON-HEI SOLOMON LO, JENS M. SCHMIDT, AND MIKKEL THORUP: Compact cactus representation of all non-trivial min-cuts. *Discr. Appl. Math.*, 303:296–304, 2021. [doi:10.1016/j.dam.2020.03.046, arXiv:1810.03865] 2

[35] ALEKSANDER MADRY: Computing maximum flow with augmenting electrical flows. In *Proc. 57th FOCS*, pp. 593–602. IEEE Comp. Soc., 2016. [doi:10.1109/FOCS.2016.70, arXiv:1608.06016] 3

[36] PASIN MANURANGSI: Inapproximability of maximum edge biclique, maximum balanced biclique and minimum k-cut from the small set expansion hypothesis. In *Proc. 44th Internat. Colloq. on Automata, Languages, and Programming (ICALP'17)*, volume 80, pp. 79:1–14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. [doi:10.4230/LIPIcs.ICALP.2017.79] 3, 4, 7

[37] DAVID W. MATULA: A linear time $2+\varepsilon$ approximation algorithm for edge connectivity. In *Proc. 4th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA'93)*, pp. 500–504. SIAM, 1993. ACM DL. 2

[38] HIROSHI NAGAMOCHI AND TOSHIHIDE IBARAKI: Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discr. Math.*, 5(1):54–66, 1992. Preliminary version in SIGAL'90. [doi:10.1137/0405004] 2

[39] HIROSHI NAGAMOCHI AND TOSHIHIDE IBARAKI: A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. [doi:10.1007/BF01758778] 2

[40] HIROSHI NAGAMOCHI AND YOKO KAMIDOI: Minimum cost subpartitions in graphs. *Inform. Process. Lett.*, 102(2–3):79–84, 2007. [doi:10.1016/j.ipl.2006.11.011] 3, 4

[41] JOSEPH NAOR AND YUVAL RABANI: Tree packing and approximating k-cuts. In *Proc. 12th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA'01)*, pp. 26–27. SIAM, 2001. ACM DL. 3, 4

[42] CRISPIN ST. J. A. NASH-WILLIAMS: Edge-disjoint spanning trees of finite graphs. *J. London Math. Soc.*, 36(1):445–450, 1961. [doi:10.1112/jlms/s1-36.1.445] 2

[43] KENT QUANRUD: Fast and deterministic approximations for k-cut. In *Proc. 22nd Internat. Conf. on Approximation Algorithms for Combinat. Opt. Probl. (APPROX'19)*, pp. 23:1–20. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. [doi:10.4230/LIPIcs.APPROX-RANDOM.2019.23, arXiv:1807.07143] 1

[44] PRASAD RAGHAVENDRA AND DAVID STEURER: Graph expansion and the Unique Games Conjecture. In *Proc. 42nd STOC*, pp. 755–764. ACM Press, 2010. [doi:10.1145/1806689.1806792] 3

[45] R. RAVI AND AMITABH SINHA: Approximating k-cuts using network strengths as a Lagrangean relaxation. *Eur. J. Operational Res.*, 186(1):77–90, 2008. Preliminary version in SODA'02. [doi:10.1016/j.ejor.2007.01.040] 3

[46] HUZUR SARAN AND VIJAY V. VAZIRANI: Finding k cuts within twice the optimal. *SIAM J. Comput.*, 24(1):101–108, 1995. Preliminary version in FOCS'91. [doi:10.1137/S0097539792251730] 2, 3, 4, 6

[47] DANIEL DOMINIC SLEATOR AND ROBERT ENDRE TARJAN: A data structure for dynamic trees. *J. Comput. System Sci.*, 26(3):362–391, 1983. Preliminary version in STOC'81. [doi:10.1016/0022-0000(83)90006-5] 18

[48] MECHTHILD STOER AND FRANK WAGNER: A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997. Preliminary version in ESA'94. [doi:10.1145/263867.263872] 2

[49] MIKKEL THORUP: Minimum k-way cuts via deterministic greedy tree packing. In *Proc. 40th STOC*, pp. 159–166. ACM Press, 2008. [doi:10.1145/1374376.1374402] 2, 10

[50] WILLIAM T. TUTTE: On the problem of decomposing a graph into n connected components. *J. London Math. Soc.*, 36(1):221–230, 1961. [doi:10.1112/jlms/s1-36.1.221] 2

[51] MINGYU XIAO, LEIZHEN CAI, AND ANDREW CHI-CHIH YAO: Tight approximation ratio of a general greedy splitting algorithm for the minimum k-way cut problem. *Algorithmica*, 59(4):510–520, 2011. [doi:10.1007/s00453-009-9316-1, arXiv:0811.3723] 6, 7

[52] NEAL E. YOUNG: Sequential and parallel algorithms for mixed packing and covering. In *Proc. 42nd FOCS*, pp. 538–546. IEEE Comp. Soc., 2001. [doi:10.1109/SFCS.2001.959930, arXiv:cs/0205039] 8

[53] NEAL E. YOUNG: Nearly linear-work algorithms for mixed packing/covering and facility-location linear programs, 2014. [arXiv:1407.3015] 4, 5, 14

AUTHOR

Kent Quanrud
Assistant professor
Department of Computer Science
Purdue University
West Lafayette, Indiana, USA
krq@purdue.edu
http://kentquanrud.com

ABOUT THE AUTHOR

Kent Quanrud is an Assistant Professor of Computer Science at Purdue University. He received his Ph. D. from the University of Illinois at Urbana-Champaign in 2019, where he was advised by Chandra Chekuri and Sariel Har-Peled. He was born in Japan and grew up in California. His research interests include approximation algorithms, randomized algorithms, combinatorial optimization, continuous optimization, online learning, and computational geometry. He is particularly interested in extremely scalable algorithms for fundamental problems in optimization.