Fixed-Parameter Algorithms for Unsplittable Flow Cover

Andrés Cristi
Universidad de Chile, Santiago, Chile
andres.cristi@ing.uchile.cl

Mathieu Mari
École Normale Supérieure, Université PSL, Paris, France
mathieu.mari@ens.fr

Andreas Wiese
Universidad de Chile, Santiago, Chile
awiese@dii.uchile.cl

Abstract

The Unsplittable Flow Cover problem (UFP-cover) models the well-studied general caching problem and various natural resource allocation settings. We are given a path with a demand on each edge and a set of tasks, each task being defined by a subpath and a size. The goal is to select a subset of the tasks of minimum cardinality such that on each edge e the total size of the selected tasks using e is at least the demand of e. There is a polynomial time 4-approximation for the problem [Bar-Noy et al., STOC 2000] and also a QPTAS [Höhn et al., ICALP 2014]. In this paper we study fixed-parameter algorithms for the problem. We show that it is W[1]-hard but it becomes FPT if we can slightly violate the edge demands (resource augmentation) and also if there are at most k different task sizes. Then we present a parameterized approximation scheme (PAS), i.e., an algorithm with a running time of $f(k) \cdot n^{O_\epsilon(1)}$ that outputs a solution with at most $(1+\epsilon)k$ tasks or assert that there is no solution with at most k tasks. In this algorithm we use a new trick that intuitively allows us to pretend that we can select tasks from OPT multiple times.

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Theory of computation → Fixed parameter tractability

Keywords and phrases Unsplittable Flow Cover, fixed parameter algorithms, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.42

Funding *Andrés Cristi*: Supported by CONICYT-PFCHA/Doctorado Nacional/2018-21180347.
Mathieu Mari: This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).
Andreas Wiese: Partially supported by FONDECYT Regular grant 1170223.

1 Introduction

In the Unsplittable Flow Cover problem (UFP-cover) we are given a path $G = (V, E)$ where each edge e has a demand $u_e \in \mathbb{N}$, and a set of tasks T where each task $i \in T$ has a start vertex $s_i \in V$ and an end vertex $t_i \in V$, defining a path $P(i)$, and a size $p_i \in \mathbb{N}$. The goal is to select a subset of the tasks $T' \subseteq T$ of minimum cardinality $|T'|$ that covers the demand of each edge, i.e., such that $\sum_{i \in T' \cap T_e} p_i \geq u_e$ for each edge e where T_e denotes the set of tasks $i \in T$ for which e lies on $P(i)$. It is the natural covering version of the well-studied Unsplittable Flow on a Path problem (UFP), see e.g., [22, 21, 9] and references therein. Also, it is a generalization of the knapsack cover problem [11] and it can model general caching in the fault model where we have a cache of fixed size and receive requests for non-uniform size pages, the goal being to minimize the total number of cache misses (see [1, 5, 16] and Appendix A). Caching and generalizations of it have been studied for several decades in computer science, see e.g., [1, 8, 20, 24]. Also, UFP-cover is motivated by

many resource allocation settings in which for instance the path specifies a time interval and the edge demands represent minimum requirements for some resource like energy, bandwidth, or number of available machines at each point in time.

UFP-cover is strongly NP-hard, since it generalizes general caching in the fault model [16], and the best known polynomial time approximation algorithm for it is a 4-approximation [5] with no improvement in almost 20 years. However, the problem admits a QPTAS for the case of quasi-polynomial input data [23] which suggests that better polynomial time approximation ratios are possible.

In this paper, we study the problem for the first time under the angle of fixed parameter tractability (FPT). We define our parameter k to be the number of tasks in the desired solution and seek algorithms with a running time of $f(k)n^{O(1)}$ for some function f. We show that by allowing such a running time we can compute solutions that are almost optimal.

1.1 Our contribution

First, we prove that UFP-cover is W[1]-hard which makes it unlikely that it admits an FPT-algorithm. In particular, this motivates studying FPT-approximation algorithms or other relaxations of the problem. We first show that under slight resource augmentation the problem becomes FPT. We define an additional parameter $\delta > 0$ controlling the amount of resource augmentation and we compute either a solution that is feasible if we decrease the demand of each edge e to $u_e/(1+\delta)$, or we assert that there is no solution of size k for the original edge demands. Key to our result is to prove that due to the resource augmentation we can assume that each edge e is completely covered by tasks whose size is comparable to u_e or it is covered by at least one task whose size is much larger than u_e. Based on this we design an algorithm that intuitively sweeps the path from left to right and on each uncovered edge e we guess which of the two cases applies. In the former case, we show that due to the resource augmentation we can restrict ourselves to only $f(k,\delta)$ many guesses for the missing tasks using e. In the latter case e belongs to a subpath in which each edge is covered by a task that is much larger than the demand of e. We guess the number of tasks in this subpath and select tasks to maximize the length of the latter. This yields a subproblem that we solve recursively and we embed the recursion into a dynamic program.

▶ **Theorem 1.** *There is an algorithm for UFP-cover with running time $k^{O(\frac{k}{\delta}\log k)} \cdot n^{O(1)}$ that either outputs a solution of size at most k that is feasible if the edge capacities are decreased by a factor $1+\delta$ or asserts that there is no solution of size k for the original edge capacities.*

We use the above algorithm to obtain a simple FPT-2-approximation algorithm *without* resource augmentation. Also, with similar ideas we derive an algorithm computing the *optimal* solution, assuming that additionally the number of different task sizes in the input is bounded by a parameter.

▶ **Theorem 2.** *There is an algorithm that solves UFP-cover in time $k^{O(k'k)} \cdot n^{O(1)}$, assuming that $|\{p_i : i \in T\}| \leq k'$.*

Then we present a parameterized approximation scheme (PAS) for UFP-cover, i.e., an algorithm with a running time of $f(k) \cdot n^{O_\epsilon(1)}$ that outputs a solution with at most $(1+\epsilon)k$ tasks or assert that there is no solution with at most k tasks. This algorithm is based on a lemma developed for UFP in which we have the same input as in UFP-cover but we want to *maximize* the weight of the selected tasks T' and require that their total size is *upper*-bounded by u_e on each edge e, i.e., $\sum_{i \in T' \cap T_e} p_i \leq u_e$. Informally, the mentioned lemma states that we can remove a set of tasks from $OPT_{SL} \subseteq OPT$ of negligible cardinality

such that on each edge e we remove one of the largest tasks of OPT using e. This yields some slack that we can use in order to afford inaccuracies in the computation. Translated to UFP-cover, the natural correspondence would be a solution in which the tasks in OPT_{SL} are not removed but selected twice. This is not allowed in UFP-cover. However, we guess a set of tasks T' that intuitively yields as much slack as OPT_{SL} and whose size is also negligible. If $OPT \cap T' \neq \emptyset$ then we cannot add the tasks in T' to OPT to gain slack since some of them are already included in OPT. Therefore, we use the following simple but useful trick: we guess $T' \cap OPT$ for which there are $2^{|T'|} \leq 2^{O(\epsilon k)}$ options, select the tasks in $T' \cap OPT$, and recurse on the remaining instance. Since $OPT \leq k$ the whole recursion tree has a complexity of $O(k^k)$ which depends only on our parameter k.

If $OPT \cap T' = \emptyset$ then $T' \cup OPT$ is a $(1+\epsilon)$-approximate solution with some slack and we can use the slack in our computation. We compute a partition of E into $O(k)$ intervals. Some of these intervals are *dense*, meaning that there are many tasks from OPT that start or end in them. We ensure that for each dense interval there is a task in T' that covers the whole interval and whose size is at least a $\Omega(1/k)$-fraction of the demand of each edge in the interval. Intuitively this is equivalent to decreasing the demand on each edge by a factor $1 + \Omega(1/k)$. If we had only dense intervals we could apply the FPT-algorithm for resource augmentation from above for the remaining problem. On the other hand, if only few tasks start or end in an interval we say that it is *sparse*. If all intervals are sparse, we devise a dynamic program that processes them in the order of their amount of slack and guesses their tasks step by step. We use the slack in order to be able to "forget" some of the previously guessed tasks which yields a DP with only polynomially many cells.

Unfortunately, in an instance there can be dense *and* sparse intervals and our algorithms above for the two special cases are completely incompatible. Therefore, we identify a type of tasks in OPT such that we can guess tasks that cover as much as those tasks, while losing only a factor of $1 + \epsilon$. Using some charging arguments, we show that then we can split the remaining problem into two independent subinstances, one with only dense intervals and one with only sparse intervals which we then solve with the algorithms mentioned above.

▶ **Theorem 3.** *There is a parameterized approximation scheme for UFP-cover.*

Our algorithms for resource augmentation, a bounded number of task sizes, and the FPT-2-approximation even work in the weighted case, at the expense of a factor $1 + \epsilon$ in the approximation ratio. Due to space constraints the details of this and many proofs are deferred to the full version of the paper.

1.2 Other related work

The study of parameterized approximation algorithms was initiated independently by Cai and Huang [10], Chen, Grohe, and Grüber [14], and Downey, Fellows, and McCartin [18]. A good survey on the topic was given by Marx [26]. Recently, the notion of approximate kernels was introduced [25]. Independently, Bazgan [7] and Cesati and Trevisan [12] established an interesting connection between approximation algorithms and parameterized complexity by showing that EPTASs, i.e., $(1+\epsilon)$-approximation algorithms with running time $f(\epsilon)n^{O(1)}$, imply FPT algorithms for the decision version. Hence a W[1]-hardness result for a problem makes the existence of an EPTAS for it unlikely.

For the unweighted case of UFP (packing) a PAS is known [27]. Note that in the FPT setting UFP is easier than UFP-cover since we can easily make the following simplifying assumptions that we cannot make in UFP-cover. First, we can assume that the input tasks are not too small: if there are k input tasks whose size is smaller than $1/k$ times the capacity

of any the edges they use, then we can simple output those tasks and we are done; otherwise we can enumerate over them and only large tasks remain. Second, the tasks are not too big since the size of a task can be assumed to be at most the minimum capacity of an edge in its path. Third, we can easily find a set of at most k edges that together intersect the path of each input task (i.e., a hitting set for the input task's paths) unless a simple greedy algorithm finds a solution of size k [27]. The best known polynomial time approximation algorithm for UFP has a ratio of $5/3 + \epsilon$ [22] and the problem admits a QPTAS [3, 6].

Recently, polynomial time approximation algorithms for special cases of UFP-cover under resource augmentation were found: an algorithm computing a solution of optimal cost if $p_i = c_i$ for each task i and a $(1 + \epsilon)$-approximation if the cost of each task equals its "area", i.e., the product of p_i and the length of $P(i)$ [17]. UFP-cover is a special case of the general scheduling problem (GSP) on one machine in the absence of release dates. The best known polynomial time result for GSP is a $(4 + \epsilon)$-approximation [15] and a QPTAS for quasi-polynomial bounded input data [2]. Also, UFP-cover is a special case of the capacitated set cover problem, e.g., [13, 4].

2 Few different task sizes

In this section, we show that UFP-cover is FPT when it is parameterized by $k + k'$ where k' is the number of different task sizes. We are given two parameters k and k' and assume $|\{p_i : i \in T\}| = k'$. We seek to compute a solution $T' \subseteq T$ with $|T'| \leq k$ such that for each edge e it holds that $p(T' \cap T_e) := \sum_{i \in T' \cap T_e} p_i \geq u_e$ or assert that there is no such solution. Denote by $T^{(\ell)}$ for $\ell = 1, \ldots, k'$ the partition of the set T into sets of tasks with equal size.

Our algorithm sweeps the path from left to right and guesses the tasks in OPT step by step (in contrast to similar such algorithms it is *not* a dynamic program). We maintain a set T' of previously selected tasks and a pointer indicating an edge e. We initialize the algorithm with e being the leftmost edge of E and $T' := \emptyset$. Suppose that the pointer is at some edge e. If the tasks in T' already cover the demand of e, i.e., $p(T_e \cap T') \geq u_e$, then we move the pointer to the edge on the right of e. Otherwise, in OPT the edge e must be covered by a task that is not in T'. For each group $T^{(\ell)}$ we guess the number of tasks using e that are missing in T' compared to OPT, i.e., we guess $k_\ell := |T_e \cap OPT \cap T^{(\ell)}| - |T_e \cap T' \cap T^{(\ell)}|$. Since there are at most k' groups $T^{(\ell)}$, the number of possible guesses is bounded by $(k+1)^{k'}$. For each group $T^{(\ell)}$ we add to T' the k_ℓ tasks in $(T_e \cap T^{(\ell)}) \setminus T'$ with rightmost endvertex. Then we move the pointer to the edge on the right of e. Overall, we want to select at most k tasks. Therefore, at each guessing step, we enumerate only guesses that ensure that we do not select more than k tasks altogether. Hence, the total number of possible guesses overall is bounded by $((k+1)^{k'})^k = k^{O(k'k)}$. Each of them yields a set T'. In case that the resulting set T' is not a feasible solution we reject the guesses that lead to T'.

Assume from now on that all guesses were correct. In the next lemma we show that then we obtain a feasible solution. The intuition for the proof is as follows: suppose that the pointer is at some edge e and we select additional tasks from a group $T^{(\ell)}$. These additional tasks were not necessary in order to cover the demands of the edges on the left of e. All tasks in $T^{(\ell)}$ have exactly the same size. Therefore, the best choice is to select the tasks in $T_e \cap T^{(\ell)}$ with rightmost endvertices.

▶ **Lemma 4.** *Assume that the given instance has a solution of size at most k. Then the resulting set T' satisfies $\sum_{i \in T' \cap T_e} p_i \geq u_e$ for each edge e.*

The total number of guesses is bounded by $k^{O(k'k)}$ and for each set of guesses we can compute the corresponding set T' in time $n^{O(1)}$. Hence, we obtain Theorem 2.

3 Resource augmentation

In this section, we turn to the case where we have resource augmentation but the number of different task sizes is arbitrary. As a consequence of Theorem 2, we first show that UFP-cover with $(1+\delta)$ resource augmentation, can be solved in time $f(k,\delta)\cdot n^{O(1)}$ if the edge demands come in a polynomial range. In Section 3.1 we generalize this algorithm to arbitrary edge demands.

For now, we assume that the edge demands come in a polynomial range. Then, for two parameters $k \in \mathbb{N}$ and $\delta > 0$ we seek to compute a solution $T' \subseteq T$ with $|T'| \leq k$ such that for each edge e it holds that $p(T' \cap T_e) := \sum_{i \in T' \cap T_e} p_i \geq \tilde{u}_e := u_e/(1+\delta)$ or assert that there is no solution $T' \subseteq T$ with $|T'| \leq k$ such that for each edge e it holds that $p(T' \cap T_e) \geq u_e$.

The idea is to round the task sizes and then use our algorithm for bounded number of task sizes. Let OPT denote a solution with at most k tasks. We group the tasks into groups such that the sizes of the tasks in the same group differ by at most a factor of $1+\delta$. For each $\ell \in \mathbb{N}$ we define the group $T^{(\ell)} := \{i \in T | p_i \in [(1+\delta)^\ell, (1+\delta)^{\ell+1})\}$. For each ℓ we round the sizes of the tasks in $T^{(\ell)}$ to $(1+\delta)^\ell$, i.e., for each $i \in T^{(\ell)}$ we define its rounded size to be $\tilde{p}_i := (1+\delta)^\ell$ (for convenience, we allow rounded the task sizes and edge demands to be fractional). As we show in the next lemma, this rounding step is justified due to our resource augmentation.

▶ **Lemma 5.** *By decreasing the demand of each edge e to $\tilde{u}_e := u_e/(1+\delta)$ we can assume for each $\ell \in \mathbb{N}$ that each task $i \in T^{(\ell)}$ has a size of $\tilde{p}_i = (1+\delta)^\ell$, i.e., for each edge e it holds that $\sum_{i \in OPT \cap T_e} \tilde{p}_i \geq \tilde{u}_e$.*

Note that w.l.o.g. we can assume that $p_i \leq \max_e u_e$ for each task i. Since we assumed that the edge demands are bounded by a polynomial in n, there are only $O(\log_{1+\delta} n)$ groups $T^{(\ell)}$ with $T^{(\ell)} \neq \emptyset$. The optimal solution contains tasks from at most k of these groups. We guess the groups $T^{(\ell)}$ that satisfy that $OPT \cap T^{(\ell)} \neq \emptyset$ in time $\binom{O(\log_{1+\delta} n)}{k} = (\frac{1}{\delta}\log n)^{O(k)}$. Note that the latter quantity is of the form $f(k,\delta)\cdot n^{O(1)}$, since $(\log n)^{O(k)} \leq n + k^{O(k)}$. We delete the tasks from all other groups. This yields an instance with at most k different (rounded) task sizes, and then we can apply Theorem 2 with $k' = k$. Hence, there is an algorithm with running time $(\frac{1}{\delta}\log n)^{O(k)} \cdot k^{O(k^2)} \cdot n^{O(1)} = f(k,\delta)\cdot n^{O(1)}$ if the edge demands are in a polynomial range.

3.1 Arbitrary demands

We extend the above algorithm now to the case of arbitrary demands. To this end, we start with a shifting step that intuitively partitions the groups above into supergroups such that the sizes of two tasks in different supergroups differ by at least a factor of $2k/\delta$. In particular, one task from one supergroup will be larger than any k tasks from supergroups with smaller tasks together. We define K to be the smallest integer such that $k(1+\delta)^{-K-1} < \delta/2$, i.e., $K = O(\frac{1}{\delta}\log k)$. Let $\alpha \in \{0, ..., k\}$ be an offset to be defined later. Intuitively, we remove an $\frac{1}{k+1}$-fraction of all groups $T^{(\ell)}$ and combine the remaining groups into supergroups. With a shifting argument we ensure that no task from OPT is contained in a deleted group. Formally, we define a supergroup $\mathcal{T}^{(s)} := \bigcup_{\ell=K(\alpha+s(k+1))}^{K(\alpha+(s+1)(k+1)-1)-1} T^{(\ell)}$ for each integer s. In particular, each supergroup contains $K \cdot k$ groups.

▶ **Lemma 6.** *There exists an offset $\alpha \in \{0, ..., k\}$ such that for each task $i \in OPT$ there is a supergroup $\mathcal{T}^{(s)}$ such that $i \in \mathcal{T}^{(s)}$.*

We guess the value α due to Lemma 6. If the edge demands are no necessarily polynomially bounded we can no longer guess the groups that contains tasks from OPT since there can be up to $\Omega(n)$ groups. Instead, for each edge e we define a level s_e to be the largest value s such that $(1+\delta)^{K(\alpha+s(k+1))} \leq \hat{u}_e := \tilde{u}_e/(1+\delta)$. Note that $(1+\delta)^{K(\alpha+s(k+1))}$ is a lower bound on the size of each task in $\mathcal{T}^{(s)}$. In the next lemma, using resource augmentation we prove that for each edge e it holds that the tasks in $\bigcup_{\ell'} \mathcal{T}^{(s_e)} \cap OPT$ are sufficient to cover the demand of e or that in OPT the edge e is completely covered by one task in a supergroup $\mathcal{T}^{(s')}$ with $s' > s_e$.

▶ **Lemma 7.** *For each edge e it holds that $p(OPT \cap \mathcal{T}^{(s_e)} \cap T_e) \geq \hat{u}_e$ or that there is a task $i \in OPT \cap \bigcup_{s=s_e+1}^{\infty} \mathcal{T}^{(s)} \cap T_e$. In the latter case it holds that $p_i \geq \tilde{p}_i \geq \hat{u}_e$.*

In order to solve our problem, we define a set of subproblems that we solve via dynamic programming. Let us denote $\mathcal{T}^{(\geq s)} := \bigcup_{s' \geq s} \mathcal{T}^{(s')}$. Each subproblem is characterized by a subpath $\tilde{E} \subseteq E$, and integers $\tilde{k}, \tilde{s}$ with $0 \leq \tilde{k} \leq k$ and $\tilde{s} \in \{-1, ..., O(\log \max_e u_e)\}$. A tuple $(\tilde{E}, \tilde{k}, \tilde{s})$ represents the following subproblem: select a set of tasks $T' \subseteq \mathcal{T}^{(\geq \tilde{s})}$ with $|T'| \leq \tilde{k}$ such that for each edge $e \in \tilde{E}$ it holds that $\sum_{i \in T' \cap T_e} \tilde{p}_i \geq \hat{u}_e$. Note that the subproblem $(E, k, -1)$ corresponds to the original problem that we want to solve. Moreover, the number of DP-cells is polynomial in the input length. Notice that there are only a polynomial number of values $\tilde{s}$ for which $\mathcal{T}^{(\tilde{s})}$ is non-empty.

Suppose we are given a subproblem $(\tilde{E}, \tilde{k}, \tilde{s})$ and assume that we already solved each subproblem of the form $(\tilde{E}', \tilde{k}', \tilde{s}')$ where $\tilde{E}' \subseteq \tilde{E}$, $\tilde{k}' \leq \tilde{k}$, and $\tilde{s}' > \tilde{s}$. Assume that in OPT each edge $e \in \tilde{E}$ is covered by at least one task in $\mathcal{T}^{(\geq \tilde{s})}$ (it will turn out that we need to compute a feasible solution to $(\tilde{E}, \tilde{k}, \tilde{s})$ only in this case). Hence, for covering the reduced edge demands $\hat{u}$ we do not need the tasks in supergroups $\mathcal{T}^{(s')}$ with $s' < \tilde{s}$. Our algorithm sweeps the path from left to right and guesses the tasks in OPT step by step (where OPT denotes the optimal solution to the original input instance). We maintain a pointer at some edge e and a set T' of previously selected tasks. We initialize the algorithm with e being the leftmost edge of $\tilde{E}$ and $T' := \emptyset$. Suppose that the pointer is at some edge e. If the tasks in T' already cover the reduced demand of e, i.e., $p(T_e \cap T') \geq \hat{u}_e$, then we move the pointer to the edge on the right of e. Otherwise, in OPT the edge e must be covered by a task that is not in T'. We guess whether $p(OPT \cap \mathcal{T}^{(\geq \tilde{s}+1)} \cap T_e) \geq \hat{u}_e$ or $p(OPT \cap \mathcal{T}^{(\tilde{s})} \cap T_e) \geq \hat{u}_e$. Since we assumed that e is covered by a task in $\mathcal{T}^{(\geq \tilde{s})}$, Lemma 7 yields that one of these two cases applies.

Suppose we guessed that $p(OPT \cap \mathcal{T}^{(\geq \tilde{s}+1)} \cap T_e) \geq \hat{u}_e$. For any two edges e_1, e_2 denote by P_{e_1,e_2} the subpath of E starting with e_1 and ending with e_2 (including e_1 and e_2). Let e' be the rightmost edge on the right of e such that each edge $e'' \in P_{e,e'}$ the set $OPT \cap T_{e''}$ contains at least one task in $\mathcal{T}^{(\geq \tilde{s}+1)}$. Let $\tilde{k}'$ denote the number of tasks in $OPT \cap \mathcal{T}^{(\geq \tilde{s}+1)}$ whose path intersects $P_{e,e'}$. We guess $\tilde{k}'$. Then we determine the rightmost edge e'' such that $(P_{e,e''}, \tilde{k}', \tilde{s}')$ is a yes-instance, where $\tilde{s}' \geq \tilde{s}+1$ is the smallest integer such that $\mathcal{T}^{(\tilde{s}')} \neq \emptyset$. We add to T' the tasks in the solution of $(P_{e,e''}, \tilde{k}', \tilde{s}')$ and move the pointer to the edge on the right of e''.

Assume now that we guessed that $p(OPT \cap \mathcal{T}^{(\tilde{s})} \cap T_e) \geq \hat{u}_e$. Observe that $\mathcal{T}^{(\tilde{s})}$ consists of only Kk non-empty groups $T^{(\ell)}$. For each of these groups $T^{(\ell)}$ we guess $k_\ell := |OPT \cap T_e \cap T^{(\ell)}| - |T' \cap T_e \cap T^{(\ell)}|$. Note that there are only $(k+1)^{Kk}$ possible guesses. For each group $T^{(\ell)}$ we add to T' the k_ℓ tasks in $(T_e \cap T^{(\ell)}) \setminus T'$ with rightmost endvertex. Then we move the pointer to the edge on the right of e.

Like before, at each guessing step, we enumerate only guesses that ensure that we do not select more than $\tilde{k}$ tasks altogether. Hence, the total number of possible guesses overall is bounded by $2^{\tilde{k}}(\tilde{k}+1)^{O(K\tilde{k})} = k^{O(\frac{k}{\delta} \log k)}$. We store in the cell $(\tilde{E}, \tilde{k}, \tilde{s})$ the set T' of minimum

size if a set of size $\tilde{k}$ was found. Finally, we output the solution in the cell $(E, k, -1)$ if it contains a feasible solution. If it does not contain a feasible solution we output that there is no solution of size k for the original edge capacities u. Theorem 1 follows.

4 FPT-2-approximation algorithm

We present an FPT-2-approximation algorithm *without* resource augmentation (for arbitrary edge demands), i.e., an algorithm that runs in time $f(k)n^{O(1)}$ and finds a solution of size at most $2k$ or asserts that there is no solution of size at most k. Suppose we are given an instance (I, k). First, we call the algorithm for resource augmentation from Section 3 with $\delta = 1$. If this algorithm asserts that there is no solution of size at most k then we stop. Otherwise, let ALG denote the found solution. We guess $ALG \cap OPT$. Note that there are only 2^k possibilities for $ALG \cap OPT$. If $ALG \cap OPT = \emptyset$ then the solution $OPT \cup ALG$ covers each edge e to an extent of at least $3/2 \cdot u_e$, i.e., $p((OPT \cup ALG) \cap T_e) \geq 3/2 \cdot u_e$. Therefore, we create a new UFP-cover instance I' whose input tasks are identical with the tasks in I and in which the demand of each edge e is changed to $u'_e := 3/2 \cdot u_e$. We invoke our algorithm for the resource augmentation setting from Section 3 to I' where we look for a solution of size at most $|ALG| + k \leq 2k$ and we set $\delta := 1/2$. Let ALG' be the returned solution. It holds that $|ALG'| \leq |ALG| + k \leq 2k$ and ALG' covers each edge e to an extent of at least $3/2 \cdot u_e/(1 + 1/2) = u_e$. We output ALG'.

If $ALG \cap OPT \neq \emptyset$ then we generate a new instance I'' in which the tasks in $ALG \cap OPT$ are already taken, i.e., the demand of each edge e is reduced to $u''_e := u_e - p(ALG \cap OPT \cap T_e)$ and the set of input tasks consists of $T \setminus (ALG \cap OPT)$. We recurse on I'' where the parameter k is set to $k - |ALG \cap OPT|$. Observe that $OPT'' := OPT \setminus ALG$ is a solution to I'' and if $|OPT| \leq k$ then $|OPT''| \leq k - |ALG \cap OPT|$. The resulting recursion tree has depth at most k with at most 2^k children per node and hence it has at most $2^{O(k^2)}$ nodes in total. This yields the following theorem.

▶ **Theorem 8.** *There is an algorithm for UFP-cover with a running time of $2^{O(k^2)} \cdot n^{O(1)}$ that either finds a solution of size at most $2k$ or asserts that there is no solution of size k.*

5 Parameterized approximation scheme

In this section, we present a PAS for UFP-cover. Given a parameter k, we seek to compute a solution of size at most $(1 + \epsilon)k$ or assert that there is no solution of size at most k. The running time of our algorithm is $k^{O(k)} n^{(1/\epsilon)^{O(1/\epsilon)}}$. Let OPT denote a solution with at most k tasks and let $\epsilon > 0$ such that $1/\epsilon \in \mathbb{N}$.

We describe first how we guess a partition of $E = I_0 \dot\cup I_1 \dot\cup \cdots \dot\cup I_r$ into $O(k)$ many subpaths that we denote as *intervals*. Also, we will guess a set of tasks $T_S \subseteq T$ such that if we add T_S to OPT then we obtain a certain amount of slack that we will use in the computation later. If $T_S \cap OPT \neq \emptyset$ then we will simply guess $T_S \cap OPT$ and recurse, *without* losing anything in the approximation ratio.

We group the tasks into groups such that the tasks in the same group have the same size, up to a factor $1+\epsilon$. Formally, for each integer ℓ we define $T^\ell := \{i \in T, p_i \in [(1+\epsilon)^\ell, (1+\epsilon)^{\ell+1})\}$ and we say that a task i is *of level* ℓ if $i \in T^\ell$. Then we run the 4-approximation algorithm from [5] to obtain a solution S. If $|S| > 4k$ then $OPT > k$ and we stop. For each edge e let $OPT_e^{1/\epsilon}$ denote the $1/\epsilon$ largest tasks in $OPT \cap T_e$ (breaking ties in an arbitrary fixed way). Intuitively, we would like to select the tasks OPT_{SL} due to the following lemma from [6, Lemma 3.1].

▶ **Lemma 9** ([6]). *There is a set $OPT_{SL} \subseteq OPT$ with $|OPT_{SL}| \leq \gamma\epsilon|OPT|$ such that for each edge e with $|OPT \cap T_e| \geq 1/\epsilon$ it holds that $OPT_{SL} \cap OPT_e^{1/\epsilon} \neq \emptyset$, where γ is a universal constant that is independent of the given instance.*

We cannot guess OPT_{SL} directly. Instead, we run the following algorithm that computes a set $T_S = T_S^{(1)} \cup T_S^{(2)} \cup T_S^{(3)}$ with at most $O(|OPT_{SL}|)$ tasks that gives us similar slack as OPT_{SL}. The reader may imagine that $T_S^{(1)} \cup T_S^{(2)} = OPT_{SL}$ and that $T_S^{(3)}$ are additional tasks that we select. We initialize $T_S^{(1)} = \emptyset$. Let $\tilde{V}$ be the set of start and end vertices of the tasks in S. We partition E according to the vertices in $\tilde{V}$. Formally, we consider the partition $E = \tilde{I}_1 \dot\cup \cdots \dot\cup \tilde{I}_{\tilde{r}}$ of E such that for each $\tilde{I}_j = \{v_1, ..., v_s\}$ we have that $v_1 \in \tilde{V}$ and $v_s \in \tilde{V}$ and for each $s' \in \{2, ..., s-1\}$ we have that $v_{s'} \notin \tilde{V}$. We say that a task i starts in an interval $\tilde{I}_j$ if $\tilde{I}_j$ is the leftmost interval that contains an edge of $P(i)$ and a task i ends in an interval $\tilde{I}_j$ if $\tilde{I}_j$ is the rightmost interval that contains an edge of $P(i)$. For each pair of intervals $\tilde{I}_j, \tilde{I}_{j'}$ we guess whether there is a task from OPT_{SL} that starts in $\tilde{I}_j$ and ends in $\tilde{I}_{j'}$. If yes, we add to $T_S^{(1)}$ the largest task with this property. Additionally, for each interval $\tilde{I}_j$ in which at least one task from OPT_{SL} starts or ends we add to $T_S^{(1)}$ the largest task $i^* \in T$ such that $\tilde{I}_j \subseteq P(i^*)$ and define $\bar{s}_j := p_{i^*}$. One can show that the maximum demand of an edge $e \in \tilde{I}_j$ is upper-bounded by $4k\bar{s}_j$ since i^* is at least as large as the largest task $i \in S$ with $\tilde{I}_j \subseteq P(i)$ and each task $i \in S$ starts or ends at a vertex in $\tilde{V}$. On the other hand, $\bar{s}_j$ is as large as k tasks of size at most $\frac{1}{k}\bar{s}_j$ together and hence we can ignore tasks of the latter kind for $\tilde{I}_j$ if we have $\bar{s}_j$ units of slack in $\tilde{I}_j$. Let L denote the set of values ℓ such that there is an interval $\tilde{I}_j$ and a task $i \in T^\ell$ with $p_i \in [\frac{1}{2k}\bar{s}_j, 4k\bar{s}_j]$. One can show that $|L| \leq O_\epsilon(k \log k)$ and $|T_S^{(1)}| \leq O(\epsilon k)$.

Next, we define a set $T_S^{(2)}$ of additional slack tasks. We maintain a queue $Q \subseteq V$ of vertices that we call *interesting* and a set of tasks $T_S^{(2)}$. At the beginning, we initialize $T_S^{(2)} := \emptyset$ and $Q := \tilde{V}$. In each iteration we extract an arbitrary vertex v from Q. Let Q' be the set of vertices that were removed from the queue Q in an earlier iteration. For each vertex v let T_v denote the input tasks i whose path $P(i)$ uses v, i.e., such that $P(i)$ contains an edge e incident to v. For each group T^ℓ with $\ell \in L$ we guess whether there is a task in OPT_{SL} that uses v but that does not use any vertex in Q', i.e., we guess whether there is a task in $OPT_{SL} \cap T^\ell \cap T_v \setminus \bigcup_{v' \in Q'} T_{v'}$. We add to $T_S^{(2)}$ the task with leftmost startvertex and the task with rightmost endvertex from $T^\ell \cap T_v \setminus \bigcup_{v' \in Q'} T_{v'}$. For each added task, we add its start- and its endvertex to Q if it has not been in Q before. The algorithm terminates once Q is empty.

Let $T_S^{(2)}$ be the resulting set. One can show that $|T_S^{(2)}| \leq O(\epsilon k)$. Let now V' be the set of start- and endvertices of tasks in $S \cup T_S^{(1)} \cup T_S^{(2)}$ and let $I_0 \cup I_1 \cup \cdots \cup I_r = E$ be the partition into subpaths defined by the vertices in V'. In the following, we partition intervals into three groups according to the number of tasks from OPT that start or end in them. Given an interval I let d be the number of tasks that start or end in I. Let α be some constant in $\{5, \cdots, 5/\epsilon\}$. We say that I is *sparse* if $d \leq 1/\epsilon^\alpha$, *medium* if $1/\epsilon^\alpha < d \leq 1/\epsilon^{\alpha+5}$ and *dense* if $d > 1/\epsilon^{\alpha+5}$.

▶ **Lemma 10.** *There exists an integer $\alpha \in \{5, \ldots, 5/\epsilon\}$ such that the number of tasks in OPT that start or end in a medium interval is at most $2\epsilon k$.*

We guess α and for each interval I_j we guess whether it is sparse, medium, or dense. Note that there are in total $\frac{5}{\epsilon}3^{O(k)}$ many guesses. We select now some more tasks that will provide us with additional slack. For each medium or dense interval I_j we select the largest task $i \in T$ such that $I_j \subseteq P(i)$. Also, for each maximal set of contiguous sparse intervals

$I_j \cup I_{j+1} \cup \cdots \cup I_{j'} =: \mathcal{I}$ we select the largest task $i \in T$ such that $\mathcal{I} \subseteq P(i)$. Let $T_S^{(3)}$ denote the resulting set. We have that $|T_S^{(3)}|$ is at most twice the total number of intervals that are medium or dense. In each of the latter intervals there are at least $1/\epsilon^5$ tasks from OPT that start or end. Therefore, $|T_S^{(3)}| \leq \epsilon k$. We call $T_S := T_S^{(1)} \cup T_S^{(2)} \cup T_S^{(3)}$ the *slack tasks*. For each interval I_j we denote by $\hat{s}_j$ the slack in the interval given by T_S, i.e., $\hat{s}_j = \min_{e \in I_j} p\left(T_e \cap T_S\right)$. To summarize, we obtained the following properties of our intervals and T_S.

▶ **Lemma 11.** *We have that $I_0 \dot\cup I_1 \dot\cup \cdots \dot\cup I_r$ is a partition of E into $O(k)$ intervals and $|T_S| \leq O(\epsilon k)$. For each edge e that is the leftmost or the rightmost edge of an interval I_j we have that there are at most $1/\epsilon$ tasks $i' \in OPT \cap T_e$ such that $\hat{s}_j(1+\epsilon) < p_{i'} < 4k\hat{s}_j$. For each dense interval I_j we have that $\hat{s}_j \geq \frac{1}{4k} \max_{e \in I_j} u_e$. Also, for each maximal set of contiguous sparse intervals $I_j \cup I_{j+1} \cup \cdots \cup I_{j'} =: \mathcal{I}$ we have that $\min_{j'' : j \leq j'' \leq j'} \hat{s}_{j''}$ is at least size of the largest task $i \in OPT$ with $\mathcal{I} \subseteq P(i)$ (if OPT contains such a task i).*

If the tasks in T_S are not contained in OPT then $OPT \cup T_S$ is a solution of size $(1+O(\epsilon))k$ in which each edge has some slack and hence we can use this slack algorithmically. Otherwise, we recurse: We guess $OPT \cap T_S$ and if $OPT \cap T_S \neq \emptyset$ then we recurse on a new instance where we assume that $OPT \cap T_S$ is already selected. Formally, this instance has input task $\bar{T} := T \setminus (OPT \cap T_S)$, each edge $e \in E$ has demand $\bar{u}_e := u_e - \sum_{i \in T_e \cap (OPT \cap T_S)} p_i$, and the parameter is $\bar{k} := k - |OPT \cap T_S|$. Together with the guesses above, this yields $k^{O(k^2)}$ many guesses. Hence, the recursion tree has depth at most k and each internal node has at most $k^{O(k^2)}$ children which yields $k^{O(k^3)}$ vertices in total. In the sequel, we will assume that $OPT \cap T_S = \emptyset$ and solve the remaining problem without any further recursion in time $f(k)n^{O(1)}$ for some function f.

5.1 Medium intervals

We describe a routine that essentially allows us to reduce the problem to the case where there are no medium intervals. From Lemma 10 we know that there are no more than $2\epsilon k$ tasks in OPT that start or end in a medium interval. Therefore, for those tasks we can afford to make mistakes that cost us a constant factor, i.e., we can select $O(\epsilon k)$ instead of $2\epsilon k$ of those tasks.

Let $T_{\text{med}} \subseteq T$ be the set of tasks that start or end in a medium interval. Let I_j be a medium interval. In OPT, the demand of the edges in I_j is partially covered by tasks $i \in OPT \setminus T_{\text{med}}$ that completely cross I_j, i.e., such that $I_j \subseteq P(i)$. We guess an estimate for the total size of such tasks, i.e., an estimate for $\hat{p}_j = p\left(\{i \in OPT \setminus T_{\text{med}} : I_j \subseteq P(i)\}\right)$. Formally, we guess $\hat{u}_j := \lfloor \hat{p}_j / (\hat{s}_j/3) \rfloor$.

▶ **Lemma 12.** *We have that $\hat{u}_j \in \{0, ..., 3k\}$ and for each edge $e \in I_j$ it holds that $p(T_e \cap OPT \cap T_{\text{med}}) + \hat{u}_j \hat{s}_j/3 + p(T_e \cap T_S) \geq u_e$.*

Since there are only $O(k)$ intervals, there are only $k^{O(k)}$ many guesses in total. We construct an auxiliary instance on the same graph $G = (V, E)$ with input tasks T_{med} and demand $u_e^{\text{med}} = \max\{u_e - p(T_e \cap (T_S)) - \hat{u}_\ell \hat{s}_\ell/3, 0\}$ for each $e \in E$ in a medium interval, and $u_e^{\text{med}} = 0$ for each edge $e \in E$ in a sparse or dense interval. We run the 4-approximation algorithm [5] on this instance, obtaining a set of tasks $T'_{\text{med}} \subseteq T_{\text{med}}$ with $|T'_{\text{med}}| \leq 4|OPT \cap T_{\text{med}}| \leq O(\epsilon k)$.

For our remaining computation for each medium interval I_j we define the demand u_e of each edge $e \in I_j$ to be $u_e := \hat{u}_j \hat{s}_j/3$. Lemma 12 implies that any solution T' for this changed instance yields a solution $T' \cup T'_{\text{med}} \cup T_S$ with at most $|T'| + O(\epsilon k)$ tasks for the original instance. In the sequel, denote by OPT' the optimal solution to the new instance.

5.2 Heavy vertices

Our strategy is to decouple the sparse and dense intervals. A key problem is that there are tasks $i \in OPT'$ such that $P(i)$ contains edges in sparse and in dense intervals. Intuitively, our first step is therefore to guess some of them in an approximate way.

There are vertices $v \in V'$ that are used by many tasks in $OPT' \cap T^\ell$ for some level ℓ. Formally, we say that for a set of tasks $T' \subseteq T$ a vertex $v \in V'$ is (ℓ, T')-*heavy* if there are more than $1/\epsilon^{\alpha+1}$ tasks $i \in T' \cap T^\ell \cap T_v$ such that i starts or ends in a sparse or a medium interval. We are interested in vertices $v \in V'$ that are (ℓ, OPT')-heavy for some ℓ. It turns out that we can compute a small number of levels ℓ for which this can happen based on the slacks $\hat{s}_j$ of the intervals I_j.

▶ **Lemma 13.** *Let $v \in V'$ and assume that v is (ℓ, OPT')-heavy for some $\ell \in \mathbb{N}_0$. Then $(1+\varepsilon)^\ell \in [\frac{1}{2k}\hat{s}_j, 4k \cdot \hat{s}_j]$ for some interval I_j.*

Therefore, let L denote the set of levels ℓ such that a vertex $v \in V'$ can be (ℓ, OPT')-heavy according to Lemma 13, i.e., $L := \{\ell | \exists j : (1+\varepsilon)^\ell \in [\frac{1}{2k}\hat{s}_j, 4k \cdot \hat{s}_j]\}$. Intuitively, for each level $\ell \in L$ and each (ℓ, OPT')-heavy vertex $v \in V'$ we want to select a set of tasks $\bar{T}_{\ell,v} \subseteq T^\ell \cap T_v$ that together cover as much as the tasks in OPT' due to which v is (ℓ, OPT')-heavy, i.e., the tasks in $OPT' \cap T^\ell \cap T_v$.

To this end, we do the following operation for each level $\ell \in L$. We perform several iterations. We describe now one iteration and assume that $v' \in V'$ is the vertex that we processed in the previous iteration (at the first iteration v' is undefined and let $T_{v'} := \emptyset$ in this case). We guess the leftmost vertex $v \in V'$ on the right of v' that is $(\ell, OPT' \setminus T_{v'})$-heavy. Let $OPT'_{\ell,v} := OPT' \cap T^\ell \cap T_v \setminus T_{v'}$. We want to compute a set $\bar{T}_{\ell,v}$ that is not much bigger than $OPT'_{\ell,v}$, i.e., $|\bar{T}_{\ell,v}| \leq (1 + O(\epsilon))|OPT'_{\ell,v}|$ and that covers at least as much on each edge e as $OPT'_{\ell,v}$, i.e., $p(T_e \cap \bar{T}_{\ell,v}) \geq p(T_e \cap OPT'_{\ell,v})$. We initialize $\bar{T}_{\ell,v} := \emptyset$. We consider each pair of intervals I_j and $I_{j'}$ such that all edges of I_j are on the left of v (but might have v as an endpoint) and all edges of $I_{j'}$ are on the right of v (but might have v as an endpoint) and such that I_j or $I_{j'}$ is sparse or medium. We guess the number $k^{v,\ell}_{j,j'}$ of tasks from $OPT' \cap T^\ell \cap T_v \setminus T_{v'}$ that start in I_j and end in $I_{j'}$ (and hence are contained in T_v). If I_j is sparse or medium (and hence then $I_{j'}$ can be anything), we add to $\bar{T}_{\ell,v}$ the $k^{v,\ell}_{j,j'}$ tasks from $T^\ell \setminus T_{v'}$ with rightmost endvertex that start in I_j and end in $I_{j'}$. If I_j is dense (and hence then $I_{j'}$ is sparse or medium) we add to $\bar{T}_{\ell,v}$ the $k^{v,\ell}_{j,j'}$ tasks from $T^\ell \setminus T_{v'}$ with leftmost startvertex that start in I_j and end in $I_{j'}$. Note that $\sum_{j,j'} k^{v,\ell}_{j,j'} = |OPT'_{\ell,v}|$. Intuitively, the tasks in $\bar{T}_{\ell,v}$ cover each edge of E to a similar extent as the tasks in $OPT'_{\ell,v}$. We will show that the difference is compensated by additionally adding the following tasks to $\bar{T}_{\ell,v}$: we add to $\bar{T}_{\ell,v}$ the $\lfloor 2/\epsilon^\alpha + 2\epsilon|OPT'_{\ell,v}| \rfloor$ tasks from $T^\ell \cap T_v \setminus (\bar{T}_{\ell,v} \cup T_{v'})$ with leftmost start vertex. After this, we add to $\bar{T}^\ell_v$ the $\lfloor 2/\epsilon^\alpha + 2\epsilon|OPT'_{\ell,v}| \rfloor$ tasks from $T^\ell \cap T_v \setminus (\bar{T}_{\ell,v} \cup T_{v'})$ with rightmost end vertex. Let $\bar{T}_{\ell,v}$ denote the resulting set. We prove that it covers as much as $OPT'_{\ell,v}$ and that it is not much bigger than $|OPT'_{\ell,v}|$.

▶ **Lemma 14.** *For each edge $e \in E$ in a dense or a sparse interval, we have that $p(T_e \cap \bar{T}_{\ell,v}) \geq p(T_e \cap OPT'_{\ell,v})$. For each edge e in a medium interval, we have that $p(T_e \cap \bar{T}_{\ell,v} \setminus T_{\text{med}}) \geq p(T_e \cap OPT'_{\ell,v} \setminus T_{\text{med}})$. Also, it holds that $|\bar{T}_{\ell,v}| \leq (1 + O(\epsilon))|OPT'_{\ell,v}|$.*

We continue with the next iteration where now v' is defined to be the vertex v from above. We continue until in some iteration there is no vertex $v \in V'$ on the right of v' that is $(\ell, OPT' \setminus T_{v'})$-heavy. Let $V'_\ell \subseteq V'$ denote all vertices that at some point were guessed as being the $(\ell, OPT' \setminus T_{v'})$-heavy vertex v above. Let $T_H := \bigcup_{\ell \in L} \bigcup_{v \in V'_\ell} \bar{T}_{\ell,v}$ denote the set of computed tasks and define $OPT'_H := \bigcup_{\ell \in L} \bigcup_{v \in V'_\ell} OPT'_{\ell,v}$.

▶ **Lemma 15.** *We have that $p(T_e \cap T_H) \geq p(T_e \cap OPT'_H)$ for each edge e in a dense and a sparse interval, and $p(T_e \cap T_H \setminus T_{\text{med}}) \geq p(T_e \cap OPT'_H \setminus T_{\text{med}})$ for each edge e in a medium interval. Also, it holds that $|T_H| \leq (1 + O(\epsilon))|OPT'_H|$.*

It remains to compute a set of tasks T' such that $T' \cup T_H \cup T_S$ is feasible. Intuitively, T' should cover as much as $OPT' \setminus OPT'_H$ on each edge. To this end, we decouple the problem into one for the dense intervals and one for the sparse intervals. One problem for this is that even after selecting the tasks in T_H we might need to select additional tasks that have one endpoint in a dense interval and the other one in a sparse interval. However, we will show that since we selected the tasks in T_S, for each dense interval I there are only constantly many such tasks that still matter. We show that we can afford to select such tasks twice (once in the subproblem for the dense intervals and once in the subproblem for the sparse intervals) since we can charge them to the many tasks that start or end in I. For this charging to work we use that in a dense interval there can be many more tasks that start or end than in a sparse interval.

5.3 Dense intervals

Recall that for each dense interval I_j we have that $\hat{s}_j \geq \frac{1}{4k} \max_{e \in I_j} u_e$ (see Lemma 11). Hence, intuitively it suffices to compute a solution for I_j that is feasible under $(1 + \frac{1}{4k})$-resource augmentation. So in order to compute a set of tasks T' that cover the remaining demand in all dense intervals I_j (after selecting T_H) we could apply the algorithm for resource augmentation from Section 3 directly as a black box. However, there are also the sparse intervals and it might be that there are tasks $i \in OPT' \setminus OPT'_H$ that are needed for a dense interval *and* for a sparse interval. There are two types of such tasks. The first type are tasks that have at least one endpoint in a sparse interval. For each dense interval I_j there can be at most $2/\epsilon^\alpha$ such tasks in $T^\ell \cap OPT' \setminus OPT'_H$ for each group ℓ (since each of them needs to overlap one of the endpoints of I_j). We will show later that the slack due to T_S is as large as almost all of them together, all apart from $1/\epsilon^{\alpha+4}$ many. Hence, if we select the latter tasks twice (once in the subproblem for the dense intervals and once in the subproblem for the sparse intervals) we can charge them to the tasks that start or end in I_j and hence we increase our cost by at most a factor $1 + \epsilon$. The second type are tasks that start and end in a dense interval. Let $T_D \subseteq T$ denote the set of all such tasks. Note that a vertex can be crossed by more than constantly many tasks in $T_D \cap OPT' \setminus OPT'_H$. To handle those tasks, we guess an estimate for the demand that such tasks cover in the sparse intervals. Therefore, for each sparse interval $I_{j'}$ we guess a value $\hat{u}_{j'}$ such that $\hat{u}_{j'} = \left\lfloor \frac{p(T_e \cap T_D \cap OPT' \setminus OPT'_H)}{\hat{s}_j/4} \right\rfloor \cdot \hat{s}_j/4$ for each edge $e \in I_{j'}$ (note that $p(T_e \cap T_D \cap OPT' \setminus OPT'_H)$ is identical for each edge $e \in I_{j'}$). Then $p(T_e \cap T_D \cap OPT' \setminus OPT'_H)$ essentially equals $\hat{u}_{j'}$ and we show that the difference is compensated by our slack, even if we cover a bit less than $\hat{u}_{j'}$ units on each edge $e \in I_{j'}$.

▶ **Lemma 16.** *Let $I_{j'}$ be a sparse interval. Then $\hat{u}_{j'} \in \{0, \frac{\hat{s}_j}{4}, 2 \cdot \frac{\hat{s}_j}{4}, \ldots, 4k \cdot \frac{\hat{s}_j}{4}\}$ and $\hat{u}_{j'} \leq p(T_e \cap T_D \cap OPT' \setminus OPT'_H) \leq \frac{1}{(1+\frac{1}{4k})} \hat{u}_{j'} + p(T_e \cap T_S)/2$ for each $e \in I_{j'}$.*

We generate now an auxiliary instance where in each sparse interval $I_{j'}$ we reduce the demand u_e of each edge $e \in I_{j'}$ to $\hat{u}_{j'}$ (but do not change the demand on any edge in a dense interval) and remove all input tasks i such that $P(i)$ does not contain an edge of a dense interval. Also, for each remaining task i we shorten its path $P(i)$ to a path $P'(i)$ such that $P'(i)$ is the longest path contained in $P(i)$ that starts and ends on a vertex in a dense interval. We apply the algorithm from Section 3 with $(1 + \delta)$-resource augmentation to this instance with $\delta := 1/4k$. We obtain a solution $T^{(1)}$ such that for each edge e in an interval I_j, the solution

$T^{(1)}$ covers at least $u_e - \hat{s}_j/2$ when I_j is dense and $\hat{u}_j - \hat{s}_j/2$ when I_j is sparse. Notice that according to Lemma 11, we have $\hat{s}_j \geq u_e/(4k)$ for each edge e in a dense interval I_j and for each edge e lying in a maximal set of contiguous sparse intervals that is completely crossed by at least one input task.

▶ **Lemma 17.** *For each edge e in a dense interval we have that $p(T_e \cap (T^{(1)} \cup T_H \cup T_S)) \geq u_e$. For each edge e in a sparse interval $I_{j'}$ we have that $p(T_e \cap T^{(1)} \cap T_D) + p(T_e \cap T_S)/2 \geq p(T_e \cap T_D \cap OPT' \setminus OPT'_H)$.*

Due to Lemma 17 we cover the complete demand in each dense interval and some portion of the demand in each sparse interval. Therefore, for the remaining problem for each edge e in a sparse interval $I_{j'}$ we change its demand to $\bar{u}_e := u_e - \hat{u}_{j'}$. Also, we remove all tasks in T_D from the input, i.e., we work with the input tasks $\bar{T} := T \setminus T_D$. We claim that $\overline{OPT} := OPT' \setminus (OPT'_H \cup T_D)$ is a solution to the residual instance.

▶ **Lemma 18.** *For each edge e in a sparse interval we have that $p(T_e \cap \overline{OPT}) \geq \bar{u}_e$.*

5.4 Sparse intervals

Recall that in each sparse interval $I_{j'}$ there are at most $1/\epsilon^\alpha$ tasks from OPT that start or end in $I_{j'}$ (and hence the same is true for $\overline{OPT} \subseteq OPT$). Therefore, for each sparse interval we can guess these tasks in time $n^{O(1/\epsilon^\alpha)}$. If it was even true that for each sparse interval $I_{j'}$ there are at most $1/\epsilon^\alpha$ tasks $i \in OPT$ with $P(i) \cap I_{j'} \neq \emptyset$ then we could devise a simple dynamic program (DP) that sweeps the intervals from left to right and computes the optimal solution. Unfortunately, this is not true, but note that each vertex $v \in V'$ is used by at most $1/\epsilon^{\alpha+1}$ tasks in $\overline{OPT} \cap T^\ell$ for each group T^ℓ. Using this, we devise a more complicated DP that processes the intervals in the order of their slacks $\hat{s}_j$ and guesses step by step the at most $1/\epsilon^\alpha$ tasks that start or end in each of them. In order to restrict the running time to a polynomial we use the tasks in T_S in order to be able to "forget" some previously guessed tasks, i.e., we argue that the forgotten tasks have a total size that is at most the size of the slack due to T_S. Let us define a constant $\beta := 1 + \log_{1+\epsilon}\left(\frac{6}{\epsilon^{\alpha+2}}\right)$ and a constant $\Gamma := 1/\epsilon^\alpha + 1/\epsilon + (\beta+2)/\epsilon^{\alpha+1}$. Formally, each DP-cell is described by

- two intervals $I_j, I_{j'}$ such that for each interval $I_{j''}$ between I_j and $I_{j'}$ it holds that $\hat{s}_{j''} \geq \max\{\hat{s}_j, \hat{s}_{j'}\}$,
- two sets of tasks T'_j and $T'_{j'}$ of size at most Γ such that for each $i \in T'_j$ (*resp.* $i \in T'_{j'}$) it holds that $P(i) \cap I_j \neq \emptyset$ (*resp.* $P(i) \cap I_{j'} \neq \emptyset$) and $p(T_e \cap T'_j) + p(T_e \cap T_S)/2 \geq \bar{u}_e$ (*resp.* $p(T_e \cap T'_{j'}) + p(T_e \cap T_S)/2 \geq \bar{u}_e$) for each edge $e \in I_j$ (*resp.* $e \in I_{j'}$), i.e., the tasks in T'_j (*resp.* $T'_{j'}$) essentially cover the demand of I_j (*resp.* $I_{j'}$).

Such a cell $(I_j, I_{j'}, T'_j, T'_{j'})$ represents the subproblem of selecting a set of tasks $\hat{T}$ such that the path of each task $i \in \hat{T}$ lies between I_j and $I_{j'}$ and does not use any edge of $I_j \cup I_{j'}$ and such that $T'_j \cup T'_{j'} \cup \hat{T}$ cover the demand $\bar{u}_e$ for each edge between I_j and $I_{j'}$ together with half of the slack, i.e., $p(T_e \cap (T'_j \cup T'_{j'} \cup \hat{T})) + p(T_e \cap T_S)/2 \geq \bar{u}_e$.

Suppose we are given a cell $(I_j, I_{j'}, T'_j, T'_{j'})$ and we want to compute a solution $DP(I_j, I_{j'}, T'_j, T'_{j'})$ for it. Let $I_{j''}$ denote the interval between I_j and $I_{j'}$ with smallest slack $\hat{s}_{j''}$ (breaking ties arbitrarily). Let ℓ'' be the greatest integer such that $(1+\epsilon)^{\ell''} \leq \hat{s}_{j''}$. Let $T^{\geq \ell''-\beta} := \bigcup_{\ell \geq \ell''-\beta} T^\ell$. The intuition is that we guess the tasks in $\overline{OPT}$ that use $I_{j''}$ and when we recurse we forget all tasks that are not in $T^{\geq \ell''-\beta}$. We will show that our slack compensates the forgotten tasks. Therefore, we can ensure that if we always guess all tasks from $\overline{OPT}$ correctly then we will have only subproblems $(I_j, I_{j'}, T'_j, T'_{j'})$ where $|T'_j| \leq \Gamma$ and $|T'_{j'}| \leq \Gamma$. Formally, we enumerate all sets of tasks $T'_{j''} \subseteq T^{\geq \ell''-\beta}$ such that there are at most Γ tasks in $T'_{j''}$,

$P(i) \cap I_{j''} \neq \emptyset$ for each $i \in T'_{j''}$, and the tasks in $T'_{j''}$ cover the demand of $I_{j''}$ together with half of the slack in T_S, i.e., $p(T_e \cap T'_{j''}) + p(T_e \cap T_S)/2 \geq \bar{u}_e$ for each edge $e \in I_{j''}$. For a fixed guess of $T'_{j''}$ we associate the solution $T'_j \cup T'_{j'} \cup T'_{j''} \cup DP(I_j, I_{j''}, T'_j, T'_{j''}) \cup DP(I_{j''}, I_{j'}, T'_{j''}, T'_{j'})$. We define $DP(I_j, I_{j'}, T'_j, T'_{j'})$ to be the solution of minimum size associated to one of the enumerated sets $T'_{j''}$. For DP-cells $(I_j, I_{j'}, T'_j, T'_{j'})$ such that there is no interval between I_j and $I_{j'}$ we define $DP(I_j, I_{j'}, T'_j, T'_{j'}) := \emptyset$. For convenience, assume that we append two dummy intervals I_{-1} and I_{r+1} on the left and on the right of E that are not used by any task and that have zero demand on each of their edges. Also, we define that they have zero slack, i.e., $\hat{s}_{-1} = \hat{s}_{r+1} = 0$. We output the solution $DP(I_{-1}, I_{r+1}, \emptyset, \emptyset)$.

In order to show that the above DP is correct, one key step is to argue that it is unproblematic to neglect the tasks that are not in $T^{\geq \ell'' - \beta}$ in each respective step. This is shown in the following lemma.

▶ **Lemma 19.** *Let $I_j, I_{j'}$ be two intervals such that for each interval $I_{j''}$ between I_j and $I_{j'}$ it holds that $\hat{s}_{j''} \geq \max\{\hat{s}_j, \hat{s}_{j'}\}$. Let ℓ'' be the greatest integer such that $(1+\epsilon)^{\ell''} \leq \hat{s}_{j''}$ for all intervals $I_{j''}$ between I_j and $I_{j'}$. Then for each edge e between I_j and $I_{j'}$ it holds that $d\left(T_e \cap \overline{OPT} \cap T^{\geq \ell'' - \beta}\right) + p(T_e \cap T_S)/2 \geq \bar{u}_e$.*

Also, we need to show that when we enumerate the sets $T'_{j''}$ above one candidate set consists of the tasks in $\overline{OPT}$ that use $I_{j''}$ but neither I_j nor $I_{j'}$ and that in particular the latter set contains at most Γ tasks.

▶ **Lemma 20.** *Let $I_{j''}$ be a sparse interval and let ℓ'' be the greatest integer such that $(1+\epsilon)^{\ell''} \leq \hat{s}_{j''}$. Then there are at most Γ tasks in $\overline{OPT} \cap T^{\geq \ell'' - \beta}$ that use an edge of $I_{j''}$.*

Equipped with Lemmas 19 and 20 we can prove that the above DP is correct by arguing that it will produce $\overline{OPT}$ if it makes the corresponding guesses for each DP-cell. Also, by construction the returned solution is feasible. This yields the following lemma.

▶ **Lemma 21.** *There is an algorithm with a running time of $n^{O(1/\epsilon^{\alpha+4})}$ that computes a set $T^{(2)} \subseteq \bar{T}$ with $|T^{(2)}| \leq |\overline{OPT}|$ and $p(T_e \cap T^{(2)}) + p(T_e \cap T_S)/2 \geq \bar{u}_e$ for each edge e.*

It remains to argue that our computed sets $T'_{\text{med}}, T^{(1)}, T^{(2)}, T_H, T_S$ together form a feasible solution and do not contain too many tasks. With the following lemma we complete the proof of Theorem 3.

▶ **Lemma 22.** *We have that $T'_{\text{med}} \cup T^{(1)} \cup T^{(2)} \cup T_H \cup T_S$ is a feasible solution to the original input instance (T, E) and $|T'_{\text{med}} \cup T^{(1)} \cup T^{(2)} \cup T_H \cup T_S| \leq (1 + O(\epsilon))k$.*

6 $W[1]$-hardness

In this section we prove that UFP-cover is $W[1]$-hard if the parameter k represents the number of tasks in the optimal solution. Our proof goes along the lines of the proof that UFP (packing) is $W[1]$-hard for the same parameter as in [27].

▶ **Theorem 23.** *UFP-cover problem is $W[1]$-hard when parameterized by the number of tasks in the optimal solution.*

We give a reduction from the k-subset sum problem which is $W[1]$-hard [19]. Given a set of n values $A = \{a_1, ..., a_n\}$, a target value B and an integer k, the goal is to choose exactly k values from A that sum up to exactly B.

Suppose we are given an instance of k-subset sum. First, we claim that we can assume w.l.o.g. some properties of it.

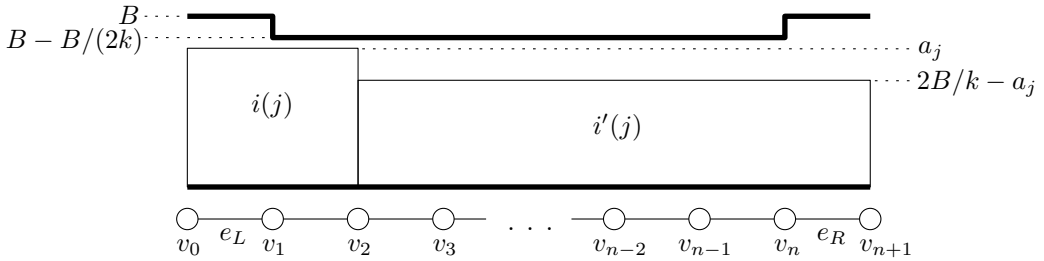

Figure 1 Sketch of the reduction used in order to prove Theorem 23. The sketch shows the tasks $i(j)$ and $i'(j)$ for only one index j. The figure is essentially identical to a figure in [27], taken with consent of the author.

▶ **Lemma 24.** *W.l.o.g. we can assume that there are values $\epsilon_1, ..., \epsilon_n$, not necessarily positive, such that $a_i = B/k + \epsilon_i$ for each $i \in [n]$ and that $\sum_{i=1}^{n} |\epsilon_i| < B/(2k)$.*

We construct an instance of UFP-cover that admits a solution with $2k$ tasks if and only if the given k-subset sum is a yes-instance. Our UFP-cover instance has a path with $n+2$ vertices $v_0, v_1, ..., v_{n+1}$. Denote the leftmost and the rightmost edge by e_L and e_R, respectively. We define $u(e_L) = u(e_R) = B$. For all other edges e we define $u(e) := B - B/(2k)$. Assume that the values in S are ordered such that $a_1 \geq a_2 \geq ... \geq a_n$. Let $j \in [n]$. We introduce two tasks $i(j), i'(j)$ with $s(i(j)) := v_0$, $t(i(j)) := v_j$, $p(i(j)) := a_j$ $s(i'(j)) := v_j$, $t(i'(j)) = v_{n+1}$, and $p(i'(j)) := 2B/k - a_j$. See Figure 1 for a sketch.

In order to get some intuition about the constructed instance, we prove the following lemma.

▶ **Lemma 25.** *Any feasible solution contains at least $2k$ tasks. Among them are k tasks covering e_L and k tasks covering e_R. If a task covers e_L then it does not cover e_R and vice versa.*

In the next lemma we show how to construct a solution with $2k$ tasks if the given k-subset sum instance is a yes-instance.

▶ **Lemma 26.** *If the given k-subset sum instance is a yes-instance, then the constructed UFP-cover instance has a solution with $2k$ tasks.*

Conversely, we show that if the UFP-cover instance has a solution with at most $2k$ tasks then the k-subset sum instance is a yes-instance. Suppose we are given such a solution for the UFP-cover instance. First, we establish that for each $j \in [n]$ the solution selects either both $i(j)$ and $i'(j)$ or none of these two tasks.

▶ **Lemma 27.** *Given a solution T' to the UFP-cover instance with $2k$ tasks. For each $j \in [n]$ we have that either $\{i(j), i'(j)\} \subseteq T'$ or $\{i(j), i'(j)\} \cap T' = \emptyset$.*

Suppose we are given a solution T' to the UFP instance with $2k$ tasks (for which hence Lemma 27 applies). Let J' be the set of indices j such that $i(j) \in T'$. Note that Lemma 27 implies that $|J'| = k$.

▶ **Lemma 28.** *We have that $\sum_{j \in J'} a_j = B$.*

Hence, we proved that the constructed UFP-cover instance has a solution with $2k$ tasks if and only if the k-subset sum instance is a yes-instance. This implies that UFP-cover is $W[1]$-hard when parameterized by the number of tasks in the optimal solution. This completes the proof of Theorem 23.

7 Conclusion and open questions

In this paper we presented a PAS for UFP-cover and showed that the problem is FPT under resource augmentation or if additionally the number of different task sizes are bounded by a parameter. It remains open whether the problem is FPT if *only* the number task sizes is bounded by a parameter, but not the number of tasks in the optimal solution. Also, we showed that UFP-cover is $W[1]$-hard. Our $W[1]$-hardness proof is based on a reduction from the k-subset sum problem, which can be solved in pseudopolynomial time $O(nB)$. Hence, it is open whether UFP-cover is FPT if the input data are polynomially bounded. Our PAS can be simplified in this setting, however, it crucially relies on the slack obtained by selecting ϵk additional tasks and thus does not solve the problem optimally in this case.

References

1 Susanne Albers, Sanjeev Arora, and Sanjeev Khanna. Page replacement for general caching problems. In *SODA*, volume 99, pages 31–40. Citeseer, 1999.

2 Antonios Antoniadis, Ruben Hoeksma, Julie Meißner, José Verschae, and Andreas Wiese. A QPTAS for the General Scheduling Problem with Identical Release Dates. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.31.

3 Nikhil Bansal, Amit Chakrabarti, Amir Epstein, and Baruch Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC 2006)*, pages 721–729. ACM, 2006.

4 Nikhil Bansal, Ravishankar Krishnaswamy, and Barna Saha. On capacitated set cover problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 38–49. Springer, 2011.

5 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001. doi:10.1145/502102.502107.

6 Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 47–58, 2015. doi:10.1137/1.9781611973730.5.

7 Cristina Bazgan. Schémas d'approximation et Complexité Paramétrée, Rapport du stage (DEA). Technical report, Universitée Paris Sud, 1995.

8 Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.

9 Paul Bonsma, Jens Schulz, and Andreas Wiese. A constant-factor approximation algorithm for unsplittable flow on paths. *SIAM Journal on Computing*, 43:767–799, 2014.

10 Liming Cai and Xiuzhen Huang. Fixed-parameter approximation: Conceptual framework and approximability results. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 96–108, 2006. doi:10.1007/11847250_9.

11 Robert D Carr, Lisa K Fleischer, Vitus J Leung, and Cynthia A Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the 11th annual ACM-SIAM symposium on Discrete algorithms (SODA 2000)*, pages 106–115. Society for Industrial and Applied Mathematics, 2000.

12 Marco Cesati and Luca Trevisan. On the efficiency of polynomial time approximation schemes. *Inf. Process. Lett.*, 64(4):165–171, 1997. doi:10.1016/S0020-0190(97)00164-6.

13 Deeparnab Chakrabarty, Elyot Grant, and Jochen Könemann. On column-restricted and priority covering integer programs. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 355–368. Springer, 2010.

14 Yijia Chen, Martin Grohe, and Magdalena Grüber. On parameterized approximability. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 109–120, 2006. doi:10.1007/11847250_10.

15 Maurice Cheung, Julián Mestre, David B Shmoys, and José Verschae. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. *SIAM Journal on Discrete Mathematics*, 31(2):825–838, 2017.

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

17 Andrés Cristi and Andreas Wiese. Better approximations for general caching and UFP-cover under resource augmentation. Unpublished, 2019.

18 Rodney G. Downey, Michael R. Fellows, and Catherine McCartin. Parameterized approximation problems. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 121–129, 2006. doi:10.1007/11847250_11.

19 Michael R Fellows and Neal Koblitz. Fixed-parameter complexity and cryptography. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 121–131. Springer, 1993.

20 P. A. Franaszek and T. J. Wagner. Some distribution-free aspects of paging algorithm performance. *J. ACM*, 21(1):31–39, January 1974. doi:10.1145/321796.321800.

21 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. To augment or not to augment: Solving unsplittable flow on a path by creating slack. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, 2017. To appear.

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

23 Wiebke Höhn, Julián Mestre, and Andreas Wiese. How unsplittable-flow-covering helps scheduling with job-dependent cost functions. *Algorithmica*, 80(4):1191–1213, 2018.

24 Sandy Irani. Page replacement with multi-size pages and applications to web caching. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 701–710. ACM, 1997.

25 Daniel Lokshtanov, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*, pages 224–237. ACM, 2017.

26 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.

27 Andreas Wiese. A (1+epsilon)-approximation for unsplittable flow on a path in fixed-parameter running time. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 67:1–67:13, 2017.

A Reduction from Generalized Caching in the fault model

A reduction from generalized caching in the fault model to UFP-cover was given in [1, 5]. For completeness we present the reduction here using our notation. In the fault model of general caching we are given a value $M \in \mathbb{N}$ that denotes the size of the cache and we are given a set of pages $\mathcal{P}$. Each page $q \in \mathcal{P}$ has a (not necessarily unit) size $s(q) \in \mathbb{N}$. Also we are given a set of requests $\mathcal{R}$ where each request $j \in \mathcal{R}$ is characterized by a time $t_j \geq 0$ and a page $q_j \in \mathcal{P}$ meaning that at time t_j the page q_j has to be present in the cache. The goal

is to decide at what times we bring each page into the cache in order to minimize the total number of these transfers, assuming that initially the cache is empty. We show here how to reduce this problem to UFP-cover with unit weights.

▶ **Lemma 29.** *Given an instance $(\mathcal{P}, \mathcal{R}, M)$ of general caching in the fault model, in polynomial time we can compute an instance (V, E, T, u) of UFP-cover such that for any solution to $(\mathcal{P}, \mathcal{R}, M)$ with cost C, there is a solution $T' \subseteq T$ to (V, E, T, u) with $|T'| = C$, and vice versa.*

Proof. W.l.o.g. we can restrict ourselves to solutions of $(\mathcal{P}, \mathcal{R}, M)$ where each page enters the cache only when it is requested and leaves the cache only right after it is requested, and to instances where each page is requested at least once and $M < \sum_{q\in\mathcal{P}} s(q)$. Thus, a solution is completely defined by deciding at each point in time whether we evict the page that was just requested or whether we keep it in the cache until it is requested again. We construct the path (V, E) by defining one edge $e(t)$ for each time t such that there is a request $j \in \mathcal{R}$ with $t_j = t$, and ordering the edges on the path by increasing values of t. For defining the tasks T, we initialize $T := \emptyset$. Then, for every page $q \in \mathcal{P}$ and every pair $j_1, j_2 \in R$ of consecutive requests of page q, we add a task $i(j_1, j_2)$ to T with size $p_{i(j_1,j_2)} = s(q)$. The subpath $P_{i(j_1,j_2)}$ is the one that starts in the right vertex of $e(t_{j_1})$ and ends in the left vertex of $e(t_{j_2})$. We define the demand of the edge $e(t)$ to be $u_{e(t)} = p(T_{e(t)}) - M + \sum_{j\in R:t_j=t} s(q_j)$ for every time t. We also add an extra edge e_0 at the left of E with capacity $u_{e_0} = \sum_{q\in\mathcal{P}} s(q)$ and we add a task i^*_q with $P_{i^*_q} = \{e_0\}$ and $p_{i^*_q} = s(q)$ for each page $q \in \mathcal{P}$. The cost of these tasks is exactly the total cost of loading each page into the cache once, i.e., the first time that the respective page is requested.

Given a solution to $(\mathcal{P}, \mathcal{R}, M)$, we construct a solution T' for (V, E, T, u) in the following way. For every page q and every pair of consecutive requests j_1, j_2 of page q, we add $i(j_1, j_2)$ to T' if and only if page q is evicted from (and therefore re-loaded into) the cache between t_{j_1} and t_{j_2}. We also add i^*_q to T' for all $q \in \mathcal{P}$. It is clear that $|T'|$ is exactly the number of times a page is brought into the cache in the original solution. We now check that T' is a feasible solution. Consider an edge $e(t) \in E$. Then $p(T_{e(t)} \setminus T')$ is the sum of sizes of the pages that are in the cache at time t that are *not* requested exactly at time t. The total size of all pages in the cache is at most M, so $p(T_{e(t)} \setminus T') + \sum_{j\in R:t_j=t} s(q_j) \leq M$. Then, as $p(T_{e(t)}) = p(T_{e(t)} \cap T') + p(T_{e(t)} \setminus T')$ and $u_{e(t)} = p(T_{e(t)}) - M + \sum_{j\in R:t_j=t} s(q_j)$ we conclude that $p(T' \cap T_{e(t)}) \geq u_{e(t)}$. Also it holds by construction that $p(T' \cap T_{e_0}) = u_{e_0}$.

Let now T' be a feasible solution to (V, E, T, u). Of course $i^*_q \in T'$ for all $q \in \mathcal{P}$. This accounts for the first time each page is brought into the cache. We construct a solution S' to $(\mathcal{P}, \mathcal{R}, M)$ as follows. For every page q and every pair of consecutive requests j_1, j_2 of page q we keep page q in the cache between t_{j_1} and t_{j_2} if and only if $i(j_1, j_2) \notin T'$. Thus, for each element in T' we have to bring a page into the cache once, and then the cost of the solution is exactly $|T'|$. We have to check that the size of the pages in the cache never exceeds M in S'. In fact, note that the total size of the pages in the cache at time t is $p(T_{e(t)} \setminus T') + \sum_{j\in R:t_j=t} s(q_j)$. But $p(T_{e(t)} \cap T') \geq u_{e(t)}$, and therefore,

$$
\begin{aligned}
& p(T_{e(t)} \cap T') \geq p(T_{e(t)}) - M + \sum_{j\in R:t_j=t} s(q_j) \\
\Leftrightarrow \quad & M \geq p(T_{e(t)}) + p(T_{e(t)} \cap T') + \sum_{j\in R:t_j=t} s(q_j) \\
\Leftrightarrow \quad & M \geq p(T_{e(t)} \setminus T') + \sum_{j\in R:t_j=t} s(q_j)\,.
\end{aligned}
$$

◀