FPT and FPT-Approximation Algorithms for Unsplittable Flow on Trees

Tomás Martínez-Muñoz
Mathematical Engineering Department, University of Chile, Santiago, Chile

Andreas Wiese
Department of Industrial Engineering, University of Chile, Santiago, Chile

Abstract

We study the unsplittable flow on trees (UFT) problem in which we are given a tree with capacities on its edges and a set of tasks, where each task is described by a path and a demand. Our goal is to select a subset of the given tasks of maximum size such that the demands of the selected tasks respect the edge capacities. The problem models throughput maximization in tree networks. The best known approximation ratio for (unweighted) UFT is $O(\log n)$. We study the problem under the angle of FPT and FPT-approximation algorithms. We prove that

- UFT is FPT if the parameters are the cardinality k of the desired solution and the number of different task demands in the input,
- UFT is FPT under $(1+\delta)$-resource augmentation of the edge capacities for parameters k and $1/\delta$, and
- UFT admits an FPT-5-approximation algorithm for parameter k.

One key to our results is to compute structured hitting sets of the input edges which partition the given tree into $O(k)$ clean components. This allows us to guess important properties of the optimal solution. Also, in some settings we can compute *core sets* of subsets of tasks out of which at least one task i is contained in the optimal solution. These sets have bounded size, and hence we can guess this task i easily.

A consequence of our results is that the integral multicommodity flow problem on trees is FPT if the parameter is the desired amount of sent flow. Also, even under $(1+\delta)$-resource augmentation UFT is APX-hard, and hence our FPT-approximation algorithm for this setting breaks this boundary.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases FPT algorithms, FPT-approximation algorithms, packing problems, unsplittable flow, trees

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.67

Funding *Tomás Martínez-Muñoz*: Partially supported by CMM ANID PIA AFB170001.
Andreas Wiese: Partially supported by FONDECYT Regular grant 1200173.

1 Introduction

The unsplittable flow on trees (UFT) problem is a natural problem which models throughput maximization in tree networks. We are given a tree $G=(V,E)$ where each edge e has a capacity $u(e)\in\mathbb{N}$. Also, we are given a set of tasks $\mathcal{T}$ where each task $i\in\mathcal{T}$ is described by a path $P(i)\subseteq E$ and a demand $d(i)\in\mathbb{N}$. Our goal is to select a set of tasks $\mathcal{T}'\subseteq\mathcal{T}$ of maximum cardinality whose combined demands respect the edge capacities, i.e., $\sum_{i\in\mathcal{T}':e\in P(i)} d(i)\leq u(e)$ for each edge e. Hence, one application is that each task models a possible transmission between two nodes in a (tree) network in which the edges have bounded capacities. Note that UFT generalizes the integral multi-commodity flow on trees problem[1]

[1] In the integral multi-commodity flow on trees problem we are given a tree G with edge capacities u, and additionally pairs $s_i,t_i\in V$ of source and sink vertices. The goal is to select an integral amount of

which is known to be APX-hard [19]. Also, it generalizes the well-studied unsplittable flow on path (UFP) problem which has several applications, e.g., in caching [14], scheduling [4], and bandwidth allocation [12]. For UFT there is a $O(\log n)$-approximation algorithm [10] (and $O(\log^2 n)$-approximation algorithms for the weighted case [10, 18] and for submodular objective functions [1]). In particular, it is open to construct a $O(1)$-approximation algorithm for UFT.

In order to obtain better approximation ratios for combinatorial optimization problems, one can study them under the angle of FPT-approximation algorithms. In such algorithms, one defines some quantity to be a fixed parameter k which we define to be the size of the desired solution for our case of UFT, and then ensures that the running time is of the form $f(k)n^{O(1)}$ for some computable function f. Furthermore, for UFT we define that an FPT-α-approximation algorithm is an algorithm that either computes a solution of size at least k/α or asserts that there is no solution of size k. In the last years there several such results have been found for different problems. For example, for k-Median and k-Means there are tight FPT-approximation algorithms known with approximation ratios of $1+2/e+\epsilon$ and $1+8/e+\epsilon$, respectively [15], while the best known polynomial time algorithms for the problems have ratios of 2.611 [8] and 6.357 [3], respectively. For Facility Location there is an FPT-approximation algorithm with a ratio of essentially $1.463+\epsilon$ [15] which matches a known lower bound for (pure) approximation algorithms [21], while the best known upper bound is 1.488 [23]. For the capacitated versions of k-Median and k-Means there are FPT-$(3+\epsilon)$- and FPT-$(9+\epsilon)$-approximation algorithms known, respectively [16] while the best known polynomial time approximation ratio is only $O(\log k)$ based on an algorithm in [9] (see also [2]). For UFP there is an FPT-$(1+\epsilon)$-approximation algorithm [25] while the best known polynomial time approximation ratio is only $5/3+\epsilon$ [20]. For some W[1]-hard problems, there are even FPT-approximation algorithms known whose approximation ratios beat the best possible ratios of pure approximation algorithms. For example, for the Strongly Connected Steiner Subgraph problem, there is an FPT-2-approximation algorithm known when parametrized by the number of terminals [13], but there is a lower bound of $O(\log^{2-\epsilon} n)$ for (pure) approximation algorithms [22]. We refer to the surveys by Marx [24] and Feldmann et al. [17] for more results.

Given that for the mentioned problems FPT-approximation algorithms were found with better ratios than the best known (pure) approximation algorithms, this raises the question whether this is also possible for UFT. In this paper we answer this question in the affirmative and also show that certain special cases of UFT are even FPT.

1.1 Our contribution

Our first result is that UFT admits an *exact* FPT-algorithm if the parameters are k and the number $\bar{d}$ of different task demands in the input. Our first step is to compute a hitting set, i.e., a subset of the edges such that each input task uses at least one of them. Via a routine in [19] we construct a *structured* hitting set of size $O(k)$ or directly a solution of size k (in which case we are done). In particular, our hitting set partitions the tree into $O(k)$ clean components. Then we consider an edge e from the hitting set such that there is no edge from the hitting set underneath e. We consider all input tasks that use e but no other edge from the hitting set, let us denote them by $\mathcal{T}'$. Using some properties of our hitting set, we show that that in FPT time we can compute a *core set* of size $f(k,\bar{d})$ for $\mathcal{T}'$ (for some

flow to send between each pair (s_i,t_i), in order to maximize the total amount of sent flow.

function f) which is a set $\mathcal{T}'' \subseteq \mathcal{T}'$ such that we can assume w.l.o.g. that $\mathcal{T}''$ contains *all* tasks from $\mathcal{T}' \cap \mathrm{OPT}$. Therefore, in some sense, $\mathcal{T}''$ forms a kernel for $\mathcal{T}'$ (in the sense of being a smaller instance that we reduce our given instance to). This allows us to guess a task in $\mathcal{T}' \cap \mathrm{OPT}$ in time $f(k, \bar{d})$ or we guess that $\mathcal{T}' \cap \mathrm{OPT} = \emptyset$. In either case we make progress: either we find a task in OPT or we can delete one of the $O(k)$ edges of the hitting set. Therefore, our algorithm has only $O(k)$ iterations overall and in each of them there are only $f(k, \bar{d}) + 1$ options for our guesses.

▶ **Theorem 1.** *There is an FPT-algorithm for UFT for parameters k and $\bar{d}$ with a running time of $k^{O(k \cdot \bar{d}^k)} \cdot O(n^2)$.*

Note that the above mentioned integral multi-commodity flow problem (on trees) can be modeled by UFT instances in which $d(i) = 1$ for each task i and hence $\bar{d} = 1$. Therefore, we obtain an FPT-algorithm for this problem as a by-product. In particular, note that this problem is APX-hard [19] while our algorithm computes an *optimal* solution of size k.

▶ **Corollary 2.** *There is an FPT-algorithm for integral multi-commodity flow on trees where the parameter k denotes the amount of sent flow.*

Using the algorithm for UFT above we construct an FPT-algorithm for the general case of UFT under $(1+\delta)$-resource augmentation: our algorithm either computes a solution of size k that is feasible if we increase the capacity of each edge by a factor $1+\delta$, or we assert that there is no solution of size k for the original edge capacities. Key for this is to use the available recource augmentation to reduce the given instance to a set of smaller instances in which the input tasks have only $O(\log((k/\delta)^k))$ different demands. On each of these instances we invoke the algorithm from above and combine the obtained solutions. Due to the resource augmentation, we can ensure that this union forms a feasible solution.

▶ **Theorem 3.** *There is an FPT-algorithm for UFT under $(1+\delta)$-resource augmentation with a running time of $k^{(k/\delta)^{O(k)}} n^{O(1)}$.*

Note that our results implies an (exact) FPT-algorithm for UFP (i.e., on paths, rather than trees) under $(1+\delta)$-resource augmentation, which was not known before. Also, already UFP is W[1]-hard [25] for parameter k (and hence also UFT), which justifies that we use resource augmentation or the additional parameter $\bar{d}$. Furthermore, it establishes a distinction in comparison to (pure) approximation algorithms for UFT, since UFT is still APX-hard under resource augmentation, assuming that δ is sufficiently small.

▶ **Theorem 4** (implied by [19, Section 4]). *UFT is* APX*-hard under $(1+\delta)$-resource augmentation for any $\delta < 1/2$.*

Then we present an FPT-5-approximation algorithm for UFT (i.e., without resource augmentation) where the fixed-parameter is only k (and not also $\bar{d}$). Recall that the best known polynomial time approximation algorithm for (unweighted) UFT has an approximation ratio of only $O(\log n)$. Intuitively, let $i^* \in \mathrm{OPT}$ be the task of smallest demand in OPT. We guess which edges of our hitting set are used by i^*. Then we select the input task i of smallest demand that uses exactly these edges of the hitting set. By an exchange argument, we show that there are seven task in OPT such that we can replace these seven (unknown) tasks in OPT by our (known) task i and still obtain a feasible solution. For proving this, we again exploit some structural properties of our hitting set. Intutively, our task i pays for seven tasks in OPT. This yields a 7-approximation algorithm when we iterate this routine (and this argument). In order to improve the approximation ratio to 5, we guess additional properties of i^* such that we can compute a task i' which we can replace by only five tasks from OPT. This yields an FPT-5-approximation algorithm.

▶ **Theorem 5.** *There is an FPT-5-approximation algorithm for UFT with a running time of* $k^{O(k)}n^{O(1)}$.

1.2 Other related work

For the mentioned integral multi-commodity flow problem on trees there is a 2-approximation algorithm in the unweighted case [19] and a 4-approximation algorithm in the weighted case [11]. Under the no-bottleneck-assumption (NBA), i.e., assuming that $\max_{i\in\mathcal{T}} d(i) \le \min_{e\in E} u(e)$ there is a 48-approximation algorithm for UFT [11]. For uniform edge capacities and bounded node degrees, there is a 3.542-approximation algorithm for UFT [6]. For UFP there is no better approximation algorithm known under uniform edge capacities (and hence neither for the NBA) than the mentioned $(5/3+\epsilon)$-approximation for the general case [20]. However, UFP admits a QPTAS [5, 7], i.e., a $(1+\epsilon)$-approximation in time $n^{(\log n)^{O(1)}}$ for any constant $\epsilon > 0$, while for UFT a QPTAS is unlikely to exist since the problem is APX-hard [19].

2 Hitting sets and structuring the tree

In all our algorithms we will use hitting sets as the backbone of our computations. We define that a *hitting set* is a set of edges $E' \subseteq E$ such that each input task uses at least one edge in E', i.e., $P(i) \cap E' \neq \emptyset$ for each $i \in \mathcal{T}$. In this section, first we show that a result in [19] implies that in time $n^{O(1)}$ we can compute a hitting set of size at most $2k$ or directly find a solution of size k. Then, we use this as a base to compute a more structured hitting set E_{hs} of size $O(k)$. Afterwards, we use E_{hs} to partition the tree G and establish some structural properties that we can assume without loss of generality. Throughout the paper, we denote by n the number of bits in the input.

First, we invoke the mentioned result to compute an initial hitting set E'.

▶ **Lemma 6** (implicit in [19, Section 5]). *There is an algorithm with a running time of* $n^{O(1)}$ *that either outputs a set of tasks* $\mathcal{T}'$ *with* $|\mathcal{T}'| = k$ *such that* $P(i) \cap P(i') = \emptyset$ *for all* $i, i' \in \mathcal{T}'$ *with* $i \neq i'$, *or it outputs a hitting set* E' *with* $|E'| \le 2k$.

If the algorithm returns a set $\mathcal{T}'$ of k tasks with the mentioned properties then we can simply output $\mathcal{T}'$ since it forms a feasible solution. Assume now that the algorithm returns a hitting set E'. We want to add more edges to E' in order to obtain a more structured hitting set. For any two edges e, e' we define $\text{lca}(e, e')$ to be the vertex that is the least common ancestor of the (up to four) vertices incident to e and e'; also, we denote by $P_{e,e'} \subseteq E$ the (unique) path in G that contains e and e'. Intuitively, a hitting set is *good* if for any two edges $e, e' \in E_{hs}$ the edges of $P_{e,e'}$ incident to $\text{lca}(e, e')$ are also in the hitting set.

▶ **Definition 7.** *A hitting set* E_{hs} *is* good *if for any two edges* $e, e' \in E_{hs}$ *we have that* $\delta(\text{lca}(e, e')) \cap P_{e,e'} \subseteq E_{\text{hs}}$.

We construct a good hitting set E_{hs} based on E' as follows. We take all edges in E', and for any two edges $e, e' \in E'$ we add the edges in $\delta(\text{lca}(e, e')) \cap P_{e,e'}$, see Figure 1. By some standard tree arguments one can show that then $|E_{\text{hs}}| \le 6k$. Also, E_{hs} is a good hitting set.

▶ **Lemma 8.** *Given a hitting set* $\bar{E}'$, *in time* $n^{O(1)}$ *we can construct a good hitting set* $\bar{E}_{\text{hs}}$ *with* $|\bar{E}_{\text{hs}}| \le 3|\bar{E}'|$.

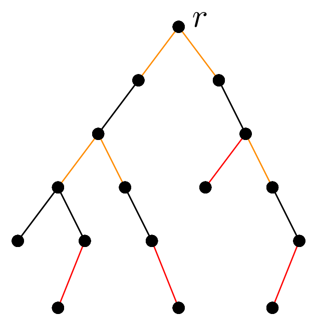

Figure 1 Assume that the red edges form a hitting set $\bar{E}'$. Then the union of the red and orange edges forms a good hitting set $\bar{E}_{\text{hs}}$. Given the red edges, the orange edges are selected according to the proof of Lemma 8.

2.1 Backbone and hanging trees

Let E_{hs} denote the good hitting set obtained by applying Lemma 8 to E'. We use E_{hs} in order to partition the edges of G into a backbone of *highway edges* (that will contain E_{hs}) and some subtrees which we will call *hanging trees*. First, we are interested in the edges that lie on some path that connects two edges in E_{hs} (see Figure 2). We say that an edge $e \in E$ is a *highway edge* if there are two edges $e', e'' \in E_{\text{hs}}$ such that $e \in P_{e',e''}$. Let $E^{\star}_{\text{hs}}$ denote the set of all highway edges and note that $E_{\text{hs}} \subseteq E^{\star}_{\text{hs}}$. Intuitively, $E^{\star}_{\text{hs}}$ forms a backbone of G. Note we cannot bound $|E^{\star}_{\text{hs}}|$ by a function of k only. On the other hand, we observe that $G[E^{\star}_{\text{hs}}]$ is a tree. If an edge $e \in E^{\star}_{\text{hs}}$ is incident to a leaf in $G[E^{\star}_{\text{hs}}]$ then we say that e is a *final edge*. Observe that then $e \in E_{\text{hs}}$ and thus there are at most $6k$ final edges. The final edges will play a special role later.

We say that the edges in $E \setminus E^{\star}_{\text{hs}}$ are *black* edges. Note that they form trees. For each connected component H of $G[E \setminus E^{\star}_{\text{hs}}]$ (i.e., the subgraph of G induced by $E \setminus E^{\star}_{\text{hs}}$), we say that H forms a *hanging tree* and we define its root s to be the vertex of H that is closest to the root of G. In the sequel, we will write H_s for a hanging tree with root s. For a hanging tree H_s we say that its *depth* is the maximum distance of a vertex in H_s to s.

By some easy transformations, we can ensure that the depth of each hanging tree H_s is bounded by a function in k and $\bar{d} := |\{d_i | i \in \mathcal{T}\}|$, i.e., $\bar{d}$ is the number of different task demands in the input. To this end, we note that if there are two edges $e, e' \in H_s$ appearing in this order on the path from s to a leaf of H_s and $u(e) \leq u(e')$, then we can contract e' since any input task T_i using e' also uses e (since $P(i)$ uses at least one edge of E_{hs} and hence $P(i)$ must pass through s). Also, we can assume w.l.o.g. that for each edge e its capacity $u(e)$ is the sum of the demands of at most k input tasks. With these ideas, we can prove the following lemma for which the intuition is that D is bounded by some parameter.

▶ **Lemma 9.** *In time $O(n^2)$ we can construct an equivalent instance $(G', \mathcal{T}')$ in which the depth of every hanging tree is at most $(\bar{d} + 1)^k + 1$.*

3 Parameterized algorithm for UFT

In this section we present an FPT-algorithm for UFT, assuming that our fixed parameters are k and $\bar{d}$ (recall that $\bar{d}$ is the number of different task demands in the input). In particular, this shows that UFT is FPT if all tasks have unit demands (which is APX-hard).

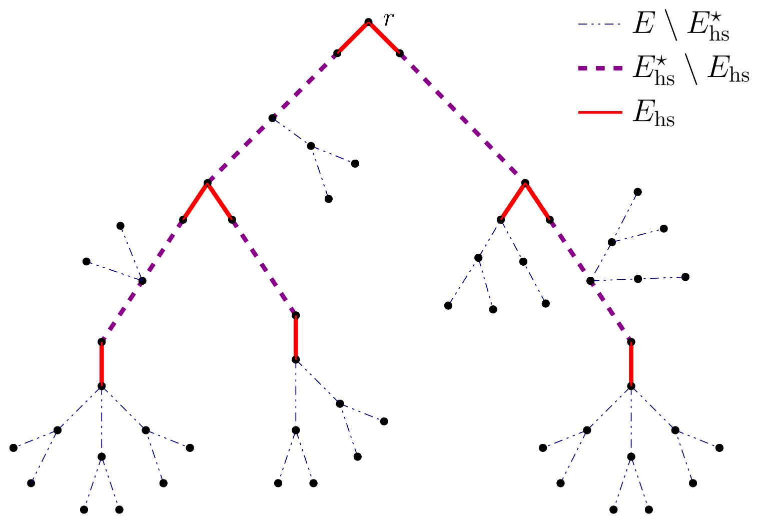
r
$E \setminus E^\star_{\text{hs}}$
$E^\star_{\text{hs}} \setminus E_{\text{hs}}$
E_{hs}

Figure 2 Partition of E, where E_{hs} are the edges in the good hitting set, $E^\star_{\text{hs}}$ are the edges in highway paths, and $E \setminus E^\star_{\text{hs}}$ are the edges in hanging trees.

Our strategy is to identify a subset $\mathcal{T}' \subseteq \mathcal{T}$ of tasks for which we can compute a core set which is intuitively a subset $K \subseteq \mathcal{T}'$ for which we can assume that it contains all tasks in $\mathcal{T}' \cap \text{OPT}$ for some optimal solution OPT. We will ensure that the size of our core set is bounded by a function in k and $\bar{d}$. Hence, in some sense it is a kernel for the set $\mathcal{T}'$.

▶ **Definition 10.** *Given a set $\mathcal{T}' \subseteq \mathcal{T}$, we say that a set $K \subseteq \mathcal{T}'$ is a* core set *for $\mathcal{T}'$ if for every feasible set of tasks $\mathcal{S} \subseteq \mathcal{T}$ there exists a feasible set of tasks $\mathcal{S}'$ such that $|\mathcal{S}'| = |\mathcal{S}|$, $\mathcal{S}' \setminus \mathcal{T}' = \mathcal{S} \setminus \mathcal{T}'$, and $\mathcal{S}' \cap \mathcal{T}' \subseteq K$.*

We will ensure that $|K| \leq f(k, \bar{d})$ for our computed core sets K, for some function f. Hence, having computed K, in time $O(f(k, \bar{d}))$ we can guess a task from $\mathcal{T}' \cap \text{OPT}$ or guess that $\mathcal{T}' \cap \text{OPT} = \emptyset$. In our main algorithm, we will compute $O(k)$ core sets for different sets $\mathcal{T}'$, guess which of their tasks are in OPT, and in this way eventually find a solution of size k in time $(f(k, \bar{d}))^{O(k)} n^{O(1)}$.

First, we prove some basic properties of core sets.

▶ **Lemma 11.** *Let $\mathcal{T}_1, \dots, \mathcal{T}_\ell$ be subsets of $\mathcal{T}$. If $K_1, \dots, K_\ell$ are core sets for $\mathcal{T}_1, \dots, \mathcal{T}_\ell$, respectively, then $\bigcup_{i=1}^{\ell} K_i$ is a core set for $\bigcup_{i=1}^{\ell} \mathcal{T}_i$.*

▶ **Lemma 12.** *Let $\mathcal{T}'$ and $\mathcal{T}''$ with $\mathcal{T}'' \subseteq \mathcal{T}' \subseteq \mathcal{T}$. If K is a core set for $\mathcal{T}''$ and $\mathcal{T}''$ is a core set for $\mathcal{T}'$ then K is also a core set for $\mathcal{T}'$.*

3.1 Constructing core sets

We first present an algorithm $\mathcal{A}^{\text{kr}}$ that computes a core set for any set of tasks $\mathcal{T}'$ for which intuitively there are two hanging trees H, H' such that all tasks in $\mathcal{T}'$ start in H and end in H'. Formally, the input of $\mathcal{A}^{\text{kr}}$ consists of a path $P = P_{s,t}$ for two nodes s and t that lie in two different hanging trees H, H', and of a value $d \in \mathbb{N}$ which is the demand of some input task. Let $\mathcal{T}'$ denote the set of all tasks $i \in \mathcal{T}$ such that $P(i)$ contains the path P, $d(i) = d$, and each edge of $P(i) \setminus E^\star_{\text{hs}}$ lies in H or H'. The algorithm $\mathcal{A}^{kr}$ computes a core set K for $\mathcal{T}'$. For each node v in some hanging tree H denote by p_v the maximum distance of v to a leaf of H. Recall that by Lemma 9 we can assume that for every $v \in V$ follows that $p_v \leq (\bar{d}+1)^k + 1$. For any two vertices v, v' denote by $P_{v,v'}$ the (unique) path from v to v', and for each vertex v denote by $\delta(v)$ the set of edges incident to v.

The algorithm $\mathcal{A}^{kr}(P,d)$ works as follows:

1. If $p_s = p_t = 0$ then return an arbitrary subset of $\mathcal{T}'$ of size $\min\{k, |\mathcal{T}'|\}$.
2. Otherwise, construct greedily a subset $A \subseteq \mathcal{T}'$ of tasks such that for every pair $i \neq j$ it holds that $P(i) \cap P(j) = P$: keep adding tasks to A until $|A| = 2k$ or if the property would not be fulfilled if we added an additional task $i \in \mathcal{T}' \setminus A$ to A.
3. If $|A| = 2k$ then return A.
4. If $|A| < 2k$ then let $\overline{E}$ be the set of edges in $(\delta(s) \cup \delta(t)) \setminus P$ that are used by tasks in A.
5. For every edge $e_j \in \overline{E}$ compute $K_j = \mathcal{A}^{\text{kr}}(P \cup \{e_j\}, d)$
6. Return $K = \bigcup_{e_j \in \overline{E}} K_j$.

We want to show that $\mathcal{A}^{\text{kr}}(P,d)$ returns a core set for $\mathcal{T}'$. Observe that $|\bar{E}| \leq O(k)$, the recursion depth is bounded by $O((\bar{d}+1)^k)$, and hence we output at most $k^{O((\bar{d}+1)^k)}$ tasks in total. Intuitively, if in a recursive call it holds that $|A| < 2k$ then $\bar{E}$ contains all edges in $(\delta(s) \cup \delta(t)) \setminus P$ that are used by tasks in $\mathcal{T}'$. Thus, for each task $i \in \mathcal{T}'$ there will be a recursive call $\mathcal{A}^{\text{kr}}(P \cup \{e_j\}, d)$ such that $P(i) \subseteq P \cup \{e_j\}$. On the other hand, if $|A| = 2k$ in some recursive call then A itself is a coreset for this call: if OPT contains a task $i \in \mathcal{T}' \setminus A$ then by the pigeon hole principle there must be another task $i' \in A \setminus \text{OPT}$ such that no task in OPT uses an edge in $P(i') \setminus P$ and hence we can swap i and i' in OPT, using that $d(i) = d(i')$. We formalize this in the following lemma.

▶ **Lemma 13.** *Let s,t be two nodes in two different hanging trees, let $d \in \mathbb{N}$, and let $\mathcal{T}' = \{i \in \mathcal{T} | P_{s,t} \subseteq P(i) \wedge d(i) = d\}$. Then $\mathcal{A}^{kr}(P_{s,t}, d)$ returns a core set for $\mathcal{T}'$ of size at most $(4k)^{p_s+p_t} \cdot k$ in time $(4k)^{p_s+p_t} \cdot k \cdot O(n)$.*

Proof. Let $\mathcal{T}_{s,t}$ be the set of tasks $i \in \mathcal{T}$ such that $P_{s,t} \subseteq P(i)$. We define the *instance depth* to be $p_s + p_t$, and we give a proof by induction on the instance depth. In the base case $p_s = p_t = 0$. Then in the first step of the algorithm two options can arise:

- $K = \mathcal{T}'$. In this case it follows directly that K is a core set for $\mathcal{T}'$.
- $K \neq \mathcal{T}'$. Let OPT be a feasible solution of size k. In this case it follows that K has size $2k$, and then we can replace all the tasks in $\text{OPT} \cap \mathcal{T}'$ by tasks of K.

Now consider the case that $p_s + p_t > 0$. Given an edge $e_j \in \overline{E}$, let s_j and t_j be the end nodes of the path $P \cup \{e_j\}$. It follows that $\mathcal{T}' \cap \mathcal{T}_{s_j,t_j}$ is the set of tasks in $\mathcal{T}'$ that uses e_j. Since $|P_{s_j,t_j}| = |P_{s,t}| + 1$, then $p_{s_j} + p_{t_j} = p_s + p_t - 1$, implying by the inductive hypothesis that K_j is a core set of $\mathcal{T}' \cap \mathcal{T}_{s_j,t_j}$. We define $\mathcal{T}'' = \bigcup_{j \in \{1,\ldots,|\overline{E}|\}} \mathcal{T}' \cap \mathcal{T}_{s_j,t_j}$. Lemma 11 implies that K is a core set for $\mathcal{T}''$. It remains to show that $\mathcal{T}''$ is a core set for $\mathcal{T}'$. If $|A| < 2k$ then each task in $\mathcal{T}'$ uses an edge from $\overline{E}$, which implies that $\mathcal{T}' = \mathcal{T}''$, so we conclude that $\mathcal{T}''$ is a core set for $\mathcal{T}'$. Since K is a core set for $\mathcal{T}''$ and $\mathcal{T}''$ is a core set for $\mathcal{T}'$, the Lemma 12 implies that K is a core set of $\mathcal{T}'$.

Assume now that $|A| = 2k$. Let OPT be a feasible solution of size k, such that among all solutions OPT′ of size k with $\text{OPT} \setminus \mathcal{T}' \subseteq \text{OPT}'$, the solution OPT is the solution that maximizes $\left|\text{OPT}' \cap A\right|$. Let us assume that there is a task $i \in (\text{OPT} \cap \mathcal{T}') \setminus A$. Each task in OPT shares an edge in $E \setminus P$ with at most two tasks from A. Therefore, there are at most $2(k-1)$ tasks $i' \in A$ such that $P(i') \setminus P$ contains an edge that is used by some task in OPT. Therefore, there exists a task $i'' \in A$ such that no edge in $P(i'') \setminus P$ is used by any tasks in OPT. Therefore, $\text{OPT} \cup \{i''\} \setminus \{i\}$ is a feasible solution and $|(\text{OPT} \cup \{i''\} \setminus \{i\}) \cap A| > |\text{OPT} \cap A|$ which is a contradiction. Therefore, A is a core set for $\mathcal{T}'$.

Now we analyse the running time of $\mathcal{A}^{\text{kr}}$. Each of the recursive calls generated by $\mathcal{A}^{\text{kr}}$ is associated to a path P_{s_i,t_i} such that $|P_{s_i,t_i}| = |P_{s,t}| + 1$, and therefore $p_{s_i} + p_{t_i} \leq p_s + p_t - 1$. Therefore, the recursion depth is at most $p_s + p_t$. In the recursion tree each node has at most

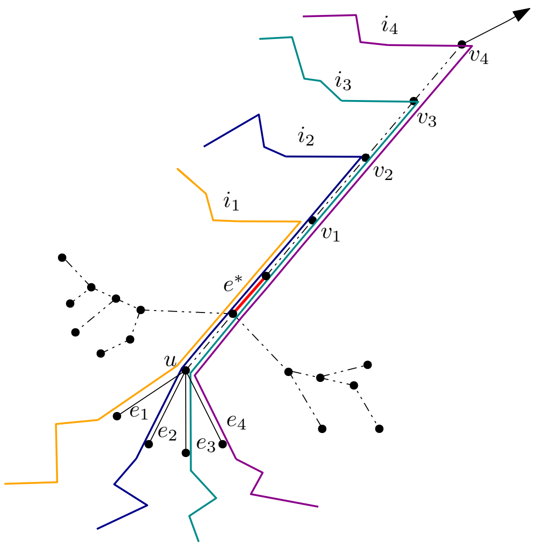

Figure 3 The sets $A = \{i_1, i_2, i_3, i_4\}$, $\overline{V} = \{v_1, v_2, v_3, v_4\}$, and $\overline{E} = \{e_1, e_2, e_3, e_4\}$ as they are defined in algorithm $\mathcal{A}^{\text{fin}}(P, d)$.

$4k$ childrens. If in a leaf of the recursion tree a core set is returned, this core set contains at most k tasks. We conclude that K is a core set of size at most $(4k)^{p_s+p_t} \cdot k$ which we compute in time $(4k)^{p_s+p_t} \cdot k \cdot O(n)$. ◀

We will use $\mathcal{A}^{\text{kr}}$ as a subroutine in a different algorithm $\mathcal{A}^{\text{fin}}$ for computing core sets which we describe now. Let e^* be a final edge. Let $H(e^*)$ be the hanging tree underneath e^*, i.e., that is rooted at the vertex incident to e^* that is further away from the root. The input of $\mathcal{A}^{\text{fin}}$ consists of a path $P \subseteq H(e^*) \cup \{e^*\}$ and of a value $d \in \mathbb{N}$ which is intuitively the demand of some input task. Let $\mathcal{T}'$ denote the set of all tasks i such $P \subseteq P(i)$, $P(i) \cap E_{\text{hs}} = \{e^*\}$ and $d(i) = d$. The algorithm $\mathcal{A}^{\text{fin}}$ will compute a core set for $\mathcal{T}'$. In order to present the algorithm, we introduce some definitions. We say that a task $i \in \mathcal{T}'$ *turns* at a node v if v is the endnode of the path $P(i) \cap E^{\star}_{\text{hs}}$ that is closest to the root. For each task i we denote by $v(i)$ the vertex at which i turns. We define that the *level* $\ell(i)$ of a task $i \in \mathcal{T}'$ is the distance between $v(i)$ and e^*, i.e., the number of edges of the path that connects $v(i)$ with the vertex incident to e^* that is closer to the root.

Our algorithm $\mathcal{A}^{\text{fin}}(P, d)$ works as follows (see Figure 3 for an illustration):

1. Initialize $A = \emptyset$. Consider tasks in $\mathcal{T}'$ ordered non-decreasingly by their levels, add each task $i \in \mathcal{T}'$ to A if for each previously added task $i' \in A$ it holds that $P(i) \cap P(i') \subseteq E^{\star}_{\text{hs}} \cup P$. Stop if $|A| = 2k$ or if no more task in $\mathcal{T}'$ can be added to A.
2. Let $\overline{V} = \{v_1, ..., v_{|V'|}\}$ be the vertices at which the tasks in A turn, u be the endvertex of P that is in $H(e^*)$, and $\overline{E}$ be the set of edges in $\delta(u) \setminus P$ that are used by tasks in A.
3. For every vertex $v_j \in \overline{V}$ we compute $K^{\text{kr}}_j = \mathcal{A}^{\text{kr}}(P_{v_j,u}, d)$.
4. For every edge $e_j \in \overline{E}$ we define $P_j = P \cup \{e_j\}$ and compute $K^{\text{fin}}_j = \mathcal{A}^{\text{fin}}(P_j, d)$.
5. Return

$$K = \left(\bigcup_{j \in \{1,...,|\overline{V}|\}} K^{\text{kr}}_j \right) \cup \left(\bigcup_{j \in \{1,...,|\overline{E}|\}} K^{\text{fin}}_j \right).$$

We want to show that $\mathcal{A}^{\text{fin}}(P,d)$ computes a core set with bounded size for $\mathcal{T}'$. If $|A| = 2k$ we can show by an exchange argument that there is an optimal solution OPT such that each task in $\text{OPT} \cap \mathcal{T}'$ turns at some vertex in V' or use some edge in E'. If $|A| < 2k$ one can show that this is automatically satisfied. In the calls to $\mathcal{A}^{\text{kr}}$ we obtain core sets that together contain all tasks in $\text{OPT} \cap \mathcal{T}'$ that turn at some vertex in V'. The recursive calls to $\mathcal{A}^{\text{fin}}$ compute core sets that together contain all tasks in $\text{OPT} \cap \mathcal{T}'$ that use some edge in E'. Observe that for $\mathcal{A}^{\text{fin}}$ the tasks are ordered, favoring tasks of lower level, since those intutively use less edges on $E^{\star}_{\text{hs}}$ and thus intersect with fewer other tasks on these edges. We define $p_{\max}$ to be the maximum value p_u over all nodes $v \in V$. Since due to Lemma 9 we can assume that that $p_v \leq (\bar{d}+1)^k + 1$ for every $v \in V$, we obtain that $p_{\max} \leq (\bar{d}+1)^k + 1$.

▶ **Lemma 14.** *Let e^* be a final edge, let $P \subseteq H(e^*) \cup \{e^*\}$ be a path, and let $\mathcal{T}'$ be the set of all tasks i such that $P \subseteq P(i)$, $P(i) \cap E_{\text{hs}} = \{e^*\}$, and $d(i) = d$. The algorithm $\mathcal{A}^{\text{fin}}(P,d)$ computes a core set for $\mathcal{T}'$ of size at most $(4k)^{2p_{\max}} \cdot k$ in time $(4k)^{2p_{\max}} \cdot k^2 \cdot O(n)$.*

3.2 UFT Algorithm

Now we describe our main FPT-algorithm for UFT which will use $\mathcal{A}^{\text{fin}}(e,d)$ as a subroutine. Recall that there are at most $6k$ edges in E_{hs}. We take an arbitrary final edge $e \in E_{\text{hs}}$. We guess whether there is a task $i \in \text{OPT}$ that uses e but no other edge in E_{hs}, i.e., such that $P(i) \cap E_{\text{hs}} = \{e\}$. If yes, we guess $d(i)$ for which there are only $\bar{d}$ options. Then we call $\mathcal{A}^{\text{fin}}(e, d(i))$ and obtain a core set K of size at most $(4k)^{2p_{\max}}k^2$. Assume w.l.o.g. that OPT is an optimal solution with the property that K contains all tasks in OPT that use e but no other edge in E_{hs} (using that K is a core set). In time $(4k)^{2p_{\max}}k^2$ we guess a task $i \in K \cap \text{OPT}$. We add i to our computed solution and remove i from the set of input tasks $\mathcal{T}$. We update the edges capacities, taking into account that we selected i, i.e., we update $u(e) := u(e) - d(i)$ for each edge $e \in P(i)$. At this point, we remove from $\mathcal{T}$ each task $i' \in \mathcal{T}$ such that $u(e) < d(i)$ for some edge $e \in P(i)$. We keep on guessing whether there is a task $i \in \text{OPT}$ that uses e but no other edge in E_{hs}, i.e., $P(i) \cap E_{\text{hs}} = \{e\}$. If yes, we repeat the process above. Otherwise, we know that there is no more task $i \in \text{OPT}$ with $P(i) \cap E_{\text{hs}} = \{e\}$. Then we remove e from E_{hs}, we remove from $\mathcal{T}$ all input tasks i with $P(i) \cap E_{\text{hs}} = \{e\}$, and we define $E^{\star}_{\text{hs}}$ newly based on the changed set E_{hs}. Then we apply Lemma 9 to the resulting instance.

We repeat the above process with a final edge e' in the (changed) set E_{hs} (note that possibly e' was not a final edge in the original set E_{hs}). We continue until we selected k tasks in total. If all our guesses were correct, then $\mathcal{S}$ forms a feasible solution. Moreover, there are at most $O(k)$ guesses in total with at most $\bar{d} \cdot (4k)^{2p_{\max}}k^2$ possibilities for each guess: if we guess that there is a task $i \in \text{OPT}$ with $P(i) \cap E_{\text{hs}} = \{e\}$ then subsequently we select such a task and this happens at most k times. On the other hand, if we guess that there is no task $i \in \text{OPT}$ with $P(i) \cap E_{\text{hs}} = \{e\}$ then subsequently we remove e from E_{hs} and the initial set E_{hs} has only $O(k)$ edges. For each guess there are at most $\bar{d} \cdot (4k)^{2p_{\max}}k^2$ options. Hence, there are only $\left(\bar{d} \cdot (4k)^{2p_{\max}}k^2\right)^{O(k)}$ possibilities for all our guesses overall. Therefore, we obtain a running time of $k^{O(k \cdot \bar{d}^k)} \cdot O(n^2)$ overall, which yields the proof of Theorem 1.

4 Resource augmentation

In this section we present an FPT-algorithm for UFT under $(1+\delta)$-resource augmentation. Formally, we present an algorithm with a running time of the form $f(k,\delta)n^{O(1)}$ for some computable function f that outputs a set $\mathcal{S} \subseteq \mathcal{T}$ with $|\mathcal{S}| = k$ that is feasible under $(1+\delta)$-resource augmentation, i.e., $\sum_{i:\, i \in \mathcal{S} \cap \mathcal{T}_e} d(i) \leq (1+\delta)u(e)$ for each $e \in E$, or outputs that there is no solution of size k (for the original edge capacities).

Our strategy is to split the input tasks $\mathcal{T}$ into groups such that the demands of any two tasks in different groups differ at least by a factor k/δ. Then we can compute a solution for each group separately and their union is a (global) solution under resource augmentation, as the following lemma shows.

▶ **Lemma 15.** *Consider sets of tasks $\{\mathcal{T}_j\}_{j\in\mathbb{N}}$ with $\mathcal{T}_j \subseteq \mathcal{T}$ for each $j \in \mathbb{N}$ such that for each $j, j' \in \mathbb{N}$ with $j \neq j'$ and any two tasks $T_i \in \mathcal{T}_j$, $T_{i'} \in \mathcal{T}_{j'}$ it holds that $d(i) > \frac{k}{\delta}d(i')$ or $d(i') > \frac{k}{\delta}d(i)$. For each $j \in \mathbb{N}$ let $\mathcal{S}_j \subseteq \mathcal{T}_j$ be a solution with at most k tasks that is feasible under $(1+\delta')$-resource augmentation for some $\delta' \geq 0$, and suppose that $\delta < 1/2$.Then $\bigcup_{j\in\mathbb{N}} \mathcal{S}_j$ is feasible under $(1 + 2\delta' + 2\delta)$-resource augmentation.*

On the other hand, if we can ensure that within each group the task demands differ by at most a factor $(k/\delta)^k$, then we can compute a solution of size k for this group that is feasible under $(1 + \delta)$-resource augmentation as follows: we first round the task demands to powers of $1 + \delta$ and then invoke our algorithm due to Theorem 1, using that the number of different task demands is only $\log_{1+\delta}((k/\delta)^k)$.

▶ **Lemma 16.** *Consider a set of tasks $\mathcal{T}' \subseteq \mathcal{T}$ such that for any two tasks $i, i' \in \mathcal{T}'$ it holds that $(\frac{\delta}{k})^k d(i) \leq d(i') \leq (\frac{k}{\delta})^k d(i)$. Then in time $k^{O\left(k^{k+2}\cdot\left(\log_{1+\delta}(k/\delta)\right)^{k+1}\right)} \cdot O(n^2)$ we can compute for any $k' \leq k$ a solution $\mathcal{S}' \subseteq \mathcal{T}'$ of size k' that is feasible under $(1 + \delta)$-resource augmentation, or assert that there is no solution of size k' (for the original edge capacities).*

Now with a shifting argument we can guess sets of tasks $\{\mathcal{T}_j\}_{j\in\mathbb{N}}$ that satisfy the condition of Lemma 15, such that each set $\mathcal{T}_j$ satisfies the condition of Lemma 16, and additionally $\bigcup_{j\in\mathbb{N}} \mathcal{T}_j$ contains a solution of size k. We apply Lemma 16 to each set $\mathcal{T}_j$ to find the largest $k' \leq k$ for which there exists a solution, denote by $\mathcal{S}_j$ the largest solution found, and output the union $\bigcup_{j\in\mathbb{N}} \mathcal{S}_j$ which is feasible under $(1 + 4\delta)$-resource augmentation due to Lemma 15.

▶ **Theorem 17.** *There is an FPT-algorithm for UFT under $(1 + \delta)$-resource augmentation with a running time of $k^{O\left(\left(k\cdot\log_{1+\delta}(k/\delta)\right)^{k+2}\right)} \cdot O(n^2)$.*

5 FPT-approximation algorithm

In this section we present an FPT-5-approximation algorithm for UFT, i.e., given a parameter k, in time $f(k)n^{O(1)}$ our algorithm either finds a solution of size $k/5$ or asserts that there are no solution of size k. First, we present a simpler FPT-7-approximation algorithm and then explain how to improve it to an FPT-5-approximation.

Again, we invoke Lemmas 6 and 8 to construct a good hitting set E_{hs} (or directly obtain a solution of size k). Since $|E_{\text{hs}}| = O(k)$, we observe that the graph $(V, E \setminus E_{\text{hs}})$ has $K = O(k)$ connected components; we denote them by $G_1, ..., G_K$. Since E_{hs} is a hitting set, we obtain the following lemma.

▶ **Lemma 18.** *For each task $i \in \mathcal{T}$ the two endvertices of $P(i)$ lie in two different components in $\{G_1, ..., G_K\}$.*

Let $i_1^* \in \text{OPT}$ denote the task with smallest demand in OPT. We guess the two components of the two endvertices of $P(i_1^*)$. We select the input task i_1 with smallest demand whose path has its endvertices in the same two components. It might be that $i_1 \neq i_1^*$; however, we will show that we can find seven tasks $i'_1, ..., i'_7 \in \text{OPT}$ such that if we replace $i'_1, ..., i'_7$ by i_1 then we still obtain a feasible solution. Intuitively, the task i_1 (which we select) pays for the tasks $i'_1, ..., i'_7$ (which we lose). We continue iteratively until we picked $\lceil k/7 \rceil$ tasks.

Formally, our algorithm runs in $\lceil k/7 \rceil$ rounds where in each round we select one task i. Let $\mathrm{OPT}_0 = \mathrm{OPT}$. Suppose that after $k' \in \{0, ..., \lceil k/7 \rceil - 1\}$ rounds we selected tasks $i_1, ..., i_{k'}$ and defined a solution $\mathrm{OPT}_{k'} \subseteq \mathrm{OPT}$ such that $\mathrm{OPT}_{k'} \cup \{i_1, ..., i_{k'}\}$ is feasible (note that this is clearly true for $k' = 0$). Let $i^*_{k'+1} \in \mathrm{OPT}_{k'}$ denote a task with smallest demand in $\mathrm{OPT}_{k'}$, i.e., such that $d(i^*_{k'+1}) \leq d(i)$ for each $i \in \mathrm{OPT}_{k'}$. We guess the two components from $\{G_1, ..., G_K\}$ that contain the two endvertices of $P(i^*_{k'+1})$. Let $i_{k'+1}$ denote the input task i of smallest demand with the property that $P(i)$ has its endvertices in the same components and $\{i, i_1, ..., i_{k'}\}$ forms a feasible solution. We select $i_{k'+1}$ and prove in the next lemma that intuitively we can select $i_{k'+1}$ if we are willing to sacrifice at most seven tasks from $\mathrm{OPT}_{k'}$.

▶ **Lemma 19.** *There exist tasks $i'_1, ..., i'_7 \in \mathrm{OPT}_{k'}$ such that $\mathrm{OPT}_{k'} \setminus \{i'_1, ..., i'_7\} \cup \{i_1, ..., i_{k'}, i_{k'+1}\}$ is feasible.*

Proof. Let G' and G'' denote the two components from $\{G_1, ..., G_K\}$ that contain the two endvertices of $P(i^*_{k'+1})$. The idea behind the proof is to identify seven tasks $i'_1, ..., i'_7 \in \mathrm{OPT}_{k'}$ such that $d(i'_\ell) \geq d(i^*_{k'+1}) \geq d(i_{k'+1})$ for each $\ell \in \{1, ..., 7\}$ and such that the edges of their paths contain all the edges of $P(i_{k'+1})$ that are used by tasks in $\mathrm{OPT}_{k'}$ (i.e., edges on which selecting $i_{k'+1}$ potentially causes conflicts with other tasks in $\mathrm{OPT}_{k'}$). We will select three tasks for the edges in $P(i_{k'+1}) \cap E(G')$, three for the edges in $P(i_{k'+1}) \cap E(G'')$, and one for the edges in $P(i_{k'+1}) \setminus (E(G') \cup E(G''))$.

Consider the edges in $P(i_{k'+1}) \cap E(G')$. We partition them into $P(i_{k'+1}) \cap E(G') \setminus E^\star_{\mathrm{hs}}$ and $P(i_{k'+1}) \cap E(G') \cap E^\star_{\mathrm{hs}}$. If an edge $e \in P(i_{k'+1}) \cap E(G') \setminus E^\star_{\mathrm{hs}}$ is used by some task $i \in \mathrm{OPT}_{k'}$, then e is also used by the task $i' \in \mathrm{OPT}_{k'}$ that maximizes $|P(i') \cap P(i_{k'+1}) \cap E(G') \setminus E^\star_{\mathrm{hs}}|$. Let i'_1 denote this task i'. Regarding the edges in $P(i_{k'+1}) \cap E(G') \cap E^\star_{\mathrm{hs}}$, note that they form a path, and let u and v be its end vertices. Let i'_2 denote the task $i \in \mathrm{OPT}_{k'}$ such that $u \in V(P(i))$ and i maximizes $|P(i) \cap E(G') \cap E^\star_{\mathrm{hs}}|$. Similarly, let i'_3 denote the task $i \in \mathrm{OPT}_{k'}$ such that $v \in V(P(i))$ and i maximizes $|P(i) \cap E(G') \cap E^\star_{\mathrm{hs}}|$. Then, if some task $i \in \mathrm{OPT}_{k'}$ uses an edge $e \in P(i_{k'+1}) \cap E(G') \cap E^\star_{\mathrm{hs}}$ then $e \in P(i'_2)$ or $e \in P(i'_3)$ (see Figure 4). In a similar way, we identify three tasks i'_4, i'_5 and i'_6 for $P(i_{k'+1}) \cap E(G'')$.

Finally, all the edges in $P(i_{k'+1}) \setminus (G' \cup G'')$ are used by $i^*_{k'+1}$, and we define $i'_7 := i^*_{k'+1}$. If an edge e is used by some task $i \in \mathrm{OPT}_{k'}$, then $e \in P(i'_\ell)$ for some $\ell \in \{1, ..., 7\}$. Since $d(i'_\ell) \geq d(i^*_{k'+1}) \geq d(i_{k'+1})$ for each $\ell \in \{1, ..., 7\}$, the tasks $i'_1, ..., i'_7$ satisfy the claim of the lemma. ◀

We remark that the tasks $i'_1, ..., i'_7$ might not be pairwise distinct. We define $\mathrm{OPT}_{k'+1} := \mathrm{OPT}_{k'} \setminus \{i'_1, ..., i'_7\}$ and iterate. Since in each iteration we picked one task and removed at most seven tasks from OPT, one can show easily that at the end we select $\lceil k/7 \rceil$ tasks, assuming that $|\mathrm{OPT}| \geq k$. Our running time is $k^{O(k)} n^{O(1)}$ since in each of the k iterations there are $O(k^2)$ options for our guesses.

▶ **Theorem 20.** *There is an FPT-7-approximation algorithm for UFT with a running time of $k^{O(k)} n^{O(1)}$.*

5.1 Improvement to an FPT-5-approximation

We improve our approximation factor to 5 by doing additional guesses when we select the task $i_{k'}$ in each round k'. Assume again that at the beginning of round $k'+1$ we have already selected tasks $i_1, ..., i_k$, and defined a set $\mathrm{OPT}_{k'} \subseteq \mathrm{OPT}$ such that $\mathrm{OPT}_{k'} \cup \{i_1, ..., i_{k'}\}$ is feasible. Like before, let $i^*_{k'+1} \in \mathrm{OPT}_{k'}$ denote the task with smallest demand in $\mathrm{OPT}_{k'}$ and we guess the components from $G_1, ..., G_K$ that contain the endvertices of $P(i^*_{k'+1})$. Let

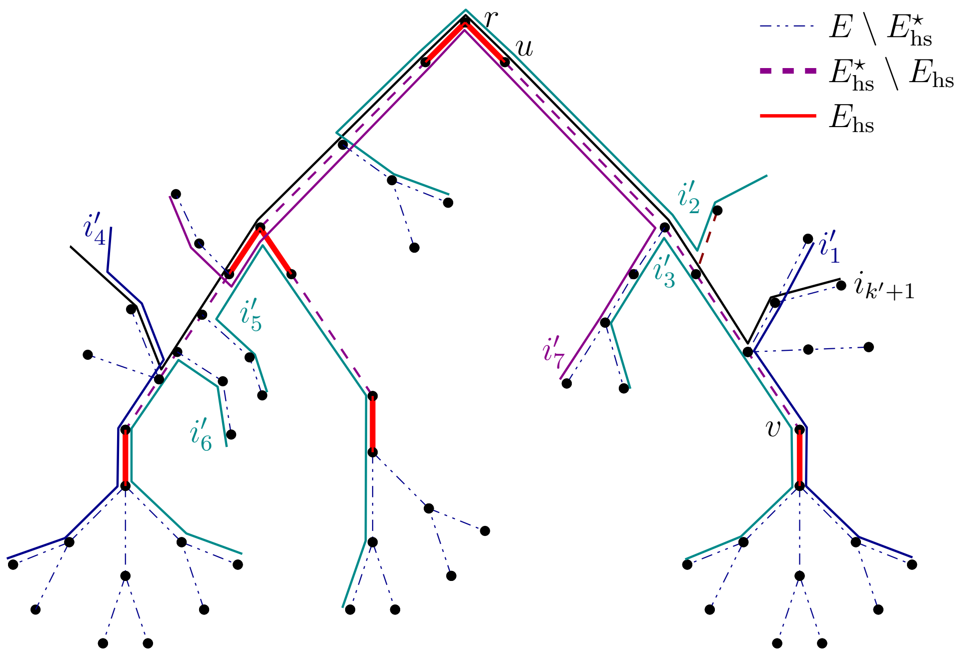
r
u
v
i'_1
i'_2
i'_3
i'_4
i'_5
i'_6
i'_7
$i_{k'+1}$
$E \setminus E^{\star}_{\text{hs}}$
$E^{\star}_{\text{hs}} \setminus E_{\text{hs}}$
E_{hs}

Figure 4 The task $i_{k'+1}$ is selected, and Lemma 19 shows that if we remove the tasks $i'_1, \ldots i'_7$ from $\text{OPT}_{k'}$, we can add the task $i_{k'+1}$ to $\text{OPT}_{k'}$ since each edge on $P(i_{k'+1})$ is used by one of the tasks $i'_1, \ldots i'_7$ or by no task from $\text{OPT}_{k'}$.

G' and G'' be these components. Like before, let $i_{k'+1}$ denote the input task i of smallest demand with the property that $P(i)$ has its endvertices in G' and G'' and $\{i, i_1, \ldots, i_{k'}\}$ forms a feasible solution. We do not select $i_{k'+1}$ yet; instead, we guess some properties of $P(i^*_{k'+1})$, more precisely, of the part of $P(i^*_{k'+1})$ within G' and G''. Since E_{hs} is a good hitting set, it holds that G' (or G'') restricted to $E^{\star}_{\text{hs}}$ is a path.

▶ **Lemma 21.** *Let $\tilde{G} \in \{G_1, \ldots, G_K\}$. Then $G[E^{\star}_{\text{hs}} \cap E(\tilde{G})]$ is a path (possibly with zero edges).*

Let $\bar{G}' := G[E^{\star}_{\text{hs}} \cap E(G')]$ and $\bar{G}'' := G[E^{\star}_{\text{hs}} \cap E(G'')]$. Observe that $P(i^*_{k'+1})$ uses some edges of $\bar{G}'$ and also $P(i_{k'+1})$ uses some edges of $\bar{G}'$. Note that the respective sets of edges are contained in each other. The same holds for $\bar{G}''$. It turns out that for Lemma 19 a particularly bad case is when $P(i_{k'+1})$ uses strictly more edges of both $\bar{G}'$ and $\bar{G}''$ than $P(i^*_{k'+1})$, i.e., then we need to remove seven tasks from $\text{OPT}_{k'}$, rather than fewer tasks. Therefore, we guess whether this bad case applies. If yes, instead of $i_{k'+1} =: i^{(1)}_{k'+1}$ we consider a task $i^{(2)}_{k'+1}$ which is the input task i of smallest demand with endvertices in G' and G'', such that $\{i, i_1, \ldots, i_{k'}\}$ forms a feasible solution and $P(i)$ uses strictly less edges than $P(i^{(1)}_{k'+1})$ of both $\bar{G}'$ and $\bar{G}''$. Again, we guess whether the bad case applies for $i^{(2)}_{k'+1}$. If yes, we replace $i^{(2)}_{k'+1}$ by some task $i^{(3)}_{k'+1}$ that uses even fewer edges of $\bar{G}'$ and $\bar{G}''$ and so on. We repeat this for at most $2k$ iterations. Formally, if for some $\ell \in \{1, \ldots, 2k\}$ we defined the task $i^{(\ell)}_{k'+1}$ then we guess whether the bad case applies for $i^{(\ell)}_{k'+1}$, i.e., whether $P(i^{(\ell)}_{k'+1})$ uses strictly more edges of both $\bar{G}'$ and $\bar{G}''$ than $P(i^*_{k'+1})$. If yes, we define the task $i^{(\ell+1)}_{k'+1}$ to be the input task i of smallest demand with endvertices in G' and G'', such that $\{i, i_1, \ldots, i_{k'}\}$ forms a feasible solution and $P(i)$ uses strictly less edges than $P(i^{(\ell)}_{k'+1})$ of both $\bar{G}'$ and $\bar{G}''$. If for some $i^{(\ell)}_{k'+1}$ with $\ell \in \{1, \ldots, 2k\}$ the bad case does not apply then intuitively we can show that for adding $i^{(\ell)}_{k'+1}$ we need to remove only five tasks from OPT, rather than seven. In this case we select $i_k := i^{(\ell)}_{k'+1}$ and define $\mathrm{OPT}_{k'+1} := \mathrm{OPT}_{k'} \setminus \{i'_1, ..., i'_5\}$ for the tasks $i'_1, ..., i'_5$ due to the following lemma, and continue with the next round $k' + 2$.

▶ **Lemma 22.** *Suppose that for some $\ell \in \{1, ..., 2k\}$ it holds that $P(i^{(\ell)}_{k'+1})$ does not use strictly more edges of both $\bar{G}'$ and $\bar{G}''$ than $P(i^*_{k'+1})$. Then there exist tasks $i'_1, ..., i'_5 \in \mathrm{OPT}_{k'}$ such that $\mathrm{OPT}_{k'} \setminus \{i'_1, ..., i'_5\} \cup \{i_1, ..., i_{k'}, i^{(\ell)}_{k'+1}\}$ is feasible.*

Proof. Assume w.l.o.g. that $P(i^{(\ell)}_{k'+1})$ does not use strictly more edges of $\bar{G}''$ than $P(i^*_{k'+1})$. Following the idea behind Lemma 19 we identify five tasks $i'_1, ..., i'_5 \in \mathrm{OPT}_{k'}$ such that $d(i'_j) \geq d(i^*_{k'+1}) \geq d(i^{(\ell)}_{k'+1})$ for each $j \in \{1, ..., 5\}$ and such that the edges of their paths (together) contain all the edges of $P(i^{(\ell)}_{k'+1})$ that are used by tasks in $\mathrm{OPT}_{k'}$ (i.e., edges on which selecting $i^{(\ell)}_{k'+1}$ potentially causes conflicts with other tasks in $\mathrm{OPT}_{k'}$). We will select three tasks for the edges in $P(i^{(\ell)}_{k'+1}) \cap E(G')$, one for the edges in $P(i^{(\ell)}_{k'+1}) \cap E(G'') \setminus E(\bar{G}'')$, and one for the edges in $P(i^{(\ell)}_{k'+1}) \setminus (E(G') \cup E(G''))$ and the edges in $P(i^{(\ell)}_{k'+1}) \cap E(\bar{G}'')$.

Consider the edges in $P(i^{(\ell)}_{k'+1}) \cap E(G')$. We partition them into $P(i^{(\ell)}_{k'+1}) \cap E(G') \setminus E(\bar{G}')$ and $P(i^{(\ell)}_{k'+1}) \cap E(\bar{G}')$. If an edge $e \in P(i^{(\ell)}_{k'+1}) \cap E(G') \setminus E(\bar{G}')$ is used by some task $i \in \mathrm{OPT}_{k'}$, then e is also used by the task $i' \in \mathrm{OPT}_{k'}$ that maximizes $|P(i') \cap P(i^{(\ell)}_{k'+1}) \cap E(G') \setminus E(\bar{G}')|$. Let i'_1 denote this task i'. Regarding the edges in $P(i^{(\ell)}_{k'+1}) \cap E(\bar{G}')$, note that they form a path, and let u and v be its end vertices. Let i'_2 denote the task $i \in \mathrm{OPT}_{k'}$ such that $u \in V(P(i))$ and i maximizes $|P(i) \cap E(\bar{G}')|$. Similarly, let i'_3 denote the task $i \in \mathrm{OPT}_{k'}$ such that $v \in V(P(i))$ and i maximizes $|P(i) \cap E(\bar{G}')|$. Then, if some task $i \in \mathrm{OPT}_{k'}$ uses an edge $e \in P(i^{(\ell)}_{k'+1}) \cap E(\bar{G}')$ then $e \in P(i'_2)$ or $e \in P(i'_3)$.

Regarding the edges in $P(i^{(\ell)}_{k'+1}) \cap E(G'') \setminus E(\bar{G}'')$, if an edge $e \in P(i^{(\ell)}_{k'+1}) \cap E(G'') \setminus E(\bar{G}'')$ is used by some task $i \in \mathrm{OPT}_{k'}$, then e is also used by the task $i' \in \mathrm{OPT}_{k'}$ that maximizes $|P(i') \cap P(i^{(\ell)}_{k'+1}) \cap E(G'') \setminus E(\bar{G}'')|$. Let i'_4 denote this task i'. Regarding the edges in $P(i^{(\ell)}_{k'+1}) \cap E(\bar{G}'')$ and the edges in $P(i^{(\ell)}_{k'+1}) \setminus (E(G') \cup E(G''))$, they are all used by $i^*_{k'+1}$, which we now identify as i'_5.

It follows that if an edge e is used by some task $i \in \mathrm{OPT}_{k'}$, then $e \in P(i'_j)$ for some $j \in \{1, ..., 5\}$. Since $d(i'_j) \geq d(i^*_{k'+1}) \geq d(i_{k'+1})$ for each $j \in \{1, ..., 5\}$, the tasks $i'_1, ..., i'_5$ satisfy the claim of the lemma. ◀

On the other hand, if the bad case applied to each $i^{(\ell)}_{k'+1}$ with $\ell \in \{1, ..., 2k\}$ then we can show that there must be one task $i^{(\ell^*)}_{k'+1}$ with $\ell^* \in \{1, ..., 2k\}$ such that the edges of $P(i^{(\ell^*)}_{k'+1}) \setminus E^{\star}_{\mathrm{hs}}$ are not used by any task in OPT. It turns out that this is again a good case. Intuitively, then we do not need to remove any task from OPT in order to make space for $i^{(\ell^*)}_{k'+1}$ on the edges in $P(i^{(\ell^*)}_{k'+1}) \setminus E^{\star}_{\mathrm{hs}}$ and, therefore, it turns out that we need to remove only five tasks from OPT. In this case we guess ℓ^*, select $i_k := i^{(\ell^*)}_{k'+1}$ and define $\mathrm{OPT}_{k'+1} := \mathrm{OPT}_{k'} \setminus \{i'_1, ..., i'_5\}$ for the tasks $i'_1, ..., i'_5$ due to the following lemma, and continue with the next round $k' + 2$.

▶ **Lemma 23.** *Suppose that for each $\ell \in \{1, ..., 2k\}$ it holds that $P(i^{(\ell)}_{k'+1})$ uses strictly more edges of $\bar{G}'$ and $\bar{G}''$ than $P(i^*_{k'+1})$. Then there is an $\ell^* \in \{1, ..., 2k\}$ for which there exist tasks $i'_1, ..., i'_5 \in \mathrm{OPT}_{k'}$ such that $\mathrm{OPT}_{k'} \setminus \{i'_1, ..., i'_7\} \cup \{i_1, ..., i_{k'}, i^{(\ell^*)}_{k'+1}\}$ is feasible.*

Proof. Since for each $\ell \in \{1, ..., 2k\}$ it holds that $P(i^{(\ell)}_{k'+1})$ uses strictly more edges of $\bar{G}'$ and $\bar{G}''$ than $P(i^*_{k'+1})$, then there is an $\ell^* \in \{1, ..., 2k\}$ such that $|P(i^{(\ell^*)}_{k'+1}) \cap E(\bar{G}') \setminus E^{\star}_{\mathrm{hs}}| = |P(i^{(\ell^*)}_{k'+1}) \cap E(\bar{G}'') \setminus E^{\star}_{\mathrm{hs}}| = 0$. Following the idea behind Lemma 19 we identify five tasks

$i'_1, ..., i'_5 \in \mathrm{OPT}_{k'}$ such that $d(i'_{\ell^*}) \geq d(i^*_{k'+1}) \geq d(i_{k'+1})$ for each $j \in \{1, ..., 5\}$ and such that the edges of their paths contain all the edges of $P(i_{k'+1})$ that are used by tasks in $\mathrm{OPT}_{k'}$ (i.e., edges on which selecting $i_{k'+1}$ potentially causes conflicts with other tasks in $\mathrm{OPT}_{k'}$). Now will select two tasks for the edges in $P(i^{(\ell^*)}_{k'+1}) \cap E(\bar{G}')$, two for the ones in that in $P(i^{(\ell^*)}_{k'+1}) \cap E(\bar{G}'')$ and one for the edges in $P(i^{(\ell^*)}_{k'+1}) \setminus (E(\bar{G}') \cup E(\bar{G}''))$.

Consider the edges in $P(i^{(\ell^*)}_{k'+1}) \cap E(\bar{G}'')$. We partition them into $P(i^{(\ell^*)}_{k'+1}) \cap E(G'') \setminus E(\bar{G}'')$ and $P(i^{(\ell^*)}_{k'+1}) \cap E(\bar{G}'')$. There is no edge $P(i^{(\ell^*)}_{k'+1}) \cap E(G'') \setminus E(\bar{G}'')$ used by some task $i \in \mathrm{OPT}_{k'}$, so we don't need to define a task to cover those edges. Regarding the edges in $P(i^{(\ell^*)}_{k'+1}) \cap E(\bar{G}'')$, we note that they form a path, and let u and v be their end vertices. Let i'_1 denote the task $i \in \mathrm{OPT}_{k'}$ such that $u \in V(P(i))$ and i maximizes $|P(i) \cap E(\bar{G}') \cap E^\star_{\mathrm{hs}}|$. Similarly, let i'_2 denote the task $i \in \mathrm{OPT}_{k'}$ such that $v \in V(P(i))$ and i maximizes $|P(i) \cap E(\bar{G}') \cap E^\star_{\mathrm{hs}}|$. Then, if some task $i \in \mathrm{OPT}_{k'}$ uses an edge $e \in P(i_{k'+1}) \cap E(\bar{G}'')$ then $e \in P(i'_1)$ or $e \in P(i'_2)$. In a similar way, we identify two tasks i'_3 and i'_4 for $P(i^{(\ell^*)}_{k'+1}) \cap E(G'')$.

Finally, all the edges in $P(i^{(\ell^*)}_{k'+1}) \setminus (G' \cup G'')$ are used by $i^*_{k'+1}$, and we define $i'_5 := i^*_{k'+1}$. It follows that if an edge e is used by some task $i \in \mathrm{OPT}_{k'}$, then $e \in P(i'_j)$ for some $j \in \{1, ..., 5\}$. Since $d(i'_j) \geq d(i^*_{k'+1}) \geq d(i_{k'+1})$ for each $j \in \{1, ..., 5\}$, the tasks $i'_1, ..., i'_5$ satisfy the claim of the lemma. ◀

Since in each iteration we picked one task and removed at most five tasks from OPT, one can show easily that at the end we select $\lceil k/5 \rceil$ tasks, assuming that $|\mathrm{OPT}| \geq k$. Our running time is $k^{O(k)} n^{O(1)}$ since in each of the k iterations there are $O(k^2)$ options for our guesses.

▶ **Theorem 24.** *There is an FPT-5-approximation algorithm for UFT with a running time of $k^{O(k)} n^{O(1)}$.*

References

1 Anna Adamaszek, Parinya Chalermsook, Alina Ene, and Andreas Wiese. Submodular unsplittable flow on trees. *Math. Program.*, 172(1-2):565–589, 2018. doi:10.1007/s10107-017-1218-4.

2 Marek Adamczyk, Jaroslaw Byrka, Jan Marcinkowski, Syed Mohammad Meesum, and Michal Wlodarczyk. Constant factor FPT approximation for capacitated k-median. *CoRR*, abs/1809.05791, 2018. arXiv:1809.05791.

3 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. *SIAM Journal on Computing*, 49(4):FOCS17–97, 2019.

4 E. M. Arkin and E. B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18:1–8, 1987.

5 N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *STOC*, pages 721–729. ACM, 2006.

6 Reuven Bar-Yehuda, Michael Beder, Yuval Cohen, and Dror Rawitz. Resource allocation in bounded degree trees. *Algorithmica*, 54(1):89–106, 2009.

7 Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In *SODA*, pages 47–58, 2015. doi:10.1137/1.9781611973730.5.

8 Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 737–756. SIAM, 2014.

9 Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha. Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC 1998)*, pages 114–123, 1998.

10 C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *APPROX-RANDOM*, pages 42–55, 2009.

11 C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3, 2007.

12 Bo Chen, Refael Hassin, and Michal Tzur. Allocation of bandwidth and storage. *IIE Transactions*, 34(5):501–507, 2002.

13 Rajesh Chitnis, MohammadTaghi Hajiaghayi, and Guy Kortsarz. Fixed-parameter and approximation algorithms: A new look. In *International Symposium on Parameterized and Exact Computation*, pages 110–122. Springer, 2013.

14 M. Chrobak, G. Woeginger, K. Makino, and H. Xu. Caching is hard, even in the fault model. In *ESA*, pages 195–206, 2010.

15 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT Approximations for k-Median and k-Means. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.42.

16 Vincent Cohen-Addad and Jason Li. On the Fixed-Parameter Tractability of Capacitated Clustering. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41:1–41:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.41.

17 Andreas Emil Feldmann, CS Karthik, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.

18 Zachary Friggstad and Zhihan Gao. On Linear Programming Relaxations for Unsplittable Flow in Trees. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 265–283, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.265.

19 N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–20, 1997. doi:10.1007/BF02523685.

20 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. A $(5/3 + \epsilon)$-approximation for unsplittable flow on a path: placing small tasks into boxes. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 607–619, 2018. doi:10.1145/3188745.3188894.

21 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of algorithms*, 31(1):228–248, 1999.

22 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 585–594, 2003.

23 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. In *International Colloquium on Automata, Languages, and Programming*, pages 77–88. Springer, 2011.

24 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

25 Andreas Wiese. A (1+epsilon)-approximation for unsplittable flow on a path in fixed-parameter running time. In *ICALP 2017*, volume 80 of *LIPIcs*, pages 67:1–67:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.67.