An FPTAS for Connectivity Interdiction

Chien-Chung Huang, Nidia Obscura Acosta, Sorrachai Yingchareonthawornchai

An FPTAS for Connectivity Interdiction*

Chien-Chung Huang[1], Nidia Obscura Acosta[2], and Sorrachai Yingchareonthawornchai[3]

[1] Department of Computer Science, École Normale Supérieure, Paris, France.
cchuang@di.ens.fr
[2] Department of Computer Science, Aalto University, Espoo, Finland.
nidia.obscuraacosta@aalto.fi
[3] The Hebrew University of Jerusalem, Jerusalem, Israel. sorrachai.cp@gmail.com

Abstract. In the connectivity interdiction problem, we are asked to find a global graph cut and remove a subset of edges under a budget constraint, so that the total weight of the remaining edges in this cut is minimized. This problem easily includes the knapsack problem as a special case, hence it is NP-hard. For this problem, Zenklusen [Zen14] designed a polynomial-time approximation scheme (PTAS), and for the special case of unit edge costs, exact algorithms. He posed the question of whether a fully polynomial-time approximation scheme (FPTAS) is possible for the general case. We give an affirmative answer. For the special case of unit edge costs, we also give faster exact and approximation algorithms.

Our main technical contribution is to establish a connection with an intermediate graph cut problem, called the *normalized* min-cut, which, roughly speaking, penalizes the edge weights of the remaining edges more severely, when more edges are taken out for free.

1 Introduction

We study the *connectivity interdiction problem* [Zen14]. In this problem, we are given an instance $(G = (V, E), w, c, b)$, where G is an undirected multi-graph which has weights $w : E \to \mathbb{Z}_{>0}$ and costs $c : E \to \mathbb{Z}_{>0}$ associated with its edges. A subset of edges $F \subseteq E$ is feasible if $c(F) = \sum_{e \in F} c(e)$ is no more than the budget $b \in \mathbb{Z}_{>0}$. The objective is to remove a feasible set of edges so that in the remaining graph $G' = (V, E \backslash F)$, the minimum weight of a global cut is minimized. Connectivity interdiction arises as a natural optimization problem in graphs. The potential applications include drug delivery intersection [Woo93], nuclear smuggling interdiction [MPS07], security of electric grids under terrorist attacks [SWB04], hospital infection control [Ass87], the vulnerability of the network infrastructure [MM09], and strategic military planning in the battlefield [GMT71].

* A preliminary version of this paper has appeared in Proceeding of Integer Programming and Combinatorial Optimization - 25th International Conference, IPCO 2024, Wrocław, Poland, July 3-5, 2024.

We note that connectivity interdiction can also be regarded as the *b-free min-cut problem*, that is, we aim at finding a vertex set $S \subsetneq V$ and a feasible subset of edges $F \subseteq \delta(S)$ (where $\delta(S)$ is the set of edges with exactly one endpoint in S), so that $\sum_{e\in F} c(e) \leq b$ and $\sum_{e\in\delta(S)\setminus F} w(e)$ is minimized.

It is easy to see that even in a graph as simple as two vertices connected by parallel edges, the b-free min-cut problem already contains the knapsack problem as a special case, and hence is NP-hard. A very natural special case is that of unit edge costs, that is when $c(e) = 1$ for all $e \in E$. This case is known to be in P.

In the rest of this paper, we assume that $n = |V|$ and $m = |E|$. For convenience, we define a cut as the set of edges instead of the set of vertices. That is, we say that a set of edges C is a cut if $C = \delta(S)$ for some $S \subsetneq V$. We denote the weight of a cut C, as $w(C) = \sum_{e\in C} w(e)$: the total sum of the weight of its edges. Throughout the paper, the notation $\tilde{O}(\cdot)$ hides poly-logarithmic factors in n and m. Zenklusen [Zen14] has shown the following two results for the connectivity problem.

- A PTAS for arbitrary edge costs.
- For the unit edge costs, an $\tilde{O}(m^2n^4)$ deterministic exact algorithm and an $\tilde{O}(mn^4)$ randomized exact algorithm[4].

In [Zen14] Zenklusen has posed the open question whether it is possible to obtain an FPTAS with arbitrary edge costs.

1.1 Our Results and Techniques

We give an affirmative answer to Zenklusen's question, showing an FPTAS for arbitrary edge costs. Let $(G = (V, E), w, c, b)$ be an instance of the b-free min-cut problem.

Theorem 1 (FPTAS). *With arbitrary edge costs, there is a fully polynomial time approximation scheme for the b-free min-cut problem: given $\delta > 0$, in time $O(\frac{m^2n^4}{\delta}\log(nw_{max}b))$, where $w_{\max} = \max_{e\in E} w(e)$, we can find an $(1+\delta)$-approximate solution.*

For the special case of unit edge costs, we also improve on the running time of Zenklusen. We design three algorithms that offer varying trade-offs between the running time and the accuracy.

Theorem 2. *With unit edge costs and for positive accuracy parameter $\varepsilon < 1/100$, we can*

1. *find an exact solution in $\tilde{O}(m + n^4b)$ time;*
2. *or find an $(1+\varepsilon)$-approximate solution in $\tilde{O}(\frac{m}{\varepsilon} + n^3b)$ time;*
3. *or find an $(2+\varepsilon)$-approximate solution in $\tilde{O}(\frac{m}{\varepsilon} + n^2b)$ time.*

All these algorithms are randomized and succeed with high probability.

[4] $\tilde{O}(\cdot)$ hides poly-logarithmic factors.

Our Proof Method. Let positive $\varepsilon < 1/100$ be an accuracy parameter. Let $\alpha_1 = 2 + 2\varepsilon, \alpha_2 = 2 - 4\varepsilon, \alpha_3 = 1.5 - 2\varepsilon$. We say that a cut C is an α-approximate cut if the weight of C is at most α factor times the value of a global min-cut. The key approach is to show the following.

Theorem 3. *We can re-define a weight function $\tilde{w}^{(i)}$ where $i \in \{1, 2, 3\}$ in the same graph G so that at least one α_i-approximate min-cut C w.r.t. $\tilde{w}^{(i)}$ contains a feasible edge-set $F \subseteq C$ such that $w(C \setminus F)$ is β_i-approximation to the b-free min-cut problem where $\beta_1 = 1, \beta_2 = 1 + O(\varepsilon), \beta_3 = 2 + O(\varepsilon)$.*

To design an FPTAS for general connectivity interdiction, we use the weight function[5] $\tilde{w}^{(1)}$ as stated in Theorem 3 for which one of the α_1-approximate cut C (w.r.t. the weight function $\tilde{w}^{(1)}$) has a feasible set $F \subseteq C$ such that $w(C \setminus F)$ is an optimal solution to the interdiction problem. Then, we enumerate all α_1-approximate cuts in $\tilde{O}(n^{\lfloor 2\alpha_1 \rfloor}) = \tilde{O}(n^4)$ time using Nagamochi et. al [NNI97] cut enumeration algorithm. Given a cut, computing the optimal feasible edge-set corresponds to a knapsack minimization problem, where an FPTAS is known [GL80].

Remark 1. Since our algorithm is a polynomial-time reduction to a knapsack minimization problem, this also implies a pseudo-polynomial time algorithm for general connectivity interdiction by using the pseudo-polynomial time algorithm for the knapsack minimization problem [Isl09]. For details, see Remark 2 in Section 3.

For the case of unit edge costs, we could use any $\tilde{w}^{(i)}$ where $i \in \{1, 2, 3\}$. By Theorem 3, either exact, $1+O(\varepsilon)$-,$2+O(\varepsilon)$-approximate solutions for connectivity interdiction can be found in the enumeration of α_i-approximate cuts where $i = 1, 2, 3$, respectively. By adapting the tree packing theorem [Kar00], we can show that the number of approximate cuts is $O(n^4), O(n^3), O(n^2)$, respectively. Given a cut, the optimal feasible edge set corresponds to the b heaviest edges in the cut w.r.t. w (since the cost is uniform).

Here as our goal is to have really fast algorithms; it would be inefficient to "explicitly" list all the α_i-approximate cuts and then sum up the edge weights after removing the heaviest b edges. To do so, we design a data structure that can output the value $w(C \setminus F)$ quickly. Such a data structure uses several ideas from a recent work of Chekuri and Quanrud [CQ17a] (cf. [Kar00]). They introduced the data structures for enumerating and evaluating the weights of $O(n^2)$ many $(1+\epsilon)$-approximate cuts. We extend their framework to enumerate and evaluate the weights of $O(n^3)$ and $O(n^4)$ approximate cuts. To evaluate $w(C \setminus F) = w(C) - w(F)$ quickly, we use priority queues on top of their data structures to evaluate $w(F)$ in $\tilde{O}(b)$ time. This explains the terms $\tilde{O}(n^4 b), \tilde{O}(n^3 b), \tilde{O}(n^2 b)$, respectively in Theorem 2.

[5] we will discuss how to compute $w^{(1)}$ efficiently later.

Normalized min-cut. The proof of Theorem 3 hinges on an intermediate problem that we call *the normalized min-cut problem*, defined as follows.

Definition 1 (Normalized min-cut). *Given (G, w, c, b) (originally an instance of a b-free min-cut problem), here the objective is to find a cut C, and a subset of its edges $F \subset C$ satisfying $0 \leq c(F) \leq b$, so that $\frac{w(C \setminus F)}{\delta_{c(F)}}$ is minimized where $\delta_i := b - i + 1$.*

Intuitively, the cost of a cut is "normalized" according to the total cost of edges that are taken out: the less costly the subset F taken out, the more heavily the rest of the edges $C \setminus F$ would be normalized.

In fact, originally, the normalized min-cut problem arises in a different (but related) context. In a recent paper of Chalermsook et el. [CHN+22] on survivable network design, the same problem was first introduced (under a different name "minimum normalized free cut") to deal with a certain boxing constraint in the LPs. There, a special case of unit-edge costs is actually *solved as a technical necessity.* To obtain an FPTAS in this paper, we emphasize that we do not need to solve the normalized min-cut problem per se, but rather we use its optimal solution as a certificate in the analysis of the weight function $\tilde{w}^{(i)}$ in Theorem 3. We introduce another notation to facilitate our discussion.

Definition 2. *For each $0 \leq i \leq b$, let $\lambda_i = \min_{C, F \subseteq C, c(F)=i} w(C \setminus F)$. (In case that there is no C and $F \subseteq C$ so that $c(F) = i$, λ_i is understood to be infinity.)*

Furthermore, let OPT and OPT^N denote the value of the optimal solution for the b-free min-cut and the normalized min-cut, repsectively. Then by definition,

$$\mathsf{OPT} = \min_{0 \leq i \leq b} \lambda_i, \qquad \mathsf{OPT}^N = \min\{\frac{\lambda_0}{b+1}, \frac{\lambda_1}{b}, \ldots, \frac{\lambda_i}{b-i+1}, \ldots, \frac{\lambda_{b-1}}{2}, \frac{\lambda_b}{1}\}.$$

Note that when c is unit edge costs, $\mathsf{OPT} = \lambda_b$, but for arbitrary costs this might not be true.

For both b-free min-cut and normalized min-cut, a solution involves a cut C and its feasible set of edges $F \subseteq C$. In our presentation, we often write them as a pair (C, F).

We now explain how the normalized min-cut problem comes into the picture. To define the intermediate weight function, we *truncate* the weights as follows.

Definition 3. *Let τ be a parameter. Define a global min-cut problem $(G = (V, E), \tilde{w}_\tau)$ by truncating the weights w as follows:*

$$\tilde{w}_\tau(e) = \begin{cases} \tau \cdot c(e) & \text{if } w(e) \geq \tau \cdot c(e), \\ w(e) & \text{otherwise.} \end{cases}$$

An edge $e \in E$ is heavy *if $w(e) \geq \tau \cdot c(e)$. Otherwise, it is* light.

Assuming that we are given an estimate τ of OPT^N (we will explain later how such guesses can be made and how to bound the number of such guesses). The following relations of the various optimal cuts (of different problems) is crucial.

Theorem 4 (Main).
Let (C^N, F^N) be the solution that realizes OPT^N in the normalized min-cut problem, i.e., $\mathsf{OPT}^N = \frac{w(C^N \setminus F^N)}{b - c(F^N)+1}$. Similarly, let (C^*, F^*) be a solution that realizes OPT in the b-free min-cut problem, i.e., $w(C^* \setminus F^*) = \mathsf{OPT}$.

Suppose that $\mathsf{OPT}^N(1-\varepsilon) \leq \tau \leq \mathsf{OPT}^N$ where $0 < \varepsilon \leq 1/100$. Consider the instance $(G, \tilde{w}_\tau)$ of the global min-cut problem, and denote $C^{\min}$ as an optimal solution.

Finally, let $\alpha_1 = 2 + 2\varepsilon, \alpha_2 = 2 - 4\varepsilon$, and $\alpha_3 = 1.5 - 2\varepsilon$. Then the following facts hold.

(i) $\tilde{w}_\tau(C^*) \leq \alpha_1 \cdot \tilde{w}_\tau(C^{\min})$. *That is, C^* is α_1-approximate min-cut w.r.t. $\tilde{w}_\tau$.*
(ii) If $\tilde{w}_\tau(C^*) > \alpha_2 \cdot \tilde{w}_\tau(C^{\min})$, then $\tilde{w}_\tau(C^N) \leq \alpha_2 \cdot \tilde{w}_\tau(C^{\min})$ and (C^N, F^N) is an $(1 + O(\varepsilon))$-approximate solution to the b-free min-cut problem.
(iii) If $\tilde{w}_\tau(C^*) > \alpha_3 \cdot \tilde{w}_\tau(C^{\min})$, then $\tilde{w}_\tau(C^N) \leq \alpha_3 \cdot \tilde{w}_\tau(C^{\min})$ and (C^N, F^N) is an $(2 + O(\varepsilon))$-approximate solution to the b-free min-cut problem.

Theorem 4 immediately implies Theorem 3, and gives us a trade-off in terms of the number of cuts to enumerate and the quality of the solutions from the normalized min-cut problem. Namely, the number of approximate cuts w.r.t. $\tilde{w}$ to enumerate is $O(n^{\lfloor 2\alpha_1 \rfloor}), O(n^{\lfloor 2\alpha_2 \rfloor}), O(n^{\lfloor 2\alpha_3 \rfloor})$, which is $O(n^4), O(n^3), O(n^2)$, respectively, in order to obtain exact, $(1 + O(\varepsilon))$ and $(2 + O(\varepsilon))$ approximate solutions respectively. To remove the assumption, $\mathsf{OPT}^N(1-\varepsilon) \leq \tau \leq \mathsf{OPT}^N$, we guess the value of τ by the geometric series $(1+\varepsilon)^i$ in an appropriate range.

Other Related Work. For many optimization problems, one can introduce a corresponding interdiction problem. See [SS19] for a recent survey on the interdiction problem in general. Roughly speaking, in the interdiction problem, there is a certain "interdictor", who tries to worsen the optimal value of the optimization problem as much as possible. The present work focuses on the global cut problem. But in fact, a large variety of classical optimization problems have been studied from the point of view of interdiction. Examples include: shortest paths [MMG89,SBNK95,BGV89,BSR20], network design [ASG13,DXT+12], minimum spanning tree [Zen15], graph matching [Zen10], clustering [BTV10], and network flows [CZ17].

1.2 Organization

In section 2, we establish the connections between the normalized min-cut and the b-free min-cut problems, which allows us to prove our main technical result, theorem 4. In section 3, we show our FPTAS for general edge costs, and in section 4 we show our algorithms for unit edge costs. Appendix 5.1 and 5.2 contain the details of implementing the data structures described in section 4.

2 The Normalized Min-cut Problem

We prove Theorem 4 in this section.

Proof Outline. We first show the two structural properties of the optimal solution (C^N, F^N) to the normalized min-cut problem: (1) every $e \in F^N$ is heavy, i.e., every edge in the optimal free-edge set is heavy, and (2) C^N is an $(1+2\varepsilon)$-approximate cut in the new weight function $\tilde{w}_\tau$ (Definition 3). Next, we prove that if C^* is not α-approximate cut in $\tilde{w}_\tau$, then we can establish a good lower bound on OPT in the original weight function w. This allows us to show that the optimal solution to the normalized min-cut (C^N, F^N) is a good approximate solution to the b-free min-cut problem. Depending on the parameter α, we obtain different guarantees as shown in Theorem 4. We now formalize the proofs.

Structures of (C^N, F^N). The following fact will be useful.

Lemma 1. *Suppose that OPT^N is realized by (C^N, F^N), namely, $\mathsf{OPT}^N = \frac{w(C^N \setminus F^N)}{\delta_{c(F)}}$. Then every edge $e \in F^N$ is a heavy edge.*

Proof. Suppose, for a contradiction, that $e \in F^N$ is a light edge. Then $w(e) < \tau c(e)$. Let $F' = F^N \setminus \{e\}$. Then

$$\frac{w(C^N \setminus F')}{\delta_c(F')} = \frac{w(C^N \setminus F^N) + w(e)}{\delta_{c(F^N)} + c(e)} < \frac{w(C^N \setminus F^N) + \tau c(e)}{\delta_{c(F^N)} + c(e)}$$

$$\leq \frac{w(C^N \setminus F^N) + (\frac{w(C^N \setminus F^N)}{\delta_{c(F^N)}})c(e)}{\delta_{c(F^N)} + c(e)} = \frac{w(C^N \setminus F^N)}{\delta_{c(F^N)}}$$

where in the first inequality we use the fact that $\tau \leq \mathsf{OPT}^N$. As $c(F') \leq c(F^N) \leq b$, we have obtained a contradiction, since the pair (C^N, F') reaches a value smaller than OPT^N.

A main subroutine of our algorithms is to enumerate all α-approximate cuts in the global cut instance $(G, \tilde{w}_\tau)$. We will first establish a lower bound on the optimal value of the global cut in the latter.

Lemma 2. *Given any cut C in $(G, \tilde{w}_\tau)$, then $\tilde{w}_\tau(C) \geq \tau(1+b)$.*

Proof. Let $F \subseteq C$ be the set of heavy edges of C. Suppose that $c(F) \geq b+1$. Then $\tilde{w}_\tau(C) \geq \tilde{w}_\tau(F) \geq \tau c(F) \geq \tau(1+b)$. On the other hand, suppose that $c(F) = i < b+1$. Then

$$\tilde{w}_\tau(C) = \tilde{w}_\tau(F) + \tilde{w}_\tau(C \setminus F) = i\tau + w(C \setminus F) \geq i\tau + \mathsf{OPT}^N \delta_i \geq i\tau + \tau\delta_i = \tau(1+b),$$

where in the second inequality we use the fact that $w(C \setminus F) \geq \lambda_i \geq \mathsf{OPT}^N \delta_i$.

Lemma 3. *Suppose that OPT^N is realized by (C^N, F^N). Furthermore, $C^{\min}$ is the optimal global cut in the instance $(G, \tilde{w}_\tau)$. Then*

$$\frac{\tilde{w}_\tau(C^N)}{\tilde{w}_\tau(C^{\min})} < 1 + 2\varepsilon,$$

assuming that $\varepsilon < 1/2$.

Proof. By Lemma 2, we already know that $\tilde{w}_\tau(C^{\min}) \geq \tau(1+b)$. It remains to upper bound $\tilde{w}_\tau(C^N)$. By Lemma 1, all edges in F^N are heavy. Thus $\tilde{w}_\tau(F^N) = \tau c(F^N)$. Furthermore, as $\tilde{w}_\tau(C^N \backslash F^N) \leq w(C^N \backslash F^N) = \delta_{c(F^N)}\mathsf{OPT}^N$. Therefore

$$\tilde{w}_\tau(C^N) \leq \tau c(F^N) + \delta_{c(F^N)}\mathsf{OPT}^N \leq \tau c(F^N) + \frac{\tau}{1-\varepsilon}\delta_{c(F^N)}$$

$$\leq \frac{\tau}{1-\varepsilon}(c(F^N) + \delta_{c(F^N)}) = \frac{\tau}{1-\varepsilon}(1+b).$$

In conclusion, we have $\frac{\tilde{w}_\tau(C^N)}{\tilde{w}_\tau(C^{\min})} < 1/(1-\varepsilon) < 1 + 2\varepsilon$.

Lower bound on OPT. The next lemma states that if after enumerating all α-approximate cuts in $(G, \tilde{w}_\tau)$, we still cannot find an optimal solution in the original b-free min-cut instance, we can then establish a lower bound on OPT.

Lemma 4. *Let C^* denote the optimal solution in the b-free min-cut instance (G, w, c, b). Namely, there exists $F^* \subseteq C^*$ so that $c(F^*) \leq b$ and $w(C^* \backslash F^*) = \mathsf{OPT}$. Then either C^* is an α-approximate cut in the instance $(G, \tilde{w}_\tau)$, for some $\alpha \geq 1$, or $\mathsf{OPT} \geq \tau(\alpha + (\alpha-1)b)$.*

Proof. Suppose that C^* is not an α-approximate cut in $(G, \tilde{w}_\tau)$. Let $C^{\min}$ denote the optimal global cut in $(G, \tilde{w}_\tau)$. Then

$$\alpha \leq \frac{\tilde{w}_\tau(C^*)}{\tilde{w}_\tau(C^{\min})} \leq \frac{\tilde{w}_\tau(F^*) + \tilde{w}_\tau(C^* \backslash F^*)}{\tau(1+b)} \leq \frac{c(F^*)\tau + w(C^* \backslash F^*)}{\tau(1+b)} \leq \frac{b\tau + \mathsf{OPT}}{\tau(1+b)},$$

where the second inequality uses Lemma 2.

Corollary 1. *Suppose that OPT^N and OPT are realized by (C^N, F^N) and (C^*, F^*) respectively, namely, $\mathsf{OPT}^N = \frac{w(C^N \backslash F^N)}{\delta_{c(F^N)}}$, and $w(C^* \backslash F^*) = \mathsf{OPT}$. Furthermore, C^* is not an α-approximate cut in $(G, \tilde{w}_\tau)$ for some $\alpha \geq 1$. Then*

$$\frac{w(C^N \backslash F^N)}{w(C^* \backslash F^*)} \leq \frac{1+b}{(1-\varepsilon)(\alpha + (\alpha-1)b)}.$$

Proof.

$$\frac{w(C^N \backslash F^N)}{w(C^* \backslash F^*)} = \frac{\mathsf{OPT}^N \cdot \delta_{c(F^N)}}{\mathsf{OPT}} \leq \frac{\mathsf{OPT}^N \cdot \delta_{c(F^N)}}{\tau(\alpha + (\alpha-1)b)}$$

$$\leq \frac{\frac{\tau}{1-\varepsilon}(1+b-c(F^N))}{\tau(\alpha + (\alpha-1)b)} \leq \frac{1+b}{(1-\varepsilon)(\alpha + (\alpha-1)b)},$$

where the first inequality uses Lemma 4.

Finishing the proof. We are now ready to prove Theorem 4. For each item of the statement, we need to set an appropriate value of α. We next show that by setting α to be suitably large, the optimal solution in b-free min-cut must be an α-approximate cut in $(G, \tilde{w}_\tau)$—this leads to an FPTAS for non-uniform cost and an exact algorithm for uniform costs.

Theorem 5. *Suppose that $\alpha = 2 + 2\varepsilon$, with $0 < \varepsilon \leq 1/2$. Then the optimal cut C^* that realizes* OPT *must be α-approximate cut in $(G, \tilde{w}_\tau)$.*

Proof. For a contradiction, suppose that C^* is not α-approximate in $(G, \tilde{w}_\tau)$. Then by Corollary 1,

$$\frac{w(C^N \backslash F^N)}{w(C^* \backslash F^*)} \leq \frac{1+b}{(1-\varepsilon)(\alpha + (\alpha-1)b)} < 1,$$

where the last inequality can be easily verified. Then we reach a contradiction that $C^N \backslash F^N$ is an even cheaper solution than $C^* \backslash F^*$.

Proof (Proof of Theorem 4).

We first prove part 1. Let $\alpha = 2 + 2\varepsilon$ with $\varepsilon = 0.01$. Theorem 5 already gives us that the optimal cut C^* which realizes OPT must be a $2 + 2\varepsilon$-approximate cut in $(G, \tilde{w}_\tau)$ as desired.

We next proceed to prove Parts 2 & 3. First note that by lemma 3, $\tilde{w}_\tau(C^N) \leq \alpha \cdot \tilde{w}_\tau(C^{\min})$ for $\alpha = 2 - 4\varepsilon$ and $\alpha = 1.5 - 2\varepsilon$.

Let $\alpha = 2 - 4\varepsilon$, note that contrary to part 1, we cannot ensure that the cut C^* which obtains OPT will be a $2-4\varepsilon$-approx cut in $(G, \tilde{w}_\tau)$. However, we know that by corollary 1 the cut C^N achieving the optimal value in the normalized mincut problem is an approximation to the optimum b-free min cut as follows:

$$\frac{w(C^N \backslash F^N)}{w(C^* \backslash F^*)} \leq \frac{1+b}{(1-\varepsilon)(\alpha + (\alpha-1)b)} = \frac{1+b}{(1-\varepsilon)((2-4\varepsilon) + (1-4\varepsilon)b)}$$

$$\leq \frac{1}{(1-\varepsilon)(1-4\varepsilon)} \leq 1 + 8\varepsilon.$$

For $\alpha = \frac{3}{2} - 2\varepsilon$ we have:

$$\frac{w(C^N \backslash F^N)}{w(C^* \backslash F^*)} \leq \frac{1+b}{(1-\varepsilon)((\frac{3}{2}-2\varepsilon) + (\frac{1}{2}-2\varepsilon)b)} \leq \frac{1}{(1-\varepsilon)(\frac{1}{2}-2\varepsilon)} \leq 2 + 13\varepsilon.$$

3 An FPTAS for General Connectivity Interdiction

We explain how to obtain an FPTAS with arbitrary edge costs, proving Theorem 1.

Theorem 6. *Let (C^*, F^*) be an optimal b-free min-cut of the instance $(G = (V, E), w, c, b)$. There is a weight function $\tilde{w}^* : E \to \mathbb{R}_{\geq 0}$ such that C^* is a (2.01)-approximate min-cut w.r.t. $\tilde{w}^*$. Furthermore, there is an efficient algorithm that outputs a collection $\mathcal{W}$ of weight functions such that (1) $\mathcal{W}$ contains such a function $\tilde{w}^*$ and (2) $|\mathcal{W}| = O(\log(m \cdot w_{\max} \cdot b))$.*

Theorem 6 follows from Theorem 4(i) by guessing the appropriate value of τ in the range of $(1 + \varepsilon)^i$ for some fixed ε. For each guess τ, we define the weight function as in Definition 3 according to τ. We defer the proof to the end of the section.

Proposition 1. *Given a cut C, suppose that $F \subseteq C$ is the optimal feasible edge-set, then we can compute, in $O(m^2/\delta)$ time for $\delta > 0$, a feasible edge-set F' so that $c(F') \leq b$ and $w(C \backslash F') \leq (1 + \delta) w(C \backslash F)$.*

Proposition 1 follows from a simple reduction to knapsack minimization problem for which an FPTAS is known [GL80].

Furthermore, in [Isl09] a pseudo-polynomial time algorithm running in time $O(n^2 W)$ for the knapsack minimization is shown (where $W = \max w_i$). As we will see in remark 2 this allows us to get a pseudo-polynomial time algorithm for the general case of connectivity interdiction. We will prove Proposition 1 at the end of the section.

We are now ready to state the FPTAS. We try all weight functions from Theorem 6. For each weight function, we enumerate all 2.1-approximate cuts w.r.t. $\tilde{w}_\tau$. For each cut C, we compute $(1 + \delta)$-approximate feasible edge-set $F \subseteq C$ using Proposition 1. Finally, we output the best pair of a cut and its free-edge set found so far. The algorithm is summarized in Algorithm 1.

Algorithm 1: FPTAS for general connectivity interdiction

Input: b-free min-cut instance (G, c, w, b) and accuracy parameter $\delta > 0$.
Output: (C', F'), where C' is a cut in G and F' is its free edge-set $F' \subseteq C'$.
foreach *$\tilde{w}_\tau \in \mathcal{W}$ in the set of weight functions in Theorem 6* **do**
1. Enumerate $C_1, \ldots, C_\ell$, all 2.1-approximate cuts with respect to $\tilde{w}_\tau$.
2. For all $i \leq \ell$, Compute F_i an $(1 + \delta)$-approximate feasible edge-set of C_i as described in Proposition 1.

Return the cut and its free edge-set (C', F'), such that $w(C' \setminus F') \leq w(C \setminus F)$ for all pairs (C, F) computed in the previous step.

Running Time. We use Nagamochi et. al [NNI97] cut enumeration algorithm along with the fact that the number of α-approximate min-cuts is $O(n^{\lfloor 2\alpha \rfloor}) = O(n^4)$ [Kar00]. We can compute $(1+\delta)$-approximate feasible edge-set in $O(\frac{1}{\delta} \cdot m^2)$ time using Proposition 1. By Theorem 6, we repeat for $|\mathcal{W}| = O(\log(n \cdot w_{\max} \cdot b))$ iterations. Therefore, the algorithm runs in $O(\frac{m^2 n^4}{\delta} \log(n w_{max} b))$ time.

Correctness. By Theorem 6, there is $\tilde{w}^* \in \mathcal{W}$ such that C^* is an $(2+2\varepsilon)$-min-cut w.r.t. $\tilde{w}^*$ for $\varepsilon < 1/100$. Since we enumerate all 2.1-approximate min-cuts in w^*, we are bound to encounter C^* at some point in the enumeration. By Proposition 1, we can compute a set $F \subseteq C^*$ so that $w(C^* \backslash F) \leq (1+\delta)w(C^* \backslash F^*) \leq (1+\delta)\mathsf{OPT}^*$. As a result, the solution (C', F') returned by the algorithm must be an $(1+\delta)$-approximate solution to the b-free min-cut problem.

Remark 2. Note that to obtain a polynomial time algorithm for the general connectivity interdiction problem when the value $W = \max w_i$ is polynomial in the input size, we replace step 2 in algorithm 1 with the pseudo-polynomial time algorithm from [Isl09].

We now prove Theorem 6 and proposition 1.

Proof (Proof of Theorem 6). Observe that by the definition of OPT^N, it follows easily that[6]

$$\frac{1}{b+1} \leq \mathsf{OPT}^N \leq \lambda_0.$$

Trivially, $\lambda_0 \leq mw_{\max}$. Thus $\mathsf{OPT}^N \in [\frac{1}{b+1}, mw_{\max}]$. We define the collection of weight functions as follows. Let $\mathcal{W} = \{\}$ and let $\varepsilon \leq 1/100$. For all $j = 0, 1, \cdots$ so that $\frac{(1+\varepsilon)^j}{b+1} \leq mw_{\max}$, set $\tau = \frac{(1+\varepsilon)^j}{b+1}$, and define $\tilde{w}_\tau$ according to τ (as in Definition 3) and add it into $\mathcal{W}$. The number of weight functions is $|\mathcal{W}| = O(\log mw_{\max} b)$; furthermore, one of the τs must satisfy $\mathsf{OPT}^N(1-\varepsilon) \leq \tau \leq \mathsf{OPT}^N$ for $\varepsilon \leq 1/100$. Therefore, Theorem 4(i) applies to that weight function.

Proof (Proof of Proposition 1). Finding the optimal set F^* reduces to the following knapsack minimization problem.

> (Knapsack Minimization) Find a subset $A^* \subseteq C^*$, which respects $c(A^*) \geq (\sum_{e \in C^*} c(e)) - b$, so that $w(A^*)$ is minimized.

$C^* \backslash F^*$ is exactly the optimal solution A^* in the above knapsack problem. It is known [GL80] that, in $O(m^2/\delta)$ time for $\delta > 0$, one can find a solution A so that $c(A) \geq (\sum_{e \in E} c(e)) - b$ and $w(A) \geq (1+\delta)w(A^*)$. Setting F to be $C^* \backslash A$ gives the desired proof.

4 Fast Algorithms for Unit Edge Costs

We prove Theorem 2 in this section. We assume that the given instance (G, c, w, b) has unit edge costs $c : E \to 1$. Notice that in this setting, once a cut C is chosen, the best feasible set of edges $F \subseteq C$ to be removed is simply the b heaviest edges according to w. We will call these edges the *free set* (of C).

[6] We can assume that there is no cut C such that $w(C) \leq b$. Therefore, all of the λ_is have value at least 1.

4.1 The Schematic Algorithm

We now describe the algorithm. The inputs consist of an b-free mincut instance (G, c, w, b) and two positive parameters $\varepsilon < 1/100$ and $\alpha \in \{\alpha_1, \alpha_2, \alpha_3\}$ where $\alpha_1 := 2 + 2\epsilon, \alpha_2 := 2 - 4\epsilon, \alpha_3 := 1.5 - 2\epsilon$. The parameter α controls the trade-off between the approximation ratio and running time. First, we compute an estimate τ such that $\mathsf{OPT}^N(1-\varepsilon) \leq \tau \leq \mathsf{OPT}^N$. Then, we define the weight function $\tilde{w}_\tau$ according to Definition 3. We enumerate all α-approximate cuts, and for each cut C we compute $w(C \setminus F)$ where F is the free set of C. Finally, we return the cut C' and its free set F' with the smallest value of $w(C' \setminus F')$ so far. We summarize the algorithm in Algorithm 2.

Algorithm 2: The algorithm for the unit edge costs

Input: A b-free mincut instance (G, c, w, b) and two positive parameters $\varepsilon < 1/100$ and $\alpha \in \{\alpha_1, \alpha_2, \alpha_3\}$ where $\alpha_1 := 2 + 2\epsilon, \alpha_2 := 2 - 4\epsilon, \alpha_3 := 1.5 - 2\epsilon$.

Output: (C', F'), where C' is a cut in G and F' is its free edge-set $F' \subset C'$.

1. Let τ be an estimate of OPT^N such that $\mathsf{OPT}^N(1-\varepsilon) \leq \tau \leq \mathsf{OPT}^N$.
2. Let $\mathcal{C}_{\tilde{w}_\tau}$ be the set of α-approximate min-cuts in the instance $(G, \tilde{w}_\tau)$ where $\tilde{w}_\tau$ is defined in Definition 3, which depends on the value τ.
3. For each cut $C \in \mathcal{C}_{\tilde{w}_\tau}$, compute $w(C \setminus F)$ where F is the free set of C.
4. Return the cut and its free set (C', F') such that $w(C' \setminus F') \leq w(C \setminus F)$ for all pairs (C, F) computed in the previous step.

Analysis. We next describe the analysis of Algorithm 2.

Theorem 7. *Let (C', F') be the solution returned by Algorithm 2 on the b-free mincut instance (G, c, w, b) with parameters $\varepsilon < 1/100$ and $\alpha \in \{\alpha_1, \alpha_2, \alpha_3\}$.*

1. *If $\alpha = \alpha_1$, and $\varepsilon = 0.001$, then (C', F') is optimal and it can be implemented to run in $\tilde{O}(m + n^4 b)$ time.*
2. *If $\alpha = \alpha_2$, then (C', F') is $(1 + O(\varepsilon))$-approximation and it can be implemented to run in $\tilde{O}(\frac{1}{\varepsilon} \cdot m + n^3 b)$ time.*
3. *If $\alpha = \alpha_3$, then (C', F') is $(2 + O(\varepsilon))$-approximation and it can be implemented to run in $\tilde{O}(\frac{1}{\varepsilon} \cdot m + n^2 b)$ time.*

Every implementation is randomized and succeeds with high probability.

Theorem 2 follows immediately from Theorem 7. For implementation and running time analysis, we will discuss the necessary tools in Section 4.2. Here we just focus on proving the correctness.

Correctness. We now prove the correctness of the algorithm in Theorem 7. Let (C^*, F^*) be an optimal solution to b-free min-cut problem, and (C^N, F^N) be an optimal solution to the normalized min-cut problem. In each case of $\alpha \in \{\alpha_1, \alpha_2, \alpha_3\}$, we apply Theorem 4 using $\tilde{w}_\tau$ as defined in Definition 3, where $\mathsf{OPT}^N(1-\epsilon) \leq \tau \leq \mathsf{OPT}^N$.

1. If $\alpha = \alpha_1$ and $\epsilon = 0.001$, then $\mathcal{C}_{\tilde{w}_\tau}$ contains C^*, and thus F^* was computed in step 3 of the algorithm. Therefore, at the end we have $w(C' \setminus F') \leq w(C^* \setminus F^*)$.
2. If $\alpha \in \{\alpha_2, \alpha_3\}$, then we consider two different cases depending on $\tilde{w}_\tau(C^*)$. If $\tilde{w}_\tau(C^*) \leq \alpha \tilde{w}_\tau(C^{\min})$, where $C^{\min}$ is global min-cut w.r.t. $\tilde{w}_\tau$, then $\mathcal{C}_{\tilde{w}_\tau}$ contains C^*, and then we are done by the same argument in as the previous case. Otherwise, by Theorem 4, $\tilde{w}_\tau(C^N)$ is α-approximate cut in $\tilde{w}_\tau$. Thus, $C^N \in \mathcal{C}_{\tilde{w}_\tau}$. In this case, F^N was also computed in step 3 of the algorithm. Therefore, if $\alpha = \alpha_2$, then $w(C' \setminus F') \leq w(C^N \setminus F^N) \leq (1 + O(\varepsilon))w(C^* \setminus F^*)$, where the last inequality follows from Theorem 4(ii). If $\alpha = \alpha_3$, then $w(C' \setminus F') \leq w(C^N \setminus F^N) \leq (2 + O(\varepsilon))w(C^* \setminus F^*)$, where the last inequality follows from Theorem 4(iii).

4.2 Fast Implementation

For the fast implementation of algorithm 2, we need an efficient enumeration of α-approximate cuts and the efficient evaluation of weights of cuts when their b heaviest edges are removed. We achieve the enumeration via the adaptation of the tree packing theorem by Karger [Kar00] for 3 and 4-respecting cuts (see Definition 4 below).

For the evaluation of cuts with respect to the original weight w when their b-heaviest edges are removed, we generalize a cut oracle data structure from [CQ17a] that *efficiently* computes the weight of 1 and 2-respecting cuts, to also handle 3 and 4-respecting cuts and to handle the computation of their b heaviest edges via the addition of a max-heap data structure to cut oracle.

Definition 4 (r-respecting cut). *Let $G = (V, E)$ be a graph and let T be a spanning tree of G. We say that a cut C in G r-respects T, if $|C \cap E(T)| = r$.*

We remark that given a spanning tree T, each set of r tree edges in T *induces* a unique cut in G.

Karger [Kar00] (see also the discussion [CQ17a]) gave a randomized algorithm to compute a collection of $O(\log n)$ spanning trees such that with high probability every α-approximate min-cut r-respects some spanning tree in the collection.

Theorem 8 (Tree Packing Theorem [Kar00] (Restated)).
Let $G = (V, E, w)$ be an undirected weighted graph. There is an algorithm that takes G and a parameter α (where $\alpha \in \{\alpha_1, \alpha_2, \alpha_3\}$, and $\varepsilon < 1/100$) as input and outputs a collection $\mathcal{T}$ of $O(\log n)$ spanning trees (the constant factor in $O(\cdot)$ depends on α) satisfying the following with high probability:

- *If $\alpha = \alpha_1$, then every α_1-approx. min-cut 4-respects a spanning tree in $\mathcal{T}$,*

- *If $\alpha = \alpha_2$, then every α_2-approx. min-cut 3-respects a spanning tree in $\mathcal{T}$,*
- *If $\alpha = \alpha_3$, then every α_3-approx. min-cut 2-respects a spanning tree in $\mathcal{T}$.*

Our approximate cut enumeration task now reduces to the enumeration of all $2, 3$ and 4-respecting cuts of $\mathcal{T}$, and hence the enumeration of all $O(n^2)$, $O(n^3)$ or $O(n^4)$ possible combinations of $2, 3$ or 4 edges from $\mathcal{T}$, respectively.

Let U be a set of edges drawn from tree T in the enumeration process, and C the corresponding cut induced by U in G. We denote U as $[[C]]_T$ to represent the encoding of C via the tree T. By Theorem 8, every α-approximate min-cut r-respects some tree in the tree packing $\mathcal{T}$ for some $r \leq 4$ with high probability. Then, every α-approximate min-cut is listed in the enumeration with high probability.

However, during the enumeration, we need to compute the value $w(C \setminus F) = w(C) - w(F)$ where F is the set of b heaviest edges in C. Naively, this takes $O(m)$ time per iteration. To speed up this computation, we preprocess each tree in the tree packing so that for any cut $[[C]]_T$ represented by the tree, we can compute $w(C \setminus F)$ in $\tilde{O}(b)$ time. This data structure is formalized in the following:

Lemma 5 (Cut Oracle). *There is a data structure $\mathcal{O}$ that given a weighted graph $G = (V, E, w)$, and a spanning tree T of G, supports the following operations, where $[[C]]_T$ is the set of at most 4 tree edges of T that induces C.*

- *$\mathcal{O}$.*Weight*($[[C]]_T$): Return $w(C)$ in $\tilde{O}(1)$ time.*
- *$\mathcal{O}$.*kFreeWeight*($[[C]]_T, k$): Return $w(C \setminus F)$ where $F \subseteq C$ is the set of k heaviest edges in C in $\tilde{O}(k)$ time.*
- *$\mathcal{O}$.*Cut*($[[C]]_T$): returns the cut C in $O(m)$ time.*

The data structure can be constructed in $\tilde{O}(m)$ time.

See the appendix for the proof of lemma 5 and the details about the fast implementation and the analysis of running time.

Acknowledgements

This research was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 759557, ERC Starting grant CODY 101039914, ISF grant 3011005535, Academy of Finland grant 335715, the group CASINO/ENS chair on algorithmics and machine learning, and also by the grants ANR-19-CE48-0016 and ANR-18-CE40-0025-01.

References

[ASG13] Bernardetta Addis, Marco Di Summa, and Andrea Grosso. Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth. *Discret. Appl. Math.*, 161(16-17):2349–2360, 2013.

[Ass87] Nikitas Assimakopoulos. A network interdiction model for hospital infection control. *Computers in Biology and Medicine*, 17(6):413–422, 1987.

[BGV89] Michael O. Ball, Bruce L. Golden, and Rakesh V. Vohra. Finding the most vital arcs in a network. *Operations Research Letters*, 8(2):73–76, 1989.

[BSR20] Simon Busan, Luca E. Schäfer, and Stefan Ruzika. The two player shortest path network interdiction problem. *CoRR*, abs/2004.08338, 2020.

[BTV10] Cristina Bazgan, Sonia Toubaline, and Daniel Vanderpooten. Complexity of determining the most vital elements for the 1-median and 1-center location problems. In Weili Wu and Ovidiu Daescu, editors, *Combinatorial Optimization and Applications - 4th International Conference, COCOA 2010, Kailua-Kona, HI, USA, December 18-20, 2010, Proceedings, Part I*, volume 6508 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2010.

[CHN+22] Parinya Chalermsook, Chien-Chung Huang, Danupon Nanongkai, Thatchaphol Saranurak, Pattara Sukprasert, and Sorrachai Yingchareonthawornchai. Approximating k-edge-connected spanning subgraphs via a near-linear time LP solver. In *ICALP*, volume 229 of *LIPIcs*, pages 37:1–37:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[CQ17a] Chandra Chekuri and Kent Quanrud. Approximating the held-karp bound for metric TSP in nearly-linear time. In *FOCS*, pages 789–800. IEEE Computer Society, 2017.

[CQ17b] Chandra Chekuri and Kent Quanrud. Near-linear time approximation schemes for some implicit fractional packing problems. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 801–820. SIAM, 2017.

[CZ17] Stephen R. Chestnut and Rico Zenklusen. Hardness and approximation for network flow interdiction. *Networks*, 69(4):378–387, 2017.

[DXT+12] Thang N. Dinh, Ying Xuan, My T. Thai, Panos M. Pardalos, and Taieb Znati. On new approaches of assessing network vulnerability: Hardness and approximation. *IEEE/ACM Trans. Netw.*, 20(2):609–619, 2012.

[GL80] Georgii Gens and Evgenii Levner. Complexity of approximation algorithms for combinatorial problems: A survey. *SIGACT News*, 12(3):52–65, sep 1980.

[GMT71] P. M. Ghare, D. C. Montgomery, and W. C. Turner. Optimal interdiction policy for a flow network. *Naval Research Logistics Quarterly*, 18(1):37–45, 1971.

[Isl09] Mohammad Tauhidul Islam. *Approximation algorithms for minimum knapsack problem*. PhD thesis, Lethbridge, Alta.: University of Lethbridge, Dept. of Mathematics and Computer Science, 2009.

[Kar00] David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000.

[MM09] Timothy C. Matisziw and Alan T. Murray. Modeling s-t path availability to support disaster vulnerability assessment of network infrastructure. *Comput. Oper. Res.*, 36(1):16–26, 2009.

[MMG89] K. Malik, A.K. Mittal, and S.K. Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 8(4):223–227, 1989.

[MPS07] David P. Morton, Feng Pan, and Kevin J. Saeger. Models for nuclear smuggling interdiction. *IIE Transactions*, 39(1):3–14, 2007.

[NNI97] Hiroshi Nagamochi, Kazuhiro Nishimura, and Toshihide Ibaraki. Computing all small cuts in an undirected network. *SIAM J. Discret. Math.*, 10(3):469–481, 1997.

[NW61] C. St.J. A. Nash-Williams. Edge-Disjoint Spanning Trees of Finite Graphs. *Journal of the London Mathematical Society*, s1-36(1):445–450, 01 1961.

[SBNK95] Baruch Schieber, Amotz Bar-Noy, and Samir Khuller. *The complexity of finding most vital arcs and nodes.* University of Maryland at College Park, USA, 1995.

[SS19] J. Cole Smith and Yongjia Song. A survey of network interdiction models and algorithms. *European Journal of Operational Research*, 283(3):797–811, 2019.

[SWB04] J. Salmeron, K. Wood, and R. Baldick. Analysis of electric grid security under terrorist threat. *IEEE Transactions on Power Systems*, 19(2):905–912, 2004.

[TLK99] A. Tsay, W. Lovejoy, and David Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24:383–413, 02 1999.

[Woo93] R.Kevin Wood. Deterministic network interdiction. *Mathematical and Computer Modelling*, 17(2):1–18, 1993.

[Zen10] Rico Zenklusen. Matching interdiction. *Discret. Appl. Math.*, 158(15):1676–1690, 2010.

[Zen14] Rico Zenklusen. Connectivity interdiction. *Operations Research Letters*, 42(6):450–454, 2014.

[Zen15] Rico Zenklusen. An O(1)-approximation for minimum spanning tree interdiction. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 709–728, 2015.

5 Appendix

5.1 Proof of theorem 8 & Fast Implementation

5.1.1 Proof of theorem 8 We now show the proof of the tree packing theorem (theorem 8) with the parameters adapted to our purposes.

Let $\alpha_1 = 2 + 2\varepsilon, \alpha_2 = 2 - 4\varepsilon$ & $\alpha_3 = \frac{3}{2} - 2\varepsilon$.
We recall Theorem 8.

Theorem 8 (Tree Packing Theorem [Kar00] (Restated)).
Let $G = (V, E, w)$ be an undirected weighted graph. There is an algorithm that takes G and a parameter α (where $\alpha \in \{\alpha_1, \alpha_2, \alpha_3\}$, and $\varepsilon < 1/100$) as input and outputs a collection $\mathcal{T}$ of $O(\log n)$ spanning trees (the constant factor in $O(\cdot)$ depends on α) satisfying the following with high probability:

- *If $\alpha = \alpha_1$, then every α_1-approx. min-cut 4-respects a spanning tree in $\mathcal{T}$,*
- *If $\alpha = \alpha_2$, then every α_2-approx. min-cut 3-respects a spanning tree in $\mathcal{T}$,*
- *If $\alpha = \alpha_3$, then every α_3-approx. min-cut 2-respects a spanning tree in $\mathcal{T}$.*

Definition 5 (Tree packing). *Let $G = (V, E, w)$ be a weighted graph. A tree packing is a non-negatively weighted set of spanning trees such that for each edge $e \in E$, the total weight of the tree on e is at most $w(e)$.*

The weight of a tree packing is the sum of the weights of all the trees contained in it, and a *maximum tree packing* is a tree packing with maximum weight.

In fact, Karger's [Kar00] randomized algorithm finds a tree packing of $O(\log n)$ spanning trees, whose value is at least $(1-\varepsilon)$ times the value of the maximum tree packing and such that the minimum cut 2-respects 1/3 of them with high probability in $O(m+n\log^3 n)$ time, using a previous construction based on random sampling [TLK99].

In [CQ17b], they show an algorithm to find a deterministic near-linear-time $(1-\varepsilon)$-approximate packing, in $\tilde{O}(\frac{m}{\varepsilon^2})$ time.

An easy corollary of Nash-Williams' [NW61] work on the characterization of the value of a maximum tree packing implies that this value is at least half of the value of the min-cut. In turn, this implies that on average any tree selected at random (in proportion to its weight) from a maximum tree packing contains ≤ 2 edges from the min-cut and strictly less than 3 edges from the min-cut with a constant probability.

Lemma 6 (cf. Lemma 2.3 of [Kar00]). *Let λ denote the value of the minimum cut in G. Given any weighted tree packing of value $\beta\lambda$ and any cut of value $\alpha\lambda$, at least a $\frac{1}{r}\cdot((r+1)-\alpha/\beta)$ fraction of the trees (by weight) r-constrain the cut.*

Corollary 2. *Let $\alpha \in \{\alpha_1, \alpha_2, \alpha_3\}$ and $\varepsilon < 1/100$. In any $(1-\varepsilon)$-approximation to the maximum tree packing, at least a constant fraction (the constant depends on α) of the trees (by weight)*

- *4-constrain an α-approximate min-cut if $\alpha = \alpha_1$,*
- *3-constrain an α-approximate min-cut if $\alpha = \alpha_2$, and*
- *2-constrain an α-approximate min-cut if $\alpha = \alpha_3$.*

Proof. The results follow by plugging in lemma 6 the values $\beta = (1-\epsilon)/2$, and $\alpha \in \{\alpha_1, \alpha_2, \alpha_3\}$ for $r = 4, 3, \& 2$, respectively. The corresponding constant fractions we obtain are $\geq \frac{23}{100}$ for α_1, $\geq \frac{13}{1000}$ for α_2 and $\geq \frac{50}{10000}$ for α_3.

To prove Theorem 8, we construct a $(1-\varepsilon)$-approximate tree packing in $\tilde{O}(\frac{m}{\varepsilon^2})$ time using the algorithm by [CQ17b]. From the tree packing, we select $O(\log n)$ random trees according to the weights. Since the number of α-approximate min-cuts is $O(n^{\lfloor 2\alpha \rfloor}) = n^{O(1)}$, Corollary 2 and the union bounds imply Theorem 8 as desired.

5.1.2 Cut Enumeration. Now that we can change the task of enumerating all α-approximate cuts to that of enumerating over all $2, 3$ or 4-respecting cuts of a collection of spanning trees $\mathcal{T}$, we need to show how to enumerate such respecting cuts efficiently. First, we compute the collection of spanning trees $\mathcal{T}$ using the algorithm in Theorem 8 with parameter α. For each tree T:

- If $\alpha = \alpha_1$, then we enumerate all possible combinations of (e_1, e_2, e_3, e_4) tree edges where e_2, e_3, e_4 could possibly be empty. The number combinations is $O(n^4)$.

- If $\alpha = \alpha_2$, then we enumerate all possible combinations of (e_1, e_2, e_3) tree edges where e_2, e_3 could possibly be empty. The number of combinations is $O(n^3)$.
- If $\alpha = \alpha_3$, then we enumerate all possible combinations of (e_1, e_2) tree edges where e_2 could possibly be empty. The number of combinations is $O(n^2)$.

5.1.3 The Implementation. Let us now look more closely at the fast implementation of our Algorithm 2:

In Step 1 of Algorithm 2, the value OPT^N can be $(1+\epsilon)$-approximated using the algorithm by [CHN+22] in $\tilde{O}(\frac{1}{\epsilon} \cdot m)$ time.

It remains to describe the fast implementation of Steps 2 and 3 of Algorithm 2. Recall that at this step, we are given $\tilde{w}$, a weight function on the input graph $G = (V, E)$. This differs from the input weight function w. We are also given $\alpha \in \{\alpha_1, \alpha_2, \alpha_3\}$ and $\varepsilon < 1/100$ as parameters.

1. Call the algorithm in Theorem 8 using parameter α and weight function $\tilde{w}$ to compute the collection of spanning trees $\mathcal{T}$ in $\tilde{O}(m)$ time.
2. For each tree $T \in \mathcal{T}$,
   (a) Construct the cut oracle (Lemma 5) $\mathcal{O}_T$ given the tree T in $\tilde{O}(m)$ time.
   (b) For each $[[C]]_T$ in the enumeration (see the section 5.1.2 above), call $\mathcal{O}_T.\textsc{kFreeWeight}([[C]]_T, b)$ to obtain $w(C \setminus F)$ where $F \subset C$ is the b heaviest edges in C. If the value is the smallest so far, update the solution to $[[C]]_T$.
3. Let $[[C']]_{T'}$ be a solution with the smallest value of $\mathcal{O}_{T'}.\textsc{kFreeWeight}([[C']]_{T'}, b)$ among all trees in $\mathcal{T}$. We then extract (C', F') as follows. First, compute $C' = \mathcal{O}_{T'}.\textsc{Cut}([[C']]_{T'})$ and F', which is the b heaviest edges in C'.

Analysis. The correctness follows immediately from Theorem 8 that every α-approximate cut can be enumerated via tree packing and the correctness of the data structures in Lemma 5. We now analyze the running time. We run the algorithm by [CHN+22] to obtain an estimate τ in $\tilde{O}(\frac{1}{\varepsilon} \cdot m)$ time. Tree packing and data structure preprocessing can be done in $\tilde{O}(m)$ time. The number of iterations in the enumeration is $\tilde{O}(n^4 b), \tilde{O}(n^3 b), \tilde{O}(n^2 b)$ for $\alpha = \alpha_1, \alpha_2, \alpha_3$, respectively. The evaluation of the value $\tilde{w}(C \setminus F)$ from the implicit representation of the cut $[[C]]_T$ can be done in $\tilde{O}(b)$ time. In total, it takes $\tilde{O}(n^4 b), \tilde{O}(n^3 b), \tilde{O}(n^2 b)$ time for $\alpha = \alpha_1, \alpha_2, \alpha_3$, respectively.

5.2 Data Structures

This section is devoted to proving lemma 5, which is the result of an extension of the data structure in [CQ17a] for cuts that $1, 2, 3$ and 4 respect a given spanning tree and which additionally computes the set of k heaviest edges of such 1,2, 3 or 4 respecting cuts.

The proof of Lemma 5 is a straightforward extension of the construction in [CQ17a] using Euler tour tree and range tree data structures. In order to

implement the operation $\mathcal{O}.\textsc{kFreeWeight}$, we use priority queues on top of the data structures provided by [CQ17a] to extract the top b edges efficiently.

We start by giving some preliminaries and definitions, and we then prove in lemma 7 our extension to the data structure in [CQ17a] by first giving an overview of their construction and then explaining how to easily generalize it for 3 and 4 respecting cuts. At the end of the section, we explain how to use lemma 7 to prove lemma 5 by making use of an additional max-heap data structure.

Basic graph notations. For convenience, when we say a cut we mean a set of edges $E(A, V \setminus A)$ for some vertex set A such that $\emptyset \neq A \subsetneq V$. For any graph G, we denote $V(G)$ and $E(G)$ as the set of vertices and edges in G, respectively.

Let $G = (V, E)$ be a graph and let T be a spanning tree of G.

Recall definition 4 for r-respecting cuts.

Definition 4 (r-respecting cut). *Let $G = (V, E)$ be a graph and let T be a spanning tree of G. We say that a cut C in G r-respects T, if $|C \cap E(T)| = r$.*

Remark 3. As we mentioned earlier a given set r of tree edges *induces* a unique cut back in G as follows: take the graph obtained after the removal of r edges in T and contract each of the $r + 1$ connected components into a single node to obtain T^C. This graph T^C is a tree, hence can be 2-colored with colors **red** and **blue**. Each node in $V(G)$ will get the same color as the connected component in which it is contained. This partition of the nodes in $V(G)$ into **red** and **blue** defines the cut in G.

We say that the cut is *induced* by the set of r tree edges. See fig. 1 for examples of cuts induced by the removal of $1, 2$ and 3 edges of a spanning tree.

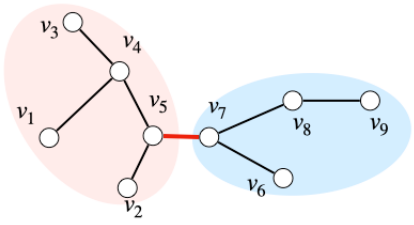
Tree packing and cut oracle. Let T be a spanning tree. If C is a cut that t-respects the tree T for some $t \leq 4$, we denote $[[C]]_T := E(T) \cap C$, a set of tree edges in T that induces the cut C. We extend the canonical cuts data structure presented in [CQ17a] to support up tp 4-respecting cuts.

Lemma 7. *There is a data structure $\mathcal{D}$ that takes as inputs a weighted graph $G = (V, E, w)$, and a spanning tree T of G, and supports the following operations.*

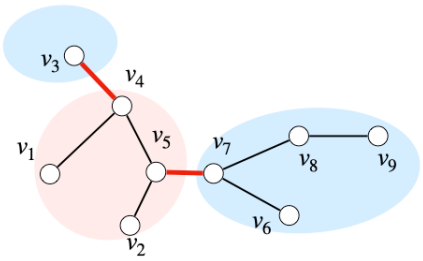
- $\mathcal{D}.\textsc{Preprocess}(G, T)$*: preprocess the inputs and return $\mathcal{C}$ a family of edge sets $D \subseteq E$ called canonical cuts such that $\sum_{D \in \mathcal{C}} |D| = \tilde{O}(m)$. This operation takes $\tilde{O}(m)$ time.*
- $\mathcal{D}.\textsc{Decompose}([[C]]_T)$*: Return $D_1, \ldots, D_\ell \in \mathcal{C}$ such that $C = \bigsqcup_{i \leq \ell} D_i$ and $\ell \leq \tilde{O}(1)$ where $\bigsqcup$ denotes the disjoint union operation. This operation takes $\tilde{O}(1)$ time.*

The operation $\mathcal{D}.\textsc{Decompose}$ is usable after the operation $\mathcal{D}.\textsc{Preprocessing}$ is completed.

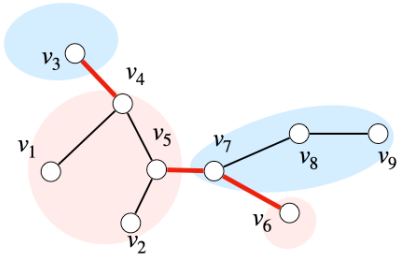
Proof. The construction of this data structure is as follows:



(a) Edge (v_5, v_7) defines a 1-respecting cut



(b) 2-respecting cut defined by edges (v_3, v_4) and (v_5, v_7)



(c) 3-respecting cut defined by edges (v_3, v_4), (v_5, v_7) and (v_7, v_6)

Fig. 1: A spanning tree T and examples of 1, 2 and 3-respecting cuts defined by the removal of 1, 2 and 3 edges of the tree. Each set of removed edges defines a unique separation of the nodes in T into two sets, **red** and **blue**.

Euler tours. Let T be a rooted spanning tree of G on $V(G) = n$ vertices. We begin by computing an Eulerian Tour of T starting from the root r. We delete from the Eulerian tour all occurrences of a node v which are not the first or the last occurrences in the tour, and we then replace the first occurrence of node v by the labelled node v^- and its last occurrence by the labelled node v^+. We denote the obtained sequence of labelled nodes by $\mathcal{S}$.

Range Trees. We build a range tree $\mathcal{R}$ from $\mathcal{S}$ as follows. Place the labeled vertices of $\mathcal{S}$ as leaves in a tree and build a natural balanced tree out of these leaves by creating internal nodes denoted by the set $\mathcal{I}$. Each such node $a \in \mathcal{I}$ induces a unique interval in sequence $\mathcal{S}$ defined by the sequence of leaves in the sub-tree rooted at node a in $\mathcal{R}$.

Canonical Intervals. We refer to an interval in $\mathcal{S}$ as *canonical*, whenever it corresponds to a sequence induced by the sub-tree rooted at a node in $\mathcal{R}$. Note that there exist $O(n)$ many such canonical intervals and we denote the set of all of them by D_I.

Claim. Take any interval I in $\mathcal{S}$ (not necessarily canonical), then I can be decomposed into the sequence of at most $O(\log n)$ canonical intervals from D_I.

Proof. Let $I = (w_1, ..., w_f)$ be an interval where the $w_i's$ are labelled nodes in $\mathcal{S}$. Take the path p_1 in $\mathcal{R}$ from w_1 to the root. Assume that w_1 is on the left sub-tree (equivalently, right sub-tree) of the root. Take as canonical intervals all the sub-trees of all right (left) children of nodes in p_1 which are not themselves in p_1. We do the same for w_f and take the disjoint union of the intervals found. Note that this ensures that the canonical intervals chosen are disjoint and at the same time that all vertices $w_i \in \mathcal{S}$ are contained in a canonical interval. Since p_1 contains $O(\log n)$ nodes in $\mathcal{R}$, this gives a decomposition into at most $O(\log n)$ many canonical intervals.

Note further, that one can similarly decompose into $O(\log n)$ canonical intervals the union and the complement of the union of any constant number of intervals in $\mathcal{S}$.

Canonical cuts. Let I_1, I_2 be two canonical intervals, then we denote by $\mathcal{C}$ the set of all edges (u, v) in $E(G)$ such that the labeled nodes u^- and v^- appear one in I_1 and the other in I_2. We call the set $\mathcal{C}$ a *canonical cut* induced by I_1 and I_2. Note that each labelled endpoint of an edge can be contained in at most $O(\log n)$ canonical intervals (since the height of $\mathcal{R}$ is $O(\log n)$) and hence each edge can be contained in $O(\log^2 n)$ canonical cuts.

We are now ready to show how to compute the operations $\mathcal{D}$.Preprocess(G, T) and $\mathcal{D}$.Decompose$([[C]]_T)$.

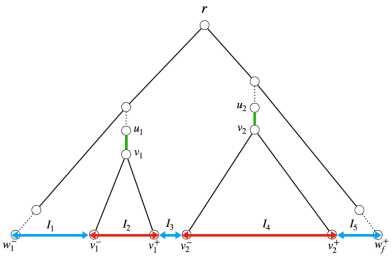
Preprocessing. For the operation $\mathcal{D}$.Preprocess(G, T), note that one can calculate the set of canonical cuts in $\tilde{O}(m)$ time as follows: First, calculate all the canonical intervals of T by computing an Eulerian tour of T, constructing its corresponding range tree $\mathcal{R}$ of size $O(n)$ in $\tilde{O}(m)$ time and by looking at the subtree of each node in $\mathcal{R}$. Second, for every such pair of canonical intervals, we can compute its set of induced non-empty canonical cuts in $O(m \log^2 n)$ time as follows: note that as we argued before each edge $e \in E(G)$ appears in at most $O(\log^2 n)$ canonical cuts, so there are at most $O(m \log^2 n)$ nonempty canonical cuts so we can construct them by adding each edge e to the $O(\log^2 n)$ canonical cuts which contain it.

This also shows that $\sum_{D \in \mathcal{C}} |D| = \tilde{O}(m)$. Note that our data structure stores both the set of canonical intervals and the set of canonical cuts, each identified by the pair of canonical intervals that induce it.

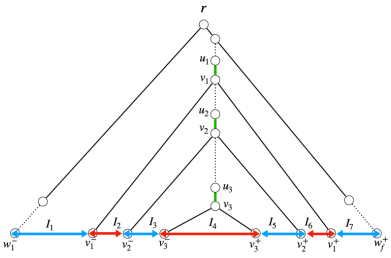
Decomposition. In [CQ17a], it is shown that all 1-or-2 respecting cuts of T can be decomposed into the disjoint union of $O(\log^2 n)$ canonical cuts. For our purposes, we need the generalization of this argument for r-respecting cuts, with $r \leq 4$.

Let us show first the idea of [CQ17a] for 1 and 2 respecting cuts and we will show that this easily generalizes to 3 and 4 respecting cuts.

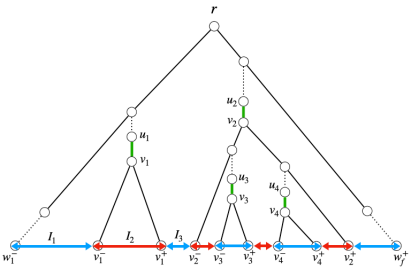
Let s be a sequence of elements $w_1, ..., w_f$ and let us define the square brackets notation as follows:



(a) 2-respecting cut where v_1 and v_2 are on edge-disjoint paths to the root.



(b) 3-respecting cut where v_1, v_2 and v_3 are on the same path to the root.



(c) 4-respecting cut where v_2 is in the paths from v_3 to the root and v_4 to the root and the path from v_1 to the root is edge disjoint of that from v_3 and v_4 to the root

Fig. 2: Examples for 2,3 and 4 -respecting cuts and their induced intervals. These intervals partition the leaf nodes into two sets, **red** and **blue**. An arrow head lays above a node when such node is included in the interval.

- $[w_i, w_j]$ with $i \leq j$ denotes the interval starting at and including element w_i and ending at and including element w_j.
- $]w_i, w_j[$ denotes the interval starting at the element appearing immediately after and not including w_i and ending at the element appearing immediately before and not including element w_j.
- $[w_i, w_j[$ denotes the interval starting at and including element w_i and ending at the element appearing immediately before and not including element w_j.
- $]w_i, w_j]$ denotes the interval starting at the element appearing immediately after and not including w_i and ending at and including element w_j.

1-respecting cuts Let $\mathcal{C}$ be a 1-respecting cut of T which is induced by the removal of an edge (u, v) in T. Assume u is contained in the unique path from v

to the root. Note that in $\mathcal{S}$, the removal of edge (u, v) also defines a partition into intervals, in particular, the interval defined by the sub-tree rooted at $v \in T$ is equivalent to the interval $I_2 = [v^-, ..., v^+]$, and its complement is the union of intervals $I_1 = [w_1^-, ..., v[$ and $I_3 =]v^+, ..., w_f^+]$. The edges in cut $\mathcal{C}$ hence correspond to edges between intervals I_1 and I_2, and between I_2 and I_3. Additionally, note that each such interval I_1, I_2, I_3 can be in turn decomposed into a set of $O(\log n)$ canonical intervals by claim 5.2. This implies that we can compute the edges in G of the 1-respecting cut $\mathcal{C}$ of T by computing all the canonical cuts between all pairs of canonical intervals (I_2, I_i) for $i = 1, 3$. Note that there are at most $O(\log^2 n)$ many pairs of canonical intervals and hence such 1-respecting cut can be represented by $O(\log^2 n)$ canonical cuts.

Note that to generalize this argument to r-respecting cuts, where r is a constant, the only change to the arguments above is the partition into intervals in $\mathcal{S}$ one would obtain from a given r-respecting cut. If we can show as above that each cut C which is r-respecting T (and which can be represented as the deletion of r edges of T) produces only a partition into constant number of intervals in $\mathcal{S}$, we can use the decomposition of each of the constant many intervals into $O(\log n)$ many canonical intervals, and hence there would be $O(\log^2 n)$ many canonical cuts representing the edges between each pair of intervals.

We will show the details for the case of $r = 2, 3$ and 4-respecting cuts.

2-respecting cuts. Let (u_1, v_1) and (u_2, v_2) be the two edges which are cut in the 2-respecting cut C and assume w.l.o.g. that u_1 is on the path from v_1 to the root and that u_2 is on the path from v_2 to the root. We have two cases: either one of the edges appears in the other's path to the root or they have (edge) disjoint paths to the root. Assume the latter and note that the removal of edge (u_1, v_1) in T defines the separation of the nodes in the sub-tree rooted at v_1 and hence corresponds to the interval $I_2 = [v_1^-, ..., v_1^+]$ in $\mathcal{S}$; similarly, the removal of edge (u_2, v_2) in T defines the separation of the nodes in the sub-tree rooted at v_2 and hence corresponds to the interval $I_4 = [v_2^-, ..., v_2^+]$ in $\mathcal{S}$. Define $I_1 = [w_1^-, ..., v_1[$, $I_3 =]v_1^+, ..., v_2 - [$ and $I_5 =]v_2^+, ..., w_f^+]$, the complement intervals to I_2 and I_4 in $\mathcal{S}$ (see fig. 2a). Finally note that since (u_1, v_1) and (u_2, v_2) are on edge disjoint paths to the root, the nodes on their sub-trees rooted at v_1 and v_2 in T will be at the same side of the cut. This implies that the edges C in G correspond to those edges between the 6 pairs of intervals (I_a, I_b), with $a = 2, 4$ and $b = 1, 3, 5$.

For the case when the paths from v_1 to the root and v_2 to the root are not edge disjoint, assume w.l.o.g. that v_1 is on v_2's path to the root. The removal of edge (u_1, v_1) corresponds to an interval $[v_1^-, ..., v_1^+]$ in $\mathcal{S}$, while the additional removal of edge (u_2, v_2) further divides this interval into intervals $I_2 = [v_1^-, ..., v_2^-[$, $I_3 = [v_2^-, ..., v_2^+]$ and $I_4 =]v_2^+, ..., v_1^+]$, where I_2 and I_4 are on different sides of the cut than I_3. Define $I_1 = [w_1^-, ..., v_1^-[$ and $I_5 = [v_1^+, ..., w_f^+]$ be the complement intervals in $\mathcal{S}$. Now, the edges in C will be given by the edges between the 6 pairs of intervals (I_a, I_b) with $a = 2, 4$ and $b = 1, 3, 5$.

3-respecting cuts. Let $e_1 = (u_1, v_1)$, $e_2 = (u_2, v_2)$ and $e_3 = (u_3, v_3)$ be the edges to be removed with u_i on the path from v_i to the root for $i = 1, 2, 3$. We have three cases:

- e_1, e_2 and e_3 are all in the same path from v_3 to the root (with (u_1, v_1) being the closest of them to the root). We can decompose into intervals $I_1 = [w_1^-, ..., v_1^-[$, $I_2 = [v_1^-, ..., v_2^-[$, $I_3 = [v_2^-, ..., v_3^-[$, $I_4 = [v_3^-, ..., v_3^+]$, $I_5 =]v_3^+, ..., v_2^+]$, $I_6 =]v_2^+, ..., v_1^+]$ and $I_7 =]v_1^+, ..., w_f^+]$ (see fig. 2b). The edges of the cut are given by the 12 pairs of intervals (I_a, I_b) with $a = 2, 4, 6$ and $b = 1, 3, 5, 7$.
- e_1, e_2 and e_3 are all on edge disjoint paths to the root. In this case, we have that the removal of e_1 defines the interval $I_2 = [v_1^-, ..., v_1^+]$, the removal of e_2 defines the interval $I_4 = [v_2^-, ..., v_2^+]$ and the removal of e_3 defines the interval $I_6 = [v_3^-, ..., v_3^+]$. The complements of these intervals in $\mathcal{S}$ are the intervals $I_1 = [w_1^-, ..., v_1^-[$, $I_3 = [v_1^+, ..., v_2^-[$, $I_5 = [v_2^+, ..., v_3^-[$ and $I_7 = [v_3^+, ..., w_f^+]$. The edges of the cut in G is given by the edges between the 12 pairs of intervals (I_a, I_b), with $a = 2, 4, 6$, $b = 1, 3, 5, 7$.
- e_1 and e_2 are in the same path from v_2 to the root and the path from v_3 to the root is edge disjoint of that from v_2 to the root. Furthermore, assume w.l.o.g. that v_3^- appears after v_2^- in $\mathcal{S}$. Then we have the intervals: $I_1 = [w_1^-, ..., v_1^-[$, $I_2 = [v_1^-, ..., v_2^-[$, $I_3 = [v_2^-, ..., v_2^+]$, $I_4 =]v_2^+, ..., v_1^+]$, $I_5 =]v_1^+, ..., v_3^-[$, $I_6 = [v_3^-, ..., v_3^+]$ and $I_7 =]v_3^+, ..., w_f^+]$. The edges of the cut are given by the 12 pairs of intervals (I_a, I_b) with $a = 2, 4, 6$ and $b = 1, 3, 5, 7$.
- e_1 is in the path from v_2 to the root and in the path from v_3 to the root, but v_2 is not on the path from v_3 to the root and neither v_3 is in the path from v_2 to the root. Furthermore, assume w.l.o.g. that v_3^- appears after v_2^- in $\mathcal{S}$. Then we have the intervals: $I_1 = [w_1^-, ..., v_1^-[$, $I_2 = [v_1^-, ..., v_2^-[$, $I_3 = [v_2^-, ..., v_2^+]$, $I_4 =]v_2^+, ..., v_3^-[$, $I_5 = [v_3^-, ..., v_3^+]$, $I_6 =]v_3^+, ..., v_1^+]$ and $I_7 =]v_1^+, ..., w_f^+]$. The edges of the cut are given by the 12 pairs of intervals (I_a, I_b) with $a = 2, 4, 6$ and $b = 1, 3, 5, 7$.

4-respecting cuts. Let $e_1 = (u_1, v_1)$, $e_2 = (u_2, v_2)$, $e_3 = (u_3, v_3)$ and $e_4 = (u_4, v_4)$ be the edges to be removed with u_i on the path from v_i to the root for $i = 1, 2, 3, 4$ and v_i^- appearing before v_{i+1}^- in $\mathcal{S}$ for $i = 1, 2, 3$. We have five cases:

- e_2 is in the paths from v_3 to the root and v_4 to the root, e_3 is not on the path from v_4 to the root, e_4 is not on the path from v_3 to the root and the path from v_1 to the root is edge disjoint of that from v_3 and v_4 to the root. Then we have the intervals: $I_1 = [w_1^-, ..., v_1^-[$, $I_2 = [v_1^-, ..., v_1^+]$, $I_3 =]v_1^+, ..., v_2^-[$, $I_4 = [v_2^-, ..., v_3^-[$, $I_5 = [v_3^-, ..., v_3^+]$, $I_6 =]v_3^+, ..., v_4^-[$, $I_7 = [v_4^-, ..., v_4^+]$, $I_8 =]v_4^+, ..., v_2^+]$ and $I_9 =]v_2^+, ..., w_f]^+$ (see fig. 2c). The edges of the cut are given by the 20 pairs of intervals (I_a, I_b) with $a = 2, 4, 6, 8$ and $b = 1, 3, 5, 7, 9$.
- e_1, e_2, e_3 and e_4 are all on edge disjoint paths to the root. In this case, we have that the removal of e_1 defines the interval $I_2 = [v_1^-, ..., v_1^+]$, the removal of e_2 defines the interval $I_4 = [v_2^-, ..., v_2^+]$, the removal of e_3 defines the interval $I_6 = [v_3^-, ..., v_3^+]$ and the removal of e_4 defines the interval $I_8 = [v_4^-, ..., v_4^+]$. The complements of these intervals in $\mathcal{S}$ are the intervals $I_1 = [w_1^-, ..., v_1^-[$,

$I_3 = [v_1^+, ..., v_2^-[$, $I_5 = [v_2^+, ..., v_3^-[$, $I_7 = [v_3^+, ..., v_4^-[$, and $I_9 = [v_4^+, ..., w_f^+]$. The edges of the cut in G is given by the edges between the 20 pairs of intervals (I_a, I_b), with $a = 1, 2, 3, 4$ and $b = 5, 6, 7, 8, 9$.

- e_1, e_2, e_3 and e_4 are all in the same path from v_4 to the root. We can decompose into intervals $I_1 = [w_1^-, ..., v_1^-[$, $I_2 = [v_1^-, ..., v_2^-[$, $I_3 = [v_2^-, ..., v_3^-[$, $I_4 = [v_3^-, ..., v_4^-[$, $I_5 = [v_4^-, ..., v_4^+]$, $I_6 =]v_4^+, ..., v_3^-]$, $I_7 =]v_3^+, ..., v_2^-]$, $I_8 =]v_2^+, ..., v_1^+]$ and $I_9 =]v_1^+, ..., w_f^+]$. The edges of the cut are given by the 20 pairs of intervals (I_a, I_b) with $a = 2, 4, 6, 8$ and $b = 1, 3, 5, 7, 9$.
- e_1 and e_2 are in the same path from v_2 to the root and the paths from v_3 and v_4 to the root are edge disjoint of that from v_2 to the root and of the other's path to the root. Then we have the intervals: $I_1 = [w_1^-, ..., v_1^-[$, $I_2 = [v_1^-, ..., v_2^-[$, $I_3 = [v_2^-, ..., v_2^+]$, $I_4 =]v_2^+, ..., v_1^+]$, $I_5 =]v_1^+, ..., v_3^-[$, $I_6 = [v_3^-, ..., v_3^+]$, $I_7 =]v_3^+, ..., v_4^-[$, $I_8 = [v_4^-, ..., v_4^+]$ and $I_9 =]v_4^+, ..., w_f^+]$. The edges of the cut are given by the 20 pairs of intervals (I_a, I_b) with $a = 2, 4, 6, 8$ and $b = 1, 3, 5, 7, 9$.
- e_1 and e_2 are on the same path from v_2 to the root, e_3 and e_4 are on the same path from v_4 to the root, and the path from v_4 to the root is edge disjoint of that from v_2 to the root. Then we have the intervals: $I_1 = [w_1^-, ..., v_1^-[$, $I_2 = [v_1^-, ..., v_2^-[$, $I_3 = [v_2^-, ..., v_2^+]$, $I_4 =]v_2^+, ..., v_1^+]$, $I_5 =]v_1^+, ..., v_3^-[$, $I_6 = [v_3^-, ..., v_4^-[$, $I_7 = [v_4^-, ..., v_4^+]$ $I_8 =]v_4^+, ..., v_3^+]$, and $I_9 =]v_3^+, ..., w_f^+]$. The edges of the cut are given by the 20 pairs of intervals (I_a, I_b) with $a = 2, 4, 6, 8$ and $b = 1, 3, 5, 7, 9$.
- e_1, e_2 and e_3 are in the same path from v_3 to the root and the path from v_4 to the root is edge disjoint of that from v_3 to the root. Then we have the intervals: $I_1 = [w_1^-, ..., v_1^-[$, $I_2 = [v_1^-, ..., v_2^-[$, $I_3 = [v_2^-, ..., v_3^-[$, $I_4 = [v_3^-, ..., v_3^+]$, $I_5 =]v_3^+, ..., v_2^+]$, $I_6 =]v_2^+, ..., v_1^+]$, $I_7 =]v_1^+, ..., v_4^-[$, $I_8 = [v_4^-, ..., v_4^+]$ and $I_9 =]v_4^+, ..., w_f^+]$. The edges of the cut are given by the 20 pairs of intervals (I_a, I_b) with $a = 2, 4, 6, 8$ and $b = 1, 3, 5, 7, 9$.
- e_1 is in the paths from v_2 to the root, v_3 to the root and v_4 to the root, e_2 is in the paths from v_3 to the root and v_4 to the root and e_3 is not on the path from v_4 to the root, neither e_4 is in the path from v_3 to the root. Then we have the intervals: $I_1 = [w_1^-, ..., v_1^-[$, $I_2 = [v_1^-, ..., v_2^-[$, $I_3 = [v_2^-, ..., v_3^-[$, $I_4 = [v_3^-, ..., v_3^+]$, $I_5 = [v_3^+, ..., v_4^-[$, $I_6 = [v_4^-, ..., v_4^+]$, $I_7 =]v_4^+, ..., v_2^+]$, $I_8 =]v_2^+, ..., v_1^+]$ and $I_9 =]v_1^+, ..., w_f]^+$. The edges of the cut are given by the 20 pairs of intervals (I_a, I_b) with $a = 2, 4, 6, 8$ and $b = 1, 3, 5, 7, 9$.

For the operation $\mathcal{D}$.DECOMPOSE($[[C]]_T$), given a $1, 2, 3$ or 4-respecting cut C as a set of edges to be removed we can use the above and decompose such cut C into constant many intervals in $\mathcal{S}$ and assigning each of them as described above, into their corresponding side of the cut a or b; then, we use claim 5.2 and decompose each such interval I into $O(\log n)$ canonical intervals in $\tilde{O}(1)$ time by traversing R from the endpoints of I to the root. Finally, for each pair of such canonical intervals I_1 and I_2, we can query its induced canonical cut from the preprocessing step in constant time. This gives us a decomposition of cut C into $O(\log^2 n)$ canonical cuts in $\tilde{O}(1)$ time.

Let us now show how to use Lemma 7 to construct the cut oracle data structure of lemma 5.

Proof (Proof of lemma 5). We will use the data structure from lemma 7 and will additionally store the weights of edges in each canonical cut so that retrieving the b edges of maximal weight in each of them can be done efficiently.

Let $\mathcal{CC} = (I_1, I_2)$ be the set of edges in a canonical cut induced by the pair of canonical intervals I_1 and I_2. For each such canonical cut, we will create a max-heap $\mathcal{H}_{(I_1,I_2)}$ with keys equal to the weights of the edges in $\mathcal{CC}$, that is, we construct $\mathcal{H}_{(I_1,I_2)}$ by inserting $w(e)$ into the heap every time a new edge e is added into the canonical cut CC. Notice that the construction of such max-heap can be implemented in linear time.

Also note that we can easily compute the total weight of each canonical cut while preprocessing each canonical cut and we can store its value together with its corresponding canonical cut. Then, given a cut $[[C]]_T$, we can compute $\mathcal{O}.\textsc{Weight}([[C]]_T)$ by computing the sum of the weights of all canonical cuts in which cut $[[C]]_T$ decomposes by the function $\mathcal{D}.\textsc{Decompose}([[C]]_T)$ of lemma 7.

For $\mathcal{O}.\textsc{kFreeWeight}([[C]]_T, k)$, we do as follows: for each of the canonical cuts that $[[C]]_T$ decomposes into, we look at its max-heap and query its maximum (in $O(1)$ time). We then select the maximum value from all the max queries to each canonical cut's max-heap and pop that value from the max-heap which contained it and store it in a variable k-sum. This ensures that the maximum weight in cut $[[C]]_T$ is removed. This whole process takes $\tilde{O}(1)$ time. We can iterate this process by popping the current maximum element among all the max-heaps and adding it to the current value of the variable k-sum. Note that after k iterations, k-sum contains the sum of the weights of the k heaviest edges in cut $[[C]]_T$, so that $\mathcal{O}.\textsc{kFreeWeight}([[C]]_T, k)$ is computed by $\mathcal{O}.\textsc{Weight}([[C]]_T) - (k\text{-sum})$. After k-sum is computed, we can re-insert back the elements which we popped out during the max queries (note that there will be k re-insertions).

Finally, $\mathcal{O}.\textsc{Cut}([[C]]_T)$ returns the cut C by finding the connected components in T (see remark 3) after the removal of the r edges in C respecting it. This can be done via DFS traversal in $O(m)$ time.