Better Approximations for General Caching and UFP-Cover Under Resource Augmentation

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

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

Abstract

In the Unsplittable Flow on a Path Cover (UFP-cover) problem we are given a path with a demand for each edge and a set of tasks where each task is defined by a subpath, a size and a cost. The goal is to select a subset of the tasks of minimum cost that together cover the demand of each edge. This problem models various resource allocation settings and also the general caching problem. The best known polynomial time approximation ratio for it is 4 [Bar-Noy et al., STOC 2000]. In this paper, we study the resource augmentation setting in which we need to cover only a slightly smaller demand on each edge than the compared optimal solution. If the cost of each task equals its size (which represents the natural bit-model in the related general caching problem) we provide a polynomial time algorithm that computes a solution of *optimal* cost. We extend this result to general caching and to the packing version of Unsplittable Flow on a Path in their respective natural resource augmentation settings. For the case that the cost of each task equals its "area", i.e., the product of its size and its path length, we present a polynomial time $(1+\epsilon)$-approximation for UFP-cover. If additionally the edge capacities are in a constant range we compute even a solution of optimal cost and also obtain a PTAS *without* resource augmentation.

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

Keywords and phrases General caching, unsplittable flow cover, approximation algorithm, resource augmentation

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.44

Funding *Andrés Cristi*: Supported by CONICYT-PFCHA/Doctorado Nacional/2018-21180347.
Andreas Wiese: Partially supported by FONDECYT Regular grant 1170223.

1 Introduction

Caching is one of the most classical problems in computer science. We are given a value $M \in \mathbb{N}$ that denotes the size of the cache and we are given a set of unit size pages $\mathcal{P}$. 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. Caching is a very well-studied problem in computer science with research on it dating back to the 1960s, see e.g., [8, 16, 9] and references therein. It admits a polynomial time algorithm in the offline setting [14] and in the online case there are several deterministic M-competitive algorithms [21, 9] and a randomized $O(\log M)$-competitive algorithm [15].

A natural generalization is the *general caching* problem where additionally each page $i \in \mathcal{P}$ has a (not necessarily unit) size $p_i \in \mathbb{N}$ and additionally a cost $w_i \in \mathbb{N}$ that we have to pay each time we bring i into the cache, the goal being to minimize the total cost. General caching can be modeled by a covering problem which turns out to be the natural covering

variant of the well-studied Unsplittable Flow on a Path problem (UFP) [13, 6, 1, 18]. We denote this covering problem by *UFP-cover*. Its input consists of a path $G = (V, E)$ and a set of tasks T. Each task $i \in T$ is characterized by a start vertex $s_i \in V$, an end vertex $t_i \in V$, a size $p_i \in \mathbb{N}$ and a cost $w_i \in \mathbb{N}$. For each edge $e \in E$ we are given a demand u_e and we denote by $T_e \subseteq T$ the set of tasks i such that e lies on the path P_i between s_i and t_i. Our goal is to select a subset of the tasks $\bar{T} \subseteq T$ such that $p(\bar{T} \cap T_e) \geq u_e$ for each edge e where for any set of tasks $T' \subseteq T$ we define $p(T') := \sum_{i \in T'} p_i$ and $w(T') := \sum_{i \in T'} w_i$. Our objective is to minimize $w(\bar{T})$. When we reduce general caching to UFP-cover each time t_j of some request j is represented by an edge of G and there is a task i for each two consecutive requests of a page i' where intuitively selecting i represents loading i' again into the cache at the time of the second request and $p_i = p_{i'}$ and $w_i = w_{i'}$; see [6, 1] for details. Hence, if we restrict the sizes and costs in the considered instances of general caching then this restricts the sizes and costs of the resulting instance of UFP-cover in the same way. Additionally, UFP-cover is motivated by resource allocation settings where, e.g., the edge demands represent minimum requirements for energy, bandwidth, or workers and the tasks represent possibilities to satisfy a part of this demand during some given time interval at a certain cost.

General caching and UFP-cover are NP-hard which motivates studying approximation algorithms for them. The best known polynomial time approximation ratio for both problems is 4 [6] and there has been no improvement on this in almost 20 years. In this paper, we study the resource augmentation setting in which we are given a value $\delta > 0$ such that for each edge e we need to cover only a demand of $(1 - \delta)u_e$ while the compared optimum needs to cover u_e. No better algorithm is known for this setting.

1.1 Our Contribution

We first study the case of UFP-cover where $p_i = w_i$ for each task i for which the best known result is still the mentioned 4-approximation for the general case [6]. We present a polynomial time algorithm that computes a solution (that is feasible under resource augmentation) whose cost is at most the cost of the optimal solution (without resource augmentation). Hence, intuitively we solve the problem optimally under resource augmentation. We consider first the special case where the edge demands are in a constant range. We prove that there are solutions of cost at most OPT that are feasible under resource augmentation and which have the following structure: we can partition E into subpaths where on each subpath there are constantly many special edges such that each task of relatively small size (*small task*) in the solution uses one of these edges. This drastically simplifies the computations for the small tasks: we design an algorithm that guesses the partition of E into subpaths and then on each subpath it approximately guesses the small tasks crossing the constantly many special edges in OPT. After that, it additionally selects tasks with relatively large sizes (*large tasks*) via a dynamic program. For the case of arbitrary edge demands we consider for each edge e the amount by which the optimal solution covers e and partition the edges according to ranges of these values. For each range we show that there is a partition of E into subpaths with constantly many special edges for the small tasks, like in the case of a constant range of edge demands. Then, we design a dynamic program that intuitively patches the solutions for these ranges together.

▶ **Theorem 1.** *For any constant $\delta > 0$ there is a polynomial time algorithm for the case of UFP-cover where $w_i = p_i$ for each task i, that computes a solution with optimal cost that is feasible under $(1 - \delta)$-resource augmentation.*

Using our techniques, we derive an algorithm for the case of general caching where $p_i = w_i$ for each page $i \in \mathcal{P}$ which is known as the *bit-model* [20], meaning that the cost w_i of bringing a page i into the cache is proportional to its size p_i (which is a natural assumption). Our algorithm computes a solution that is feasible for a slightly increased cache of size $(1+\delta)M$ and whose cost is at most the cost of the optimal solution for a cache of size M. The notions of resource augmentation for UFP-cover and general caching are not equivalent w.r.t. the known reduction from general caching to UFP-cover, i.e., an algorithm for UFP-cover under resource augmentation does *not* imply an algorithm for general caching under resource augmentation. Therefore, we derive a reduction from general caching to UFP (instead of UFP-cover) in which we have the same input as for UFP-cover but we want to select a set of tasks $\bar{T}$ of *maximum* total weight such that on each edge e the tasks $\bar{T}$ do not exceed the capacity, i.e., $p(\bar{T} \cap T_e) \leq u_e$. We argue that if we increase the size of the cache in a general caching instance by a factor $1+\delta$ then in the reduced UFP instance the capacity of each edge increases by at least a factor $1+\delta$. We adapt our new techniques for UFP-cover above to UFP and obtain an algorithm for UFP that computes a solution of value OPT if we can increase the capacity of each edge by a factor $1+\delta$ and if $p_i = w_i$ for each task i. This yields an algorithm for general caching under resource augmentation for the case that $p_i = w_i$ for each page i, computing again a solution of cost at most OPT.

▶ **Theorem 2.** *For any constant $\delta > 0$ there are algorithms with polynomial running time for the cases of general caching and UFP where $w_i = p_i$ for each page/task i that compute solutions with optimal cost that are feasible under $(1+\delta)$-resource augmentation.*

Then we study the case of UFP-cover in which the cost w_i of each task i equals its "area", i.e., its size p_i multiplied by the length of its path P_i. We first prove that if the edge capacities are in a constant range then we can compute a $(1+\epsilon)$-approximation under resource augmentation by extending techniques from [17]. Then we turn this routine into a PTAS *without* resource augmentation for the same setting. To this end, we prove that there are $(1+\epsilon)$-approximate solutions in which for each edge e either all small input tasks using it are selected or e is covered to an extent of at least $(1+\delta^2)u_e$ which yields some slack. We intuitively guess the edges e of the former type, select all input tasks using them, and then apply our algorithm for resource augmentation on the remaining edges. With similar ideas, we construct an algorithm that computes a solution with optimal cost under resource augmentation for a constant range of edge capacities.

Then we present a polynomial time $(1+\epsilon)$-approximation under resource augmentation for arbitrary edge demands, under the same assumption on the task's costs. To construct this algorithm, we provide a reduction that essentially turns a polynomial time α-approximation for the special case of a *constant range* of edge capacities into a polynomial time $(1+\epsilon)\alpha$-approximation algorithm for *arbitrary* edge capacities under resource augmentation. We apply this reduction to the previous algorithm which yields a $(1+\epsilon)$-approximation under resource augmentation. The reduction works for arbitrary cost functions and in particular it might be useful for future work. To derive such a reduction, it might seem natural to split the overall problem into subproblems corresponding to the different ranges of the edge capacities. However, in UFP-cover there can be an edge e with very small demand which in the optimal solution is covered by tasks whose total size is very large. Hence, the demand of an edge might not give us a good estimate for how much the optimal solution covers it. Therefore, our reduction is guided by the (unknown) amount by which the optimal solution covers each edge, instead of the edge demands themselves. The resulting algorithm is a dynamic program which makes repeated calls to the given algorithm for a constant range of edge capacities and in which solutions of some DP-cells yield input tasks of other cells.

▶ **Theorem 3.** *Consider the case of UFP-cover where $w_i = |P_i| \cdot p_i$ for each task i. For any constants $\epsilon, \delta > 0$ there is a polynomial time algorithm that computes*

- *a $(1+\epsilon)$-approximate solution that is feasible under $(1-\delta)$-resource augmentation,*
- *a $(1+\epsilon)$-approximate solution without resource augmentation, if the edge capacities are in a constant range,*
- *a solution with cost at most OPT that is feasible under $(1-\delta)$-resource augmentation, if the edge capacities are in a constant range.*

Due to space constraints almost all proofs are deferred to the full version of the paper.

1.2 Other related work

UFP-cover is a generalization of the knapsack-cover problem. For the latter, an LP-formulation with a constant integrality gap is known [10] based on the knapsack-cover inequalities which are also used in other settings [12, 5, 11]. On the other hand, UFP-cover is a special case of the capacitated set cover problem, e.g., [11, 4], in which we are given a set of elements with demands and a family of sets where each set has a size and one seeks to select sets such that each element is covered by sets whose total size is at least the demand of the element.

For UFP-cover there is a QPTAS if the input data is quasi-polynomially bounded [19] and with the reduction in [6, 1] the same holds for general caching. A related problem is the general scheduling problem on one machine without release dates in which we are given a set of jobs where for each job we have to pay a cost that depends on its completion time. The best known approximation algorithm for this problem is a $(4+\epsilon)$-approximation [12] that generalizes the 4-approximation for UFP-cover in [6] and there is a QPTAS for quasi-polynomially bounded input data [2].

For UFP (packing) the best known polynomial time approximation ratio is $5/3+\epsilon$ [18] and there is a QPTAS [3, 7]. For the cases that the weight of each task is proportional to its size or to its "area" even PTASs are known [7, 17]. In this paper we extend the PTAS for the latter case to UFP-cover under resource augmentation for bounded edge demands. However, for the case where $p_i = w_i$ for each task i we need a completely different approach.

2 Task costs proportional to size

Given a constant $\delta > 0$, we present a polynomial time algorithm for UFP-cover for the case that $p_i = w_i$ for each task i. Our algorithm computes a solution that is feasible under $(1-\delta)$-resource augmentation whose cost is at most the cost of the optimal solution without resource augmentation.

By adding edges with demand 0 we can assume w.l.o.g. that the start and end vertices of the input tasks of any considered instance are pairwise distinct. First, we describe an algorithm for the special case that there is a value U such that $u_e \in [\delta U, U)$ for each edge e and later we extend this algorithm to the general case.

We start by showing that there is a well-structured solution whose cost is at most $w(OPT)$. Our algorithm will later compute a solution with this structure. We classify tasks into large and small tasks. A task i is *large* if $p_i \geq \delta^3 U$ and *small* otherwise. We denote by T_L and T_S the large and small input tasks, respectively. First, we establish some properties of the optimal solution OPT that in fact hold for arbitrary task costs (assuming that $u_e \in [\delta U, U)$ for each edge e).

▶ **Lemma 4.** *Let OPT be an optimal solution. For each edge e it holds that $p(OPT \cap T_S \cap T_e) \leq (2+2\delta^3)U$ and $|OPT \cap T_L \cap T_e| \leq O(1/\delta^3)$.*

We want to cut the given instance into simpler subinstances via a partition of E into subpaths $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ such that intuitively we can compute an optimal solution for each subpath E_j separately and then output the union. For each subpath E_j we require that there are $9/\delta$ *special edges* $e_{j,1}, e_{j,2}, ..., e_{j,j'} \in E_j$ and that there is a set $T'_j \subseteq T_S$ such that each task in T'_j uses at least one of the edges $e_{j,1}, e_{j,2}, ..., e_{j,j'}$ and the tasks in T'_j, together with a global set of large tasks $T'_L \subseteq T_L$, form a feasible solution for E_j under resource augmentation. Then we define a solution T' to be the union of the sets T'_j together with T'_L. Note that a small task i might be contained in several sets T'_j and in this case we add it several times to T', i.e., we allow T' to be a multiset. Formally, we look for solutions T' that are *nice*. Figure 1 gives some intuition on how such a solution looks like.

▶ **Definition 5.** *A multiset T' is* nice *if there exists a partition of E into subpaths $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ and partition of T' into sets $T' = T'_L \dot\cup T'_1 \dot\cup T'_2 \dot\cup ... \dot\cup T'_k$ such that*

- *$T'_L = T' \cap T_L$,*
- *for each j we have that $T'_j \subseteq T_S$ and T'_j contains each task at most once,*
- *for each subpath E_j there are at most most $9/\delta$ edges $e_{j,1}, e_{j,2}, ..., e_{j,j'} \in E_j$ such that each task $i \in T'_j$ uses at least one of them, and for each $e \in E_j$ we have that $p(T_e \cap (T'_j \cup T'_L)) \geq (1-\delta/2)u_e$.*

▶ **Lemma 6.** *There exists a nice multiset T' with a corresponding partition $T' = T'_L \dot\cup T'_1 \dot\cup T'_2 \dot\cup ... \dot\cup T'_k$ such that $w(T'_L) + \sum_{k'=1}^{k} w(T'_{k'}) \leq w(OPT)$.*

Proof sketch. We define $T'_L := OPT \cap T_L$. For any two vertices $u, v \in V$ denote by $P_{u,v}$ the path between u and v. We define the partition $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ and the corresponding sets T'_j inductively. Suppose that we have already defined $k'-1$ paths $E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_{k'-1}$. Let v_0 denote the rightmost vertex of $E_{k'-1}$ and for the case that $k'=1$ let v_0 be the leftmost vertex of V. We define $e_{k',1} = \{u_1, v_1\}$ to be the leftmost edge such that the total size of small tasks $T_S \cap OPT$ whose path is contained in P_{v_0,u_1} is at least $\delta^2 U/4$. The total size of those tasks is at most $\delta^2 U/3$ since $p_i \leq \delta^3 U$ for each small task i and the end vertices of the input tasks are pairwise distinct. Inductively, suppose that we have defined j' edges $e_{k',1}, e_{k',2}, ..., e_{k',j'}$ in this way and let $e_{k',j'} = \{u_{j'}, v_{j'}\}$. We define $e_{k',j'+1} = \{u_{j'+1}, v_{j'+1}\}$ to be the leftmost edge on the right of $e_{k',j'}$ such that the total size of small tasks $T_S \cap OPT$ whose path is contained in $P_{v_{j'},u_{j'+1}}$ is at least $\delta^2 U/4$ (and hence at most $\delta^2 U/3$). We stop after defining $e_{k',9/\delta^2} = \{u_{9/\delta^2}, v_{9/\delta^2}\}$ and define $E_{k'} := P_{v_0,v_{9/\delta^2}}$. We define $T'_{k'}$ to be all tasks in $T_S \cap OPT$ whose path contains one of the edges $e_{k',1}, e_{k',2}, ..., e_{k',9/\delta^2}$. Note that the total cost of tasks in $OPT \cap T_S \setminus T'_{k'}$ that use an edge of $E_{k'}$ (i.e., small tasks of OPT that we did not add to $T'_{k'}$) is at least $\frac{9}{\delta^2} \cdot \frac{\delta^2 U}{4} \geq (2+\delta)U$. This justifies that tasks in $OPT \cap T_S$ using the rightmost edge of $E_{k'}$ might be added to $T'_{k'}$ and to $T'_{k'+1}$ and hence we have to pay twice for them (their total size and hence their total cost is at most $(2+2\delta^3)U$ by Lemma 4). We stop if during some iteration k we cannot find a next edge $e_{k,j'+1} = \{u_{j'+1}, v_{j'+1}\}$ according to our definition. In this case we define E_k to be the path between v_0 and the rightmost vertex of V and stop the construction procedure. One can show that the set $T'_L \cup \bigcup_{k'} T'_{k'}$ is feasible under resource augmentation, i.e., that $p(T_e \cap (T'_L \cup T'_j)) \geq (1-\delta/2)u_e$ for each $e \in E_j$ for each j and that $w(T') = w(OPT \cap T_L) + \sum_{k'} w(T'_{k'}) \leq w(OPT)$. ◀

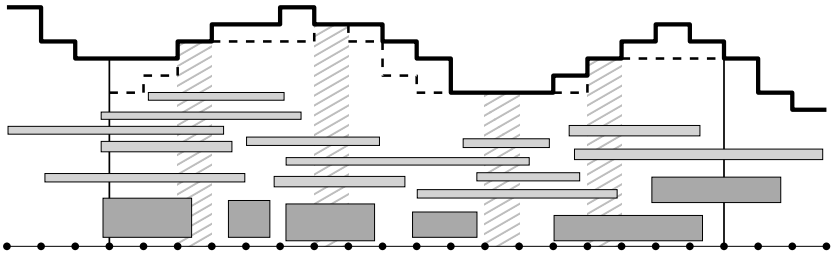

Figure 1 Some of the tasks of a nice solution covering a subpath E_j. The vertical lines are the boundaries of E_j and the shaded columns represent the few special edges $e_{j,1}, e_{j,2}, ..., e_{j,j'} \in E_j$. All small tasks (depicted in light gray) cross one of those edges. The demand covered by the solution (dashed curve) might be by a factor $(1 - \delta/2)$ smaller than the demand of the edges (thick curve). Note that the complete nice solution contains many more tasks covering E_j than the ones shown above.

The algorithm

We present now an algorithm that intuitively computes a nice solution $\bar{T}$ whose cost is at most $w(T')$. We first present such an algorithm for the case that for the partition $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ of T' it holds that $k = 1$ and then extend it later to the case that $k > 1$.

Assume that $k = 1$. We guess the at most $9/\delta^2$ edges $e_{1,1}, e_{1,2}, ..., e_{1,j'}$ in time $n^{O(1/\delta^2)}$, i.e., we enumerate all possibilities. We guess an estimate for the capacity needed by the tasks in T'_1 on each edge. To this end, for each edge $e \in E$ let $f_1(e) := p\left(T'_1 \cap T_e \cap T_{e_{1,1}}\right)$ denote the total size of the tasks in $T'_1 \cap T_{e_{1,1}}$ that use e. Intuitively, we would like to guess the function f_1, however, there are too many possibilities for it. Therefore, instead we guess the estimate $\hat{f}_1(e) := \left\lfloor \frac{f_1(e)}{\delta^4 U/36} \right\rfloor \delta^4 U/36$. Using that f_1 is non-decreasing on the left of $e_{1,1}$ and non-increasing on the right of $e_{1,1}$, we will show that $\hat{f}_1(e)$ has only $O(1/\delta^4)$ many steps. We define inductively functions $\hat{f}_2, ..., \hat{f}_{j'}$ where each function $\hat{f}_{j''}$ is an estimate for the size of the tasks in T'_1 that use $e_{1,j''}$ but *not* $e_{1,j''-1}$. Formally, we define $f_{j''}(e) := p\left(\left(T'_1 \cap T_e \cap T_{e_{1,j''}} \cap T_S\right) \setminus T_{e_{1,j''-1}}\right)$ and $\hat{f}_{j''}(e) := \left\lfloor \frac{f_{j''}(e)}{\delta^4 U/36} \right\rfloor \delta^4 U/36$ for each $j'' = 2, ..., j'$.

▶ **Lemma 7.** *For each $j'' \in \{1, ..., j'\}$ the function $\hat{f}_{j''}$ is a step function with only $O(1/\delta^4)$ many steps whose values are all integral multiples of $\delta^4 U/36$ bounded by $(2 + 2\delta^3)U$. Also, for each edge $e \in E$ we have that $\sum_{j''} \hat{f}_{j''}(e) \geq \sum_{j''} f_{j''}(e) - \delta^2 U/4$.*

We guess each function $\hat{f}_{j''}$ with $j'' \in \{1, ..., j'\}$ in time $n^{O(1/\delta^4)}$ which gives $n^{O(1/\delta^6)}$ many guesses in total. For each function $\hat{f}_{j''}$ we invoke a polynomial time algorithm that computes a set of tasks $\bar{T}_{1,j''}$ that essentially covers $\hat{f}_{j''}$ and that is at most as costly as the tasks $T'_1 \cap T_{e_{1,j''} \cap T_S} \setminus T_{e_{1,j''-1}}$ (that define the profile $f_{j''}$).

▶ **Lemma 8.** *Let $j'' \in \{1, ..., j'\}$. There is an algorithm with a running time of $n^{O(1/\delta^4)}$ that computes a set of tasks $\bar{T}_{1,j''} \subseteq T_S \cap T_{e_{1,j''}} \setminus T_{e_{1,j''-1}}$ with $p(\bar{T}_{1,j''} \cap T_e) \geq \hat{f}_{j''}(e) - \delta^4 U/36$ for each edge e, and $w(\bar{T}_{1,j''}) \leq w(T'_1 \cap T_{e_{1,j''}} \setminus T_{e_{1,j''-1}})$.*

We define $\bar{T}_1 := \bigcup_{j''} \bar{T}_{1,j''}$ to be the small tasks that we select in order to cover E_1. It remains to select the large tasks. Due to Lemma 4 each edge is used by at most $O(1/\delta^3)$ tasks in T'_L. Therefore, we can use a dynamic program that computes the cheapest set of large tasks $\bar{T}_L$ that covers the demand that is not already covered by $\bar{T}_1$ (taking into account that we have resource augmentation). Intuitively, it sweeps the path from left to right and for each edge e it guesses the at most $O(1/\delta^3)$ many large tasks in $T'_L \cap T_e$. Finally, we output $\bar{T} := \bar{T}_1 \cup \bar{T}_L$.

▶ **Lemma 9.** *There is an algorithm with a running time of $n^{O(1/\delta^3)}$ that computes a set of tasks $\bar{T}_L \subseteq T_L$ such that $p((\bar{T}_L \cup \bar{T}_1) \cap T_e) \geq (1-\delta)u_e$ for each edge $e \in E$ and $w(\bar{T}_L) \leq w(T'_L)$.*

For the case that $k > 1$ we define a dynamic program that intuitively guesses the partition $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ step by step and the large tasks at the boundaries of the subpaths. After guessing a subpath E_j and the large tasks using its boundary edges it invokes the algorithm for $k = 1$ as a subroutine on E_j and then continues with the guessing.

Formally, our DP has a cell (E'', T'') for each combination of a subpath E'' of E that contains the rightmost edge of E and a set of at most $O(1/\delta^3)$ large tasks T'' that use the leftmost edge of E'', denoted by e''_L. The reader may imagine that $E'' = E_j \cup E_{j+1} \cup ... \cup E_k$ for some j (where the subpaths E_j correspond to the nice solution T') and that T'' are the large tasks in T' that use the leftmost edge of E_j. The goal is to compute a set $\bar{T}''$ of tasks of low cost such that $\bar{T}'' \cup T''$ forms a feasible solution for E'' under resource augmentation, i.e., $p(T_e \cap (\bar{T}'' \cup T'')) \geq (1-\delta)u_e$. Given a cell (E'', T'') we intuitively guess $E_{j+1} \cup ... \cup E_k$, i.e., we try all subpaths $\bar{E}'' \subseteq E''$ that contain the rightmost edge of E and all sets $\bar{T}''$ of $O(1/\delta^3)$ large tasks that use the leftmost edge of $\bar{E}''$, denoted by $\bar{e}''_L$, such that T'' and $\bar{T}''$ are compatible, i.e., $T'' \cap T_{\bar{e}''_L} \subseteq \bar{T}''$ and $\bar{T}'' \cap T_{e''_L} \subseteq T''$. Let $\tilde{E}'' := E'' \setminus \bar{E}''$ (the reader may imagine that $\tilde{E}'' = E_j$). On $\tilde{E}''$ we apply the procedure above for the case of $k = 1$ and we slightly change the algoritm due to Lemma 9 such that we require that the large tasks $T'' \cup \bar{T}''$ are included in the computed set $\bar{T}_L$. Let $\hat{T}$ denote the resulting tasks. With the guess $(\bar{E}'', \bar{T}'')$ we associate the solution $\hat{T} \cup (\bar{T}'' \setminus T'') \cup DP(\bar{E}'', \bar{T}'')$ where $DP(\bar{E}'', \bar{T}'')$ denotes the solution stored in the cell $(\bar{E}'', \bar{T}'')$. We store in the cell (E'', T'') the solution of minimum cost among all guesses. Assume for convenience that we append an edge on the left of E with zero demand. We output the solution stored in the cell $(E, \emptyset)$.

Arbitrary demands

We generalize the above algorithm to the case of arbitrary edge demands. First, we change the definition of large and small tasks and in particular make it dependent on the (unknown) optimal solution. For each edge $e \in E$ we define $\hat{u}_e := p(OPT \cap T_e)$ and we define its level $\ell(e)$ to be the integer ℓ such that $\hat{u}_e \in [(1/\delta)^\ell, (1/\delta)^{\ell+1})$. For each task $i \in T$ we define its level $\ell(i)$ by $\ell(i) := \min_{e \in P_i} \ell(e)$. We say that a task $i \in T$ is *large* if $p_i \geq \delta^3 (1/\delta)^{\ell(i)+1}$ and *small* otherwise. For each level ℓ we define T^ℓ_S and T^ℓ_L to be the small and large tasks of level ℓ, respectively, and $T^\ell = T^\ell_S \cup T^\ell_L$. We extend the notion of a nice multiset T' to the case of arbitrary demands. Again, we have a partition of E into subpaths $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ and a partition of T' into sets $T' = T'_1 \dot\cup T'_2 \dot\cup ... \dot\cup T'_k$ but now for each subpath E_j there can be up to $18/\delta^2$ special edges *for each level* ℓ. Similarly as before, for each edge $e \in E_j$ the small tasks in T'_j crossing one of these edges cover e together with the large tasks in T' (assuming that we have resource augmentation). Due to the resource augmentation we can even require that already the tasks in T' of levels $\ell(e) - 1$ and $\ell(e)$ are sufficient for covering e.

▶ **Definition 10.** *A multiset $T' \subseteq T$ is* nice-by-levels *if there exists a partition into subpaths $E = E_1 \dot\cup ... \dot\cup E_k$ and sets $T' = T'_1 \dot\cup ... \dot\cup T'_k$ with the property that each set T'_j contains each task i at most once and for each subpath E_j and each level ℓ there is a set of at most $18/\delta^2$ edges $e_{j,1,\ell}, e_{j,2,\ell}, ... \in E_j$ of level ℓ such that $T'_j \cap T^\ell_S \subseteq \bigcup_{j'} T_{e_{j,j',\ell}} \cap T^\ell_S$. Moreover, for each edge $e \in E_j$ of some level ℓ we have that $p(T_e \cap T'_j \cap (T^\ell \cup T^{\ell-1})) \geq (1-3\delta)\hat{u}_e \geq (1-3\delta)u_e$ and $|T' \cap T_e \cap T^\ell_L| \leq 1/\delta^3$.*

▶ **Lemma 11.** *There is a set T' that is nice-by-levels with $\sum_{k'} w(T'_{k'}) \leq w(OPT)$.*

Proof sketch. Assume that we already defined a set of vertices V' and a set of special vertices for each level $\ell' \in \{0, ..., \ell - 1\}$ such that the vertices V' separate E into subpaths E_j such that for each level $\ell' \in \{0, ..., \ell - 1\}$ each subpath E_j contains at most $18/\delta^2$ special edges of level ℓ'. For each subpath E_j we consider its edges of level ℓ and apply a slight adaptation of the procedure from the proof of Lemma 6. Let v_0 be the leftmost vertex of E_j. We define $e_{j,1,\ell} = \{u_1, v_1\}$ to be the leftmost edge such that the total size of the small tasks of level ℓ ending in P_{v_0,u_1} is at least $\delta^2(1/\delta)^{\ell+1}/4$ and hence at most $\delta^2(1/\delta)^{\ell+1}/3$. Inductively, we define special edges $\{e_{j,j',\ell}\}_{j\in[k],j'\in\mathbb{N}}$ in this way. If an edge e of level $\ell' > \ell$ is selected as a special edge we replace it by the two closest edges of level ℓ. We do this operation for each level ℓ. If we defined more than $18/\delta^2$ special edges of level ℓ, then we add to V' the right vertex of one in every $18/\delta^2$ special edges, so we partition E_j into subpaths such that each subpath contains at most $18/\delta^2$ special edges of level ℓ.

Let E_j be a subpath of the final partition, let $\{e_{j,j',\ell}\}_{j',\ell\in\mathbb{N}}$ be the special edges contained in E_j, and let $T'_j := OPT \cap \left(\{i \in T_L | P_i \cap E_j \neq \emptyset\} \cup \bigcup_\ell \bigcup_{j'} T_{e_{j,j',\ell}}\right)$ be the tasks corresponding to E_j. With a similar reasoning as in Lemma 6 before we argue that for each edge $e \in E_j$ the tasks in T'_j that cross e and additionally at least one of the edges $\{e_{j,j',\ell}\}_{j',\ell\in\mathbb{N}}$ have a total size of at least $(1-\delta)\hat{u}(e)$, i.e., essentially cover e. One can show that the tasks of level $\ell(e) - 2$ or smaller covering e have a total size of at most $2\delta\hat{u}_e$, using that each of them must use the closest edge on the left or on the right of e that is of level $\ell(e) - 2$ and each of them is used by tasks in OPT of total size at most $(1/\delta)^{\ell-1} \leq \delta\hat{u}_e$. We define $T' = \bigcup_{j=1}^{k} T'_j$ to be the constructed multiset. With a similar argumentation as in Lemma 6 one can show that $\sum_{k'} w(T'_{k'}) \leq w(OPT)$, arguing that some tasks in $OPT \cap T_S$ are not included in T' and that they compensate for the additional cost of tasks in OPT that are included several times in T'. ◀

We describe now a dynamic program that intuitively computes a nice-by-levels solution $\bar{T}$ with $w(\bar{T}) \leq w(T')$. Consider first the case that $k = 1$ and let $j = 1$. Unlike the case of a constant range of edge capacities, we cannot guess all edges $\{e_{j,j',\ell}\}_{j',\ell\in\mathbb{N}}$ in one step since they can be more than constantly many. Instead, we start with $\ell := 0$ we first guess all $O(1/\delta^2)$ special edges $\{e_{j,j',\ell}\}_{j'\in\mathbb{N}}$ of level ℓ and for each of these edges $e_{j,j',\ell}$ we guess the large tasks in T' using $e_{j,j',\ell}$, i.e., $T' \cap T_L^\ell \cap T_{e_{j,j',\ell}}$ and an approximation of the profiles of the small tasks that use $e_{j,j',\ell}$ and no special edge $e_{j,\hat{j}',\hat{\ell}}$ with $\hat{\ell} < \ell$ or with $\hat{\ell} = \ell$ and $\hat{j}' < j'$ (like in Section 2).

Formally, we define $f_1(e) := p\left(T'_j \cap T_e \cap T_{e_{j,1,\ell}} \setminus \bigcup_{\ell'<\ell} \bigcup_{\hat{j}} T_{e_{j,\hat{j},\ell'}}\right)$ and for each $j' > 1$ we define $f_{j'}(e) := p\left(T'_1 \cap T_e \cap T_{e_{j,j',\ell}} \setminus \left(T_{e_{j,j'-1,\ell}} \cup \bigcup_{\ell'<\ell} \bigcup_{\hat{j}} T_{e_{j,\hat{j},\ell'}}\right)\right)$. Then we define the approximative profiles by $\hat{f}_{j'}(e) := \left\lfloor \frac{f_{j'}(e)}{\delta^4(1/\delta)^\ell/36} \right\rfloor (1/\delta)^\ell\delta^4/36$ for each j'. Note that each function $\hat{f}_{j'}(e)$ has only $O(1/\delta^5)$ many steps. Since there are at most $18/\delta^2$ special edges of each level, we can guess all functions $\hat{f}_{j'}(e)$ in time $n^{O(1/\delta^7)}$. For each guessed profile we compute small tasks that essentially cover it, like in Lemma 8.

▶ **Lemma 12.** *For each j' there is an algorithm with a running time of $n^{O(1/\delta^5)}$ that computes a set of tasks $\bar{T}_{j,j',\ell} \subseteq T_{e_{j,j',\ell}} \setminus \left(T_{e_{j,\ell'-1,\ell}} \cup \bigcup_{\ell'<\ell} \bigcup_{\hat{j}} T_{e_{j,\hat{j},\ell'}} \cup \left(T' \cap T_L^\ell \cap T_{e_{j,j',\ell}}\right)\right)$ with $p(\bar{T}_{j,j',\ell} \cap T_e) \geq \hat{f}_{j'}(e) - \delta^4(1/\delta)^\ell/36$ for each edge e, and $w(\bar{T}_{j,j',\ell}) \leq f_{j'}(e_{j,j',\ell})$.*

Let E' be a subpath between two consecutive special edges in $\{e_{j,j',\ell}\}_{j'\in\mathbb{N}}$. It might be that E' contains edges of level ℓ. Denote these edges by E'_ℓ. We want to guess the large tasks of level ℓ that use an edge in E'_ℓ. In order to do this we invoke a dynamic program that intuitively sweeps in E' from left to right and in each step guesses an edge $e \in E'_\ell$ together

with the tasks $T' \cap T_e \cap T_L^\ell$. We recurse in each subpath E'' between two consecutive edges in E'_ℓ, between the leftmost edge of E' and the leftmost edge in E'_ℓ, and between the rightmost edge of E'_ℓ and the rightmost edge of E'. In each recursive call, the arguments consists of the subpath E'', the next level $\ell+1$, and the guessed $O(1/\delta^2)$ special edges of level ℓ and their profiles, and the large tasks of level ℓ using the leftmost edge of E'' or the rightmost edge of E''. In principle, by doing this we forget (and thus lose) the amount that tasks from levels $\ell-1$ and below cover on edges in E''. However, T' is nice-by-levels and thus we have that on each edge e of some level ℓ the tasks in $T' \cap (T^\ell \cup T^{\ell-1})$ are sufficient to cover the demand u_e (under resource augmentation). For the arguments in our recursive call there is only a polynomial number of options. Therefore, we can embed this recursion into a polynomial time dynamic program. The base case is given when the path E'' is empty or if the level ℓ is so large that $T_S^\ell = T_L^\ell = \emptyset$, i.e., if $\max_{e \in E} \hat{u}_e \leq \sum_{i \in T} p_i < (1/\delta)^\ell$.

Finally, if $k > 1$ then we use the same dynamic program as before in order to guess the partition $E = E_1 \dot\cup ... \dot\cup E_k$. Thus we have the following theorem. Its proof and the complete description of the algorithm is deferred to the full version of the paper.

▶ **Theorem 13.** *For any $\delta > 0$ there is an algorithm with a running time of $n^{(1/\delta)^{O(1)}}$ for the case of UFP-cover where $w_i = p_i$ for each task i that computes a solution T' with $w(T') \leq w(OPT)$ and that satisfies that $p(T_e \cap T') \geq (1-\delta)u_e$ for each edge e.*

2.1 General caching and UFP

We use the techniques from the previous algorithm to obtain algorithms for UFP and general caching under resource augmentation. First, we show how to reduce general caching to UFP.

▶ **Lemma 14.** *Given an instance $(\mathcal{P}, \mathcal{R}, M)$ of general caching such that $p_i = w_i$ for each page $q_i \in \mathcal{P}$. In polynomial time we can compute an instance (V, E, T, u) of UFP with $u_e \leq M$ for each edge $e \in E$ and $p_i = w_i$ for each task $i \in T$ such that*

1. *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 $w(T') = w(T) - C$ and vice versa,*
2. *for any solution to $(\mathcal{P}, \mathcal{R}, M(1+\delta))$ with cost C there is a solution T' to $(V, E, T, u + \mathbb{1}\delta M)$ with $w(T') = w(T) - C$ and vice versa.*

Lemma 14 implies that in order to obtain an algorithm for general caching under resource augmentation it sufficies to provide an algorithm for UFP under $(1+\delta)$-resource augmentation, i.e., where the capacity of each edge e is increased to $(1+\delta)u_e$. We construct an algorithm for UFP of the latter type. We begin with the case of a constant range of edge capacities, i.e., $u_e \in [\delta U, U)$. We define a task $i \in T$ to be *large* if $p_i \geq \delta^3 U$ and *small* otherwise. Denote by T_L and T_S the set of large and small input tasks, respectively. Like in the case of UFP-cover we are looking for solutions that are *nice* which in this case means that there is a partition of E into subpaths $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ such that we restrict ourselves to tasks i such that $P_i \subseteq E_j$ for some $j \in \{1, ..., k\}$ and additionally for each set E_j there are some special edges $e_{j,1}, e_{j,2}, ..., e_{j,\ell}$ such that we select *each* task $i \in T_S$ with $P_i \subseteq E_j$ that does *not* use any of these special edges.

▶ **Definition 15.** *A set $T' \subseteq T$ is* nice *if there exists a partition of E into subpaths $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$ such that*

- *for each task $i \in T'$ we have that $P_i \subseteq E_j$ for some $j \in \{1, ..., k\}$,*
- *for each E_j there are at most $4/\delta^2$ edges $e_{j,1}, e_{j,2}, ..., e_{j,\ell} \in E_j$ such that $\{i \in T_S : P_i \subseteq E_j\} \setminus \bigcup_{\ell'=1}^{\ell} T_{e_{j,\ell'}} \subseteq T'$ and for each edge $e \in E_j$ we have that $p(T' \cap T_e) \leq u_e + \delta^2 U/2$.*

▶ **Lemma 16.** *There is a nice set T' that satisfies $w(T') \geq w(OPT)$.*

Our algorithm is now similar as the algorithm for UFP-cover above. If $k > 1$ then we first invoke a dynamic program that guesses the partition $E = E_1 \dot\cup E_2 \dot\cup ... \dot\cup E_k$. Let T'_j be the tasks $i \in T'$ with $P_i \subseteq E_j$. For each subpath E_j we guess the special edges $e_{j,1}, e_{j,2}, ...$ For each j' we define the profile of the small tasks using $e_{j,j'}$ and none of the edges $e_{j,1}, ..., e_{j,j'-1}$ by $f_{j'}(e) := p\left(T'_j \cap T_S \cap T_e \cap T_{e_{j,j'}} \setminus T_{e_{j,j'-1}}\right)$ and a corresponding *overestimating* profile $\hat f_{j'}(e) := \left\lceil \frac{f_{j'}(e)}{\delta^4 U/16} \right\rceil \delta^4 U/16$. The latter has $O(1/\delta^4)$ many steps for each j' so we guess all profiles $\hat f_{j'}(e)$ in time $n^{O(1/\delta^6)}$. We compute tasks for the profiles similarly as in Lemma 8.

▶ **Lemma 17.** *Let $j' \in \mathbb{N}$. There is an algorithm with a running time of $n^{O(1/\delta^4)}$ that computes a set of tasks $\bar T_{j,j'} \subseteq T_S \cap T_{e_{j,j'}} \setminus T_{e_{j,j'-1}}$ with $p(\bar T_{j,j'} \cap T_e) \leq \hat f_{j'}(e) + \delta^4 U/16$ for each edge e, and $w(\bar T_{j,j'}) \geq w(T'_j \cap T_S \cap T_{e_{j,j'}} \setminus T_{e_{j,j'-1}})$.*

Then we add all small tasks whose path is contained in E_j and that do not use any of the special edges $e_{j,1}, e_{j,2}, ...$. Let $\bar T_j$ denote the selected small tasks. If all our guesses were correct then one can show that on each edge e we have that $p(\bar T_j \cap T_e) \leq (1+\delta)u_e$. Finally, we invoke a dynamic program that selects a maximum weight set of large tasks that fits in the remaining edge capacities, similarly as in Lemma 9. We define $T'_L = T' \cap T_L$.

▶ **Lemma 18.** *There is an algorithm with a running time of $n^{O(1/\delta^3)}$ that computes a set of tasks $\bar T_L \subseteq T_L$ such that $p((\bar T_L \cup \bar T_j) \cap T_e) \leq (1+\delta)u_e$ for each edge $e \in E$ and $w(\bar T_L) \geq w(T'_L)$.*

In fact, using Lemma 14 one can show that our algorithm for UFP for a constant range of edge capacities is already sufficient for general caching under resource augmentation.

▶ **Theorem 19.** *For any $\delta > 0$ there is an algorithm with a running time of $n^{(1/\delta)^{O(1)}}$ for general caching instances $(\mathcal{P}, \mathcal{R}, M)$ where $w_i = p_i$ for each page $i \in \mathcal{P}$ that computes a solution whose cost is at most the cost of the optimal solution and that is feasible if the cache has size $(1+\delta)M$.*

For the case of arbitrary edge capacities in UFP we can use a similar generalization as for UFP-cover to obtain the following theorem.

▶ **Theorem 20.** *For any $\delta > 0$ there is an algorithm with a running time of $n^{(1/\delta)^{O(1)}}$ for the case of UFP where $w_i = p_i$ for each task i, that computes a solution T' with $w(T') \geq w(OPT)$ and that satisfies that $p(T_e \cap T') \leq (1+\delta)u_e$ for each edge e.*

3 Task costs proportional to area

In this section we study the case of UFP-cover in which $w_i = |P_i| \cdot p_i$ for each task $i \in T$. For the respective case of UFP a PTAS is known [17] that uses a sparsification step. For UFP-cover we cannot use this technique, however, in the next theorem we extend the methodology in [17] to a $(1+\epsilon)$-approximation for UFP-cover under resource augmentation if the edge capacities are in a constant range (its proof is deferred to the full version of the paper). Suppose that $u_e \in [\delta U, U)$ for some global value U. We define a task i to be *small* if $p_i \leq \delta^3 \cdot U$ and *large* otherwise, denote by T_S and T_L the respective sets of tasks.

▶ **Theorem 21.** *Let $\epsilon, \delta > 0$ and $U > 0$. Given an instance of UFP-cover for the case that $w_i = |P_i| \cdot p_i$ for each task $i \in T_S$ and $u_e \in \{0\} \cup [\delta U, U)$ for each edge e. There is an algorithm with a running time of $n^{O_{\epsilon,\delta}(1)}$ that computes a solution T' with $p(T' \cap T_e) \geq (1-\delta)u_e$ for each edge e and such that $w(T') \leq w(T_L \cap OPT) + (1+\epsilon)w(T_S \cap OPT) \leq (1+\epsilon)OPT$.*

Next, we construct a PTAS *without* resource augmentation for a constant range of edge capacities. The PTAS will use the algorithm due to Theorem 21 as a black-box. Recall that in OPT each edge is used by at most $O(1/\delta^3)$ large tasks (see Lemma 4). We prove that there is a solution OPT' with $w(OPT') \leq (1+O(\delta))w(OPT)$ such that for every edge $e \in E$ it holds that either OPT' contains all small tasks using e or OPT' covers e to an extent of at least $(1+\delta^2)u_e$. Intuitively, we construct OPT' by taking OPT and adding tasks from T_S greedily until the property is satisfied. We show that in this way each edge is used by additional tasks of size at most $O(\delta^2)$. Hence, the total cost of these additional tasks is at most $O(\delta^2 U)|E|$ while $w(OPT) \geq \delta U|E|$.

▶ **Lemma 22.** *There is a set OPT' such that $w(OPT') \leq (1+O(\delta))w(OPT)$ and on each edge $e \in E$ it holds that $p(OPT' \cap T_e) \geq (1+\delta^2)u_e$ or $T_e \cap T_S \subseteq OPT'$.*

Let E' denote the set of edges e such that OPT' contains all small tasks using e. Using a standard dynamic program we guess the edges E' step by step (see the full version of the paper for details). Then, for each edge $e \in E'$ we guess the $O(1/\delta^3)$ large tasks in $OPT' \cap T_e$ and we selects all small tasks in $T_e \cap T_S$. Observe that in this way we select all tasks in OPT' that use some edge in E'. For each subpath E'' between two consecutive edges $e_1, e_2 \in E'$ we compute a set of tasks that cover the remaining demand on E'' (i.e., the demand not covered by the tasks in $(T_{e_1} \cup T_{e_2}) \cap OPT'$). To this end, we invoke the algorithm due to Theorem 21 on an auxiliary instance defined as follows. We start with E and contract all edges that are not in E''. If for some edge $e \in E''$ the tasks in $(T_{e_1} \cup T_{e_2}) \cap OPT'$ already cover e, i.e., if $p(T_e \cap (T_{e_1} \cup T_{e_2}) \cap OPT') \geq u_e$, then we contract e as well. Otherwise, note that OPT' covers e to an extent of at least $(1+\delta^2)u_e$ and therefore the tasks in $OPT' \setminus (T_{e_1} \cup T_{e_2})$ cover e to an extent of at least $\bar{u}_e := (1+\delta^2)u_e - p\left(T_e \cap (T_{e_1} \cup T_{e_2}) \cap OPT'\right)$. Therefore, we define the demand of each edge e to be $\bar{u}_e$. One can show that $\bar{u}_e \in [\delta^2 U, U)$ (since we did not contract e). Let T' denote the solution that we obtain if we apply the algorithm due to Theorem 21 on this instance. One can show that then $T' \cup ((T_{e_1} \cup T_{e_2}) \cap OPT')$ covers the original demand u_e on each edge $e \in E''$, i.e., $p(T_e \cap T' \cup ((T_{e_1} \cup T_{e_2}) \cap OPT')) \geq u_e$ for each edge $e \in E''$. We apply this routine on each subpath E'' between two consecutive edges in E'.

Above, we constructed OPT' by adding tasks to OPT in order to create some slack. If instead we remove tasks from OPT we can construct a solution OPT'' which is cheaper than OPT, feasible under resource augmentation, and in which on each edge e we removed small tasks of total size $\delta^2 u_e$ or we removed all small tasks using e. Being guided by OPT'' instead of OPT' we can construct an algorithm that computes a solution of cost at most OPT that is feasible under resource augmentation.

▶ **Theorem 23.** *Consider the case of UFP-cover where $w_i = |P_i| \cdot p_i$ for each task i and in which the edge capacities are in a constant range. This case admits a PTAS and for any $\delta > 0$ there is an algorithm with a running time of $n^{(1/\delta)^{O(1)}}$ that computes a solution T' with $w(T') \leq w(OPT)$ and that satisfies that $p(T_e \cap T') \geq (1-\delta)u_e$ for each edge e.*

4 Reducing to a constant range of edge demands

We describe a black-box procedure that intuitively turns an α-approximation algorithm for the case of UFP-cover of a bounded range of edge demands (independently of the costs of the tasks) into an $\alpha(1+\epsilon)$-approximation algorithm under $(1-\delta)$-resource augmentation for arbitrary edge demands. As a technicality, we need that the former algorithm works on instances with some normal tasks and some artificial huge tasks T_H where each task $i \in T_H$

has the property that it alone covers the complete demand on each edge of its path P_i and we require that the algorithm loses the factor of α only on the cost of the normal tasks. Note that any restriction on the set of normal tasks, like cost proportional to the area or to the size, is preserved in the reduction.

▶ **Lemma 24.** *Let $\epsilon, \delta > 0$ and given an instance (T, E, u) of UFP-cover. Suppose there is a polynomial time approximation algorithm for instances of the form $(T \cup T_H, E', \bar{u})$ where $E' \subseteq E$ and there is a value U such that for each edge $e \in E'$ it holds that $\delta^2 U \leq \bar{u}_e \leq U$ or that $\bar{u}_e = M := 1 + \sum_{i \in T} p_i$ and that $p_{i'} = M$ for each $i' \in T_H$ and assume that this algorithm computes solutions of cost at most $w(OPT \cap T_H) + \alpha w(OPT \cap T)$. Then there is a polynomial time algorithm that computes a solution $T' \subseteq T$ for (T, E, u) with $w(T') \leq \alpha(1+\epsilon)OPT$ and $p(T_e \cap T') \geq (1-\delta)u_e$ for each edge e.*

Proof sketch. Similarly as in the algorithm for UFP-cover in Section 2 we group the edges into levels according to the extent by which they are covered in OPT. For each edge $e \in E$ we define $\hat{u}_e := p(OPT \cap T_e)$ and we define its level $\ell(e)$ to be the integer ℓ such that $\hat{u}_e \in [(1/\delta)^\ell, (1/\delta)^{\ell+1})$. For each task $i \in T$ we define its level $\ell(i) := \min_{e \in P_i} \ell(e)$. Let $T^{(\ell)}$ denote the tasks of level ℓ and let $E^{(\ell)}$ denote the edges of level ℓ. We assign the tasks into groups $\mathcal{T}^{(s)}$ such that each group contains the tasks from $1/\epsilon + 1$ consecutive levels, intuitively most tasks are contained in exactly one group and some few tasks are contained in two groups. For an offset $\beta \in \{0, ..., 1/\epsilon - 1\}$ to be defined later we define $\mathcal{T}^{(s)} := \bigcup_{\ell=\beta+s/\epsilon}^{\beta+(s+1)/\epsilon} T^{(\ell)}$ for each $s \in \mathbb{Z}$. In particular, for each $s \in \mathbb{Z}$ the tasks in $T^{(\beta+(s+1)/\epsilon)}$ are contained in two groups, $\mathcal{T}^{(s)}$ and $\mathcal{T}^{(s+1)}$. By a shifting argument there is a choice for β such that $\sum_s w(\mathcal{T}^{(s)} \cap OPT) \leq (1+\epsilon)OPT$. Similarly, we group the edges into groups. However, now each edge will be contained in only one group. For each $s \in \mathbb{Z}$ we define $\mathcal{E}^{(s)} := \bigcup_{\ell=\beta+s/\epsilon+1}^{\beta+(s+1)/\epsilon} E^{(\ell)}$. One key observation is that due to the resource augmentation the tasks in $\mathcal{T}^{(s)} \cap OPT$ are sufficient to cover the demand of all edges in $\mathcal{E}^{(s)}$.

▷ Claim 25. Let $s \in \mathbb{N}$. For each edge $e \in \mathcal{E}^{(s)}$ it holds that $T_e \cap \mathcal{T}^{(s)} \cap OPT \geq \hat{u}_e(1 - O(\delta))$.

Hence, if we knew the level of each edge then we could generate one subinstance for each group s whose input contains only the edges $\mathcal{E}^{(s)}$ and the tasks in $\mathcal{T}^{(s)}$ and then take the union of these solutions. Unfortunately, we do not know the levels of the edges. Instead, we define a dynamic program. Our DP has a cell (E', s) for each $E' \subseteq E$ and each level $s \in \mathbb{Z}$. For each $s \in \mathbb{Z}$ let $\mathcal{P}(s)$ denote (unknown) the maximal subpaths consisting of edges in $\bigcup_{s':s' \geq s} \mathcal{E}^{(s')}$.

Consider a cell (E', s), where the reader may imagine that $E' \in \mathcal{P}(s)$. The goal is to compute a set of tasks that cover E' where we restrict ourselves to solutions such that each edge $e \in E'$ is covered to an extent of at least $(1-\delta)(1/\delta)^{\beta+s/\epsilon+1}$ (even though the real demand u_e of e might be smaller). Note that if $E' \in \mathcal{P}(s)$ then $p(T_e \cap \{i \in OPT \cap \bigcup_{s'=s}^{\infty} \mathcal{T}^{(s')} | P_i \cap E' \neq \emptyset\}) \geq (1-\delta)(1/\delta)^{\beta+s/\epsilon+1}$, for each $e \in E'$. Inductively, assume that there is a value s such that for each $E' \in \mathcal{P}(s')$ with $s' > s$ the DP-cell (E', s') stores a solution of cost at most $\alpha(1+\epsilon)w\left(\{i \in OPT \cap \bigcup_{s'=s+1}^{\infty} \mathcal{T}^{(s')} | P_i \cap E' \neq \emptyset\}\right)$ that covers each edge $e \in E'$ to an extent of at least $u_e(1-\delta)$. Ideally, we would like to guess the paths $E'' \subseteq E'$ such that $E'' \in \mathcal{P}(s+1)$, take the solutions stored in the respective cells $(E'', s+1)$, and then call the α-approximation on the remaining edges on E''. However, the number of such paths E'' can be too large. Instead, for each solution in a cell $(E'', s+1)$ with $E'' \subseteq E'$ we introduce an artificial input task $i(E'', s+1)$ with $p_{i(E'',s+1)} = M$ and the weight $w_{i(E'',s+1)}$ of $i(E'', s+1)$ is defined as the cost of the solution stored in the cell $(E'', s+1)$. Let T_H denote the set of all these (polynomially many) artifical tasks. We

define demand of the edges such that for each edge $e \in E'$ we define its demand $\bar{u}_e$ to be $\bar{u}_e := (1-\delta)(1/\delta)^{\beta+s/\epsilon+1}$ if $u_e < (1/\delta)^{\beta+s/\epsilon+1}$ (recall that we restrict ourselves to solutions that cover at least this amount), $\bar{u}_e := (1-\delta)u_e$ if $(1/\delta)^{\beta+s/\epsilon+1} \leq u_e < (1/\delta)^{\beta+(s+1)/\epsilon+1}$, and $\bar{u}_e := M$ if $u_e \geq (1/\delta)^{\beta+(s+1)/\epsilon+1}$. Observe that if $u_e \geq (1/\delta)^{\beta+(s+1)/\epsilon+1}$ then $\ell(e) > s$ and hence the tasks in $OPT \cap \mathcal{T}^{(s)}$ are not needed for covering e (due to the resource augmentation). Note that one solution to this subproblem is to select the task $i(E'', s+1)$ for each $E'' \in \mathcal{P}(s+1)$ and the tasks in $OPT \cap \mathcal{T}^{(s)}$ that use E'. From Claim 25 we know that the tasks in sets $\mathcal{T}^{(s')}$ with $s' < s$ are not needed to cover the demand in E'. We invoke the α-approximation algorithm and store the solution in the cell (E', s'). One can show that the cell $(E, -1)$ then stores a $\alpha(1+\epsilon)$-approximative solution T' that satisfies $p(T_e \cap T') \geq (1-\delta)u_e$ for each $e \in E$. ◀

▶ **Corollary 26.** *Consider the case of UFP-cover where $w_i = |P_i| \cdot p_i$ for each task i. For any $\epsilon, \delta > 0$ there is a polynomial time algorithm that computes a solution T' with $w(T') \leq (1+\epsilon)w(OPT)$ and that satisfies that $p(T_e \cap T') \geq (1-\delta)u_e$ for each edge e.*

5 Conclusion and open problems

In this paper we studied approximation algorithms for UFP-cover under resource augmentation. We gave algorithms for the cases where the task costs are proportional to their "areas" or sizes, computing solutions whose costs are $(1+\epsilon)$-approximate or even optimal, respectively. It is an open question whether one can obtain such algorithms also for the general case under resource augmentation. An interesting first step would be to get an algorithm for this setting with a better approximation ratio than 4.

Our results imply that under resource augmentation UFP-cover is no longer NP-hard if the cost of each task equals its size (unless P = NP), and the same holds for the respective settings of UFP (packing) and general caching. This raises the question whether the general case of these problems is still NP-hard in the resource augmentation setting or whether one can compute a solution with optimal cost in polynomial time. Another natural open question is whether one can obtain results like ours also *without* resource augmentation. We hope that our new techniques help constructing such algorithms.

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. Nikhil Bansal and Kirk Pruhs. The geometry of scheduling. *SIAM Journal on Computing*, 43(5):1684–1698, 2014.

6 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, September 2001. doi:10.1145/502102.502107.

7 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.

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

9 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 2005.

10 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.

11 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.

12 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.

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

14 Marek Chrobak, H Karloof, Tom Payne, and S Vishwnathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

15 Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

16 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.

17 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.

18 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. A $(5/3+\varepsilon)$-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.

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

20 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.

21 Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.