A QPTAS for the General Scheduling Problem with Identical Release Dates*

Antonios Antoniadis[1], Ruben Hoeksma[2], Julie Meißner[3], José Verschae[4], and Andreas Wiese[5]

1 **Department of Computer Science, University of Bonn, Bonn, Germany**
antoniad@cs.uni-bonn.de
2 **Center for Mathematical Modeling, Universidad de Chile, Santiago, Chile**
rhoeksma@dim.uchile.cl
3 **Institut für Mathematik, Technische Universität Berlin, Berlin, Germany**
jmeiss@math.tu-berlin.de
4 **Facultad de Matemáticas & Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Santiago, Chile**
jverschae@uc.cl
5 **Department of Industrial Engineering & Center for Mathematical Modeling, Universidad de Chile, Santiago, Chile**
awiese@dii.uchile.cl

Abstract

The General Scheduling Problem (GSP) generalizes scheduling problems with sum of cost objectives such as weighted flow time and weighted tardiness. Given a set of jobs with processing times, release dates, and job dependent cost functions, we seek to find a minimum cost preemptive schedule on a single machine. The best known algorithm for this problem and also for weighted flow time/tardiness is an $O(\log \log P)$-approximation (where P denotes the range of the job processing times), while the best lower bound shows only strong NP-hardness. When release dates are identical there is also a gap: the problem remains strongly NP-hard and the best known approximation algorithm has a ratio of $e + \epsilon$ (running in quasi-polynomial time). We reduce the latter gap by giving a QPTAS if the numbers in the input are quasi-polynomially bounded, ruling out the existence of an APX-hardness proof unless $\mathsf{NP} \subseteq \mathsf{DTIME}(2^{\mathrm{polylog(n)}})$. Our techniques are based on the QPTAS known for the UFP-Cover problem, a particular case of GSP where we must pick a subset of intervals (jobs) on the real line with associated heights and costs. If an interval is selected, its height will help cover a given demand on any point contained within the interval. We reduce our problem to a *generalization* of UFP-Cover and use a sophisticated divide-and-conquer procedure with interdependent non-symmetric subproblems.

We also present a pseudo-polynomial time approximation scheme for two variants of UFP-Cover. For the case of agreeable intervals we give an algorithm based on a new dynamic programming approach which might be useful for other problems of this type. The second one is a resource augmentation setting where we are allowed to slightly enlarge each interval.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Generalized Scheduling, QPTAS, Unsplittable Flows

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.31

* This work was partially funded by Nucleo Milenio Información y Coordinación en Redes ICM/FIC RC130003, Conicyt PII Nr 20150140, and Fondecyt Nr 11140579.

44th International Colloquium on Automata, Languages, and Programming (ICALP 2017).
Editors: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl;
Article No. 31; pp. 31:1–31:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *General Scheduling Problem* (GSP) considers scheduling jobs with job dependent cost functions in a very general setting. We are given a single machine and a set of jobs J, where each job j has a release date $\rho_j \in \mathbb{N}$, a processing time $p_j \in \mathbb{N}$, and a cost function $f_j : \mathbb{N} \to \mathbb{N}_0 \cup \{\infty\}$ that is non-decreasing. The goal is to find a preemptive schedule on the machine that minimizes the total cost $\sum_{j \in J} f_j(C_j)$, where C_j is the completion time of job j in the computed schedule.

With arbitrary cost functions for the jobs, we have a lot of modeling power, which we believe makes the problem worth studying. In fact, we can model many scheduling objectives that were also studied separately, such as weighted flow time (each job j has weight w_j and $f_j(C_j) = w_j(C_j - \rho_j)$) or weighted tardiness (each job j additionally has a deadline d_j and $f_j(C_j) = \max\{w_j(C_j - d_j), 0\}$). The best known result for GSP is a $O(\log \log P)$-approximation [5] (where P denotes the range of the processing time) and no better polynomial time results are known for any of the mentioned special cases. The best known lower bound shows only strong NP-hardness [13] (even in the case without release dates), thus leaving a large gap compared to the $O(\log \log P)$-approximation. Even if all jobs have identical release dates there is a gap in our understanding: the best known results are a $(4 + \epsilon)$-approximation in polynomial time [11] and an $(e + \epsilon)$-approximation in quasi-polynomial time [12]. It is open whether this case is APX-hard. In this paper we settle the latter question: for GSP with identical release dates we present a QPTAS, i.e., a $(1 + \epsilon)$-approximation algorithm with a running time of $n^{\log(n)^{O(1)}}$ for any constant $\epsilon > 0$. This implies that the problem is *not* APX-hard, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(2^{\mathrm{poly}(\log n)})$.

In this extended abstract, many proofs and details had to be omitted due to space constraints.

1.1 General Scheduling Problem and UFP-Cover

For identical release dates, GSP is purely a sequencing problem, since a solution cannot profit from preempting jobs or leaving idle-time. Assuming that $\rho_j = 0$ for each j, the whole schedule finishes at time $T := \sum_j p_j$. Using the viewpoint from [5], we can see this problem as a covering problem. In any feasible solution, for each time t, we need that the total processing time of jobs finishing after time t is at least $D_t := T - t$. We can think of D_t as the *demand* of time point t. Now, we rephrase the problem as follows. For each job j select a completion time C_j such that for each t' the total processing time of the jobs unfinished at time t' is at least $D_{t'}$. We say that job j is *unfinished* or *active* during the interval $[0, C_j)$. An easy proof shows that, for each such choice of completion times, there exists a schedule in which every job j is finished by its completion time C_j [5].

An important special case arises when the cost function f_j of each job j attains only one of three values: zero in an interval $[0, r_j)$ (r_j should not to be confused with the release date $\rho_j = 0$), a job dependent value c_j in an interval $[r_j, d_j)$, and ∞ in $[d_j, \infty)$. In this setting, we can assume that the optimal solution selects either $[0, r_j)$ or $[0, d_j)$ to be the interval during which j is active. Moreover, we can simply remove p_j from the demand at each time $[0, r_j)$, which leaves as the only decision for j whether we pay c_j and cover p_j units of demand during $[r_j, d_j)$, or not. Thus, this special case can be reduced to the *Unsplittable Flow on a Path (UFP)-Cover* problem. In UFP-Cover, we are given a set of jobs J, each job described by a cost c_j, a size p_j, and the interval $[r_j, d_j)$ and, for each time point t, a demand D_t. The goal is to select a subset J' of the jobs such that, for each time point t, the total size of the jobs $j \in J'$ with $t \in [r_j, d_j)$ is at least D_t. Note that we do not

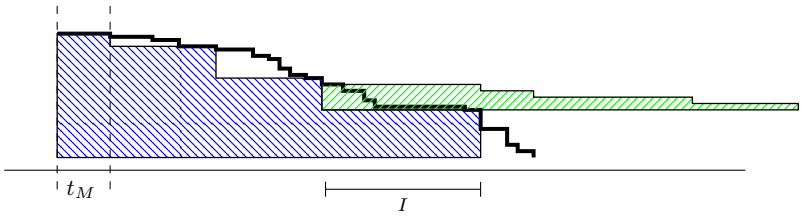


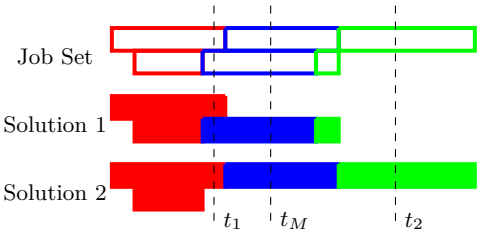
Figure 1 The bold curve denotes the size profile of job parts selected by the optimal solution that cross t_M. The blue step function shows an underestimating profile that approximates the former curve. The height of the green area (subprofile) is an (under-)estimation of the size of job parts in the optimal solution for all jobs that have a part covering t_M and whose right end point lies at interval I (i.e., the fourth step of the blue function).

require that the demand function D_t is non-increasing (but by adding jobs of zero cost one could assume this w.l.o.g.). The best known results for UFP-Cover are a 4-approximation in polynomial time [6, 8] and a QPTAS which requires the input data to be quasi-polynomially bounded [12]. Since there is the QPTAS, it is natural to conjecture that also a PTAS exists. In this paper, we make progress towards this by presenting pseudo-polynomial time approximation schemes for the settings of agreeable intervals, i.e., when for any two jobs j, j' we have that $r_j \leq r_{j'} \Rightarrow d_j \leq d_{j'}$, and for a resource augmentation setting, where we are allowed to increase each given interval $[r_j, d_j)$ by a factor of $1 + \mu$ for an arbitrarily small $\mu > 0$ while the compared optimal solution cannot do this.

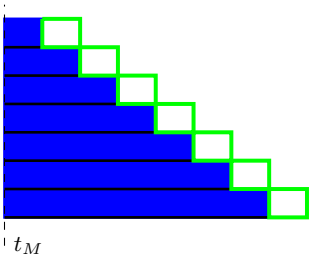
1.2 Our Contribution

Our first result is a QPTAS for GSP with identical release dates, assuming that all numbers in the input are quasi-polynomially bounded. We reduce GSP to a generalization of UFP-Cover. This *generalized UFP-Cover* problem is defined like regular UFP-Cover, but now each job j consists of K *parts*. More precisely, for each job j we are given an integral starting time r_j, a size p_j, up to K many integral end times $r_j < d_j^1 < d_j^2 < ... < d_j^K$, and corresponding accumulated costs $c_j^1, c_j^2, ..., c_j^K$. Jobs can be selected or not. If a job is not selected its cost is zero and it does not contribute to cover any demand. If job j is selected, we can choose to *extend it up to any part* $i \in \{1, \dots, K\}$, which means that then it is active during $[r_j, d_j^i)$. In this case we pay c_j^i for this job while it contributes to cover p_j units of demand to each time within $[r_j, d_j^i)$. The objective is to cover all demand D_t while minimizing the total cost. Notice that if $K = 1$ then we recover the UFP-Cover problem. On the other hand, we show that by losing a factor of $1 + \epsilon$ in the objective we can assume that $K = 1/\epsilon^2$.

Starting with this, we extend the known QPTAS for UFP-Cover, which works as follows. We consider the jobs crossing the middle time point t_M; denote them by J_M. They are split into $(\log n)^{O_\epsilon(1)}$ groups according to size and cost. For each group and each time t, we consider the total size of the jobs in the group crossing time point t in the optimal solution. This yields a function that is increasing from time 0 to time t_M, and decreasing from t_M to T. This function can be underestimated by a step-function (*profile*) with $O(1/\delta)$ (where $\delta = O_\epsilon(1)$) many steps (see the blue curve in Figure 1). One first guesses the step-function and then selects jobs that cover the demand given by this step-function greedily (which is essentially optimal). There is still some error due to the fact that the step-function underestimates the true amount that jobs in $\mathrm{OPT} \cap T_M$ cover on each time point. In the case of regular UFP-Cover, one can compensate this error by greedily selecting jobs that were not yet selected.



(a) Picking the blue part of the top job rules out Solution 1, while not picking it rules out Solution 2. Note that both solutions are incomparable since on t_1 and t_2 they cover different amounts of demand.



(b) Assume that the optimal solution selects all blue job parts (crossing t_M). Then there are still an exponential number of options for which jobs we should also select the green parts. Thus, we cannot take this decision immediately.

Figure 2 Locally Pareto-optimal choices.

In contrast to regular UFP-Cover, this approach fails for our generalization. We can think of each job as a collection of *parts* $[r_j, d_j^1), [d_j^1, d_j^2), ..., [d_j^{K-1}, d_j^K)$. The step function can only be guessed for the part that actually covers t_M. Yet, if we select that part, we need to pick all preceding parts of that job as well. This influences our options on the left side of t_M. On the other hand, if we do not pick the part that covers t_M of a certain job, succeeding parts of that job cannot be picked. This influences our options on the right side of t_M (see Figure 2a).

To address these issues we guess more fine-grained underestimating profiles. We group the jobs further such that for each job in a group the same part crosses t_M. Assume that for the jobs in the considered group their respective i-th part, $[d_j^{i-1}, d_j^i)$, covers t_M. We guess a *right underestimating profile* that estimates the total size covered to the right of t_M by these i-th parts that are selected by OPT. This profile partitions the jobs into subgroups according to the "step" of the profile in which the i-th part ends (see the green curve in Figure 1). For each of these constantly many groups we create a *subprofile* which underestimates the additional demand covered by the $(i+1)$-th parts of those jobs in OPT, i.e., by the intervals $[d_j^i, d_j^{i+1})$. We continue recursively and create underestimating subprofiles for all parts of the jobs, which gives a tree structure. We refer to this construction as *tree profiling*.

The tree profiling yields constantly many subgroups of jobs. For each of them we guess the number of jobs that the optimal solution selects (recall that the jobs in the same group have essentially the same size and cost). Then we recurse *only on the left* subproblem in which we want to cover the demand of the interval $[0, t_M)$ subject to the new constraint that from each subgroup we select the previously guessed number of jobs. Once we have a solution to this left subproblem, ideally we would like to decide how many parts of the jobs crossing t_M we select, i.e., the parts laying in the interval $[t_M + 1, T)$. Unfortunately, there can still be very many Pareto-optimal choices for this (see Figure 2b for an example). This can even happen when taking into account the information from the tree profiling. Instead, at this point we select for each job only the part that crosses t_M and we decide later about the additional parts we want to select. We recurse on the interval $[t_M + 1, T)$ and the remaining problem is to cover the demand of the interval $[t_M + 1, T)$ while we can select additional parts from the jobs that we selected already. In each subproblem we recurse on the respective middle time point, which yields a recursion depth of $O(\log n)$ and thus quasi-polynomial running time overall.

UFP-Cover for agreeable deadlines. Our second result is a pseudo-polynomial time $(1+\epsilon)$-approximation for UFP-Cover with agreeable deadlines. We first present an exact pseudo-polynomial time dynamic program (DP) for the case that the interval of each job is of the form $[0, d_j)$ or $[r_j, T)$, i.e., $r_j = 0$ or $d_j = T$. Then, we generalize this to the case where there are $1/\epsilon$ intervals $[T_0, T_1)$, $[T_1, T_2)$, ..., $[T_{1/\epsilon-1}, T_{1/\epsilon})$ and for each job j we have that $[r_j, d_j) \cap [T_\ell, T_{\ell+1})$ equals either $[r_j, T_{\ell+1})$ or $[T_\ell, d_j)$. Using the fact that the job deadlines are agreeable we can show that the time axis can be partitioned into a possibly superconstant number of intervals $[T_\ell, T_{\ell+1})$ with this property. By losing only a factor $1+\epsilon$ in the objective, we can split those into groups of at most $1/\epsilon$ consecutive intervals, each of which then yields an independent subinstance of our problem on which we apply our DP. The backbone of the latter is that the agreeable-deadlines property yields an ordering to process the jobs such that we need to remember only little information about the previously chosen jobs. We believe that this ordering and the resulting DP technique might be useful for other problems on agreeable intervals as well. Note that the opposite case where the job intervals form a laminar family has a simple exact DP. Thus, we can now handle the two "extreme" cases of the problem.

PTAS under resource augmentation. For UFP-Cover we present a pseudopolynomial time PTAS for the setting where we can enlarge each job interval $[r_j, d_j)$ by a factor $1+\mu$ for some $\mu > 0$, i.e., replace it by the interval $[r_j - \frac{\mu}{2}(d_j - r_j), d_j + \frac{\mu}{2}(d_j - r_j))$, while the compared optimal solution does not have this privilege. We use this resource augmentation to discretize the begin and end points of the intervals of the jobs. As in a similar result for UFP-packing [3], we group the jobs by the lengths of their intervals. In UFP-packing, the grouping can be done such that two jobs in different groups have intervals whose lengths differ by a large factor. Then each group can be handled almost independently. In our case we cannot establish such a property, since it requires the removal of some jobs from the input, which in turn may make our instance of UFP-Cover infeasible. Instead, our DP needs to transfer a lot of information between groups. The key for our approach is to prove that for each group it is sufficient to remember information from *one* previous group.

1.3 Other related work

The General Scheduling Problem can model a vast class of well-studied objective functions. The known $O(\log \log P)$-approximation for it [5], is even the best known result for several important special cases. For example, for the weighted flow time objective there were previously algorithms known with approximation ratios of $O(\log^2 P)$, $O(\log W)$ and $O(\log nP)$ [4, 10], where P and W denote the ranges of the job processing times and weights, respectively. Also, there is a QPTAS with a running time of $n^{O_\epsilon(\log P \log W)}$ [9].

For GSP with identical release dates the first constant factor approximation is due to Bansal and Pruhs [5] and yields an approximation ratio of 16. This was later improved to $4+\epsilon$ [11] by adapting ideas from the 4-approximation algorithm for UFP-Cover [6, 8]. For UFP-Cover this is the best known polynomial time result, while for quasi-polynomially bounded input numbers the problem even admits a QPTAS, implying a quasi-polynomial time $(e+\epsilon)$-approximation for GSP with identical release dates [12]. The used techniques are based on a QPTAS for the packing version of UFP [3]. For the latter algorithm, one can even remove the assumption that the input data is quasi-polynomially bounded [7]. The best known polynomial time results for UFP-packing are a $(2+\epsilon)$-approximation [1] and PTASs for some special cases [7].

2 QPTAS for GSP with identical release dates

We present our QPTAS for the General Scheduling Problem with identical release dates. Throughout this section we assume that all input numbers are quasi-polynomially bounded integers, and that we are given an $\epsilon > 0$ such that $1/\epsilon$ is an integer. We assume as well that we are given the number $f_{\max} = \max\{f_j(t) : f_j(t) \neq \infty, t \leq T\}$ as part of the input. First, we simplify the input such that the job cost functions attain only values that are powers of $1+\epsilon$ or ∞.

▶ **Lemma 1.** *By losing a factor $1+\epsilon$ in the objective, we can assume for each job j and each t that $f_j(t) \in \{(1+\epsilon)^k | k \in \mathbb{N}_0\} \cup \{0, \infty\}$ and that f_j is a non-decreasing step function with $O_\epsilon(poly(\log n))$ steps. We can further assume that each f_j is given explicitly, even if in the input it was given via an oracle.*

As in [5] we interpret GSP as a covering problem. Given a demand D_t for each interval $[t, t+1)$ and a set of jobs J. Each job $j \in J$ is characterized by a size p_j, a set of parts with corresponding intervals $I_j^1 = [t_j^{(0)}, t_j^{(1)}), I_j^2 = [t_j^{(1)}, t_j^{(2)})$, ... for $t_j^{(0)} \leq t_j^{(1)} \leq t_j^{(2)} \leq \dots$ and cost values $0 \leq c_j^1 < c_j^2 < \dots$. The goal is to select for each job j a prefix of its parts, i.e., a value $\sigma(j) \in \mathbb{N}_0$ such that all parts $k \leq \sigma(j)$ are selected. The cost for j is then $c_j^{\sigma(j)}$. Possibly $\sigma(j) = 0$ and then no part is selected, and thus we define $c_j^0 := 0$ for each job j. For a solution σ we say that a job j is *active* at time t if $t \in \cup_{i=1}^{\sigma(j)} I_j^i$. We require that for each t the total size of the jobs active at t is at least D_t, i.e., $\sum_{j : t \in \cup_{i=1}^{\sigma(j)} I_j^i} p_j \geq D_t$. The objective is to minimize the total cost $\sum_{j \in J} c_j^{\sigma(j)}$. We call this problem the *generalized UFP-Cover problem* (regular UFP-Cover is the special case where each job has only one part).

Using a similar argumentation as in [5] we can prove the following lemma.

▶ **Lemma 2.** *For any instance of GSP with identical release dates in which each cost function attains only polynomially many different values, we can construct in polynomial time an instance of generalized UFP-Cover such that approximations are preserved, i.e., for any $\alpha \geq 1$ an α-approximate solution for the generalized UFP-Cover instance can be transformed in polynomial time to an α-approximate solution for the GSP instance.*

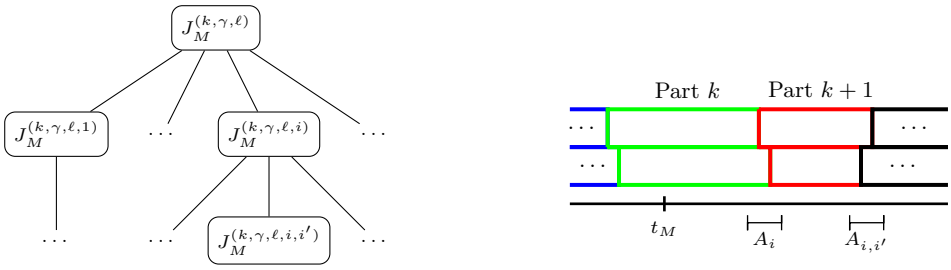
We apply Lemma 2 to reduce our given GSP instance to an instance of generalized UFP-Cover. Next, we ensure that each job has only $1/\epsilon^2$ parts.

▶ **Lemma 3.** *By losing a factor $1+\epsilon$ in the objective, we can assume that each job has at most $K := 1/\epsilon^2$ many parts, each value c_j^k is a power of $1+\epsilon$, and that $c_j^{k+1} = (1+\epsilon)c_j^k$ for each k.*

Assume w.l.o.g. that there is a value $T \leq \text{poly}(n)$ such that $I_j^k \subseteq [0, T)$ for each job j and each part k. Our algorithm is recursive. Let $t_M = \lceil \frac{T}{2} \rceil$ be the middle point of the interval $[0, T)$. The overall idea is to take a decision about the parts of jobs j that cover $[t_M, t_M + 1)$, i.e., such that $t_M \in I_j^k$ for some k, and then recursively decide on all job parts k' with $I_{j'}^{k'} \subseteq [0, t_M)$ (left subproblem) and k'' with $I_{j''}^{k''} \subseteq [t_M + 1, T)$ (right subproblem).

2.1 Tree profiling and grouping of jobs

Let $J_M \subseteq J$ denote the set of jobs j having a part k with $t_M \in I_j^k$. We partition J_M into a poly-logarithmic number of subsets according to their respective size, by the index of the



(a) Example of a tree $G^{(k,\gamma,\ell)}$. **(b)** Two jobs from group $J_M^{k,\gamma,\ell,i,i'}$.

Figure 3 Recursive partitioning of the jobs.

part that covers t_M, and by the cost of the latter. Formally, for numbers k, γ and λ we define sets

$$J_M^{(k,\gamma,\lambda)} := \{j \in J_M : t_M \in I_j^k,\ c_j^k = (1+\epsilon)^\gamma,\ \text{and } (1+\epsilon)^\lambda \le p_j < (1+\epsilon)^{\lambda+1}\}.$$

Consider one such group $J_M^{(k,\gamma,\lambda)}$. We want to partition it further. Let $\delta = \delta(\epsilon)$ be a small enough constant. First, we want to partition it into $O(1/\delta)$ subgroups $J_M^{(k,\gamma,\lambda,i)}$ such that: (i) OPT selects essentially the same number of jobs from each of these subgroups, and (ii) the k-th part of each job in the subgroup has a "similar" endpoint. Formally, we ensure the latter by partitioning the interval $[t_M, T)$ into subintervals $A_1, A_2, ...$ such that for each job j of a subgroup $J_M^{(k,\gamma,\lambda,i)}$ the k-th part ends in A_i (see Figure 3). Let $\bar{J}_M^{(k,\gamma,\lambda)}$ be the set of jobs $j \in J_M^{(k,\gamma,\lambda)} \cap OPT$ that OPT extends up to part k or further, i.e., for which OPT selects parts $I_j^1, ..., I_j^k$ and possibly more. To define our partition, we see that the respective k-th parts of the jobs in $\bar{J}_M^{(k,\gamma,\lambda)}$ cover some demand at each time point t, given by the function $\bar{f}_k(t) := \sum_{j \in \bar{J}_M^{(k,\gamma,\lambda)} : t \in I_j^k} p_j$. Observe that $\bar{f}_k$ is non-decreasing on $[0, t_M)$ and non-increasing on $[t_M, T)$. Ideally, we would like to guess $\bar{f}_k$ so that we have some idea about how much the k-th parts of the jobs in $J_M^{(k,\gamma,\lambda)}$ need to cover. Unfortunately, there are too many options on how $\bar{f}_k$ might look. Therefore, we guess $\left|\bar{J}_M^{(k,\gamma,\lambda)}\right|$ and a simpler underestimating function $\tilde{f}_k$ that approximates $\bar{f}_k$ sufficiently well, as given by the following lemma (see Figure 1). For our later purposes we need this function only on the interval $[t_M, T)$.

▶ **Lemma 4.** *There exists a function $\tilde{f}_k : [t_M, T) \to \{0, 1, ..., \sum_{j \in J} p_j\}$ such that $\tilde{f}_k$ is a step-function with at most $O(1/\delta)$ many steps, $\tilde{f}_k(t) \le \bar{f}_k(t) \le \tilde{f}_k(t) + \delta \cdot \left|\bar{J}_M^{(k,\gamma,\lambda)}\right| \cdot (1+\epsilon)^{\lambda+1}$, and $\tilde{f}$ is non-increasing.*

We use the function $\tilde{f}_k$ to split the set $J_M^{(k,\gamma,\lambda)}$ into subgroups, according to where the part k of each job $j \in J_M^{(k,\gamma,\lambda)}$ ends. Let $A_1, A_2, ...$ denote a partition of $[t_M, T)$ into $O(1/\delta)$ subintervals such that on each subinterval A_i the function $\tilde{f}_k$ is constant. For each such interval A_i we define $J_M^{(k,\gamma,\lambda,i)} \subseteq J_M^{(k,\gamma,\lambda)}$ to be the jobs $j \in J_M^{(k,\gamma,\lambda)}$ such that $(t_j^{(k)} - 1) \in A_i$ (recall that $I_j^k = [t_j^{(k-1)}, t_j^{(k)})$).

Subprofiles. It is convenient to think of a tree where $J_M^{(k,\gamma,\lambda)}$ forms the root node and the sets $J_M^{(k,\gamma,\lambda,i)}$ form the children of $J_M^{(k,\gamma,\lambda)}$. We take each such group $J_M^{(k,\gamma,\lambda,i)}$ and partition it further into $O(1/\delta)$ smaller subgroups $J_M^{(k,\gamma,\lambda,i,1)}, J_M^{(k,\gamma,\lambda,i,2)}, ...$ In the tree view, we can think of appending those as children to the node for the group $J_M^{(k,\gamma,\lambda,i)}$, see Figure 3. For

the subgroups, as before our goal is that OPT selects essentially the same number of jobs from each subgroup $J_M^{(k,\gamma,\lambda,i,i')}$ and that for each such subgroup the $(k+1)$-th part has a "similar" end point.

Recall that when we partitioned $J_M^{(k,\gamma,\lambda)}$ we estimated what the k-th part of the jobs in $J_M^{(k,\gamma,\lambda)} \cap OPT$ cover (via the function $\tilde{f}_k$) and obtained a grouping according to the steps of $\tilde{f}_k$. For the finer partitioning of $J_M^{(k,\gamma,\lambda,i)}$ we consider the jobs in $J_M^{(k,\gamma,\lambda,i)}$ for which OPT selects also the $(k+1)$-th part (and thus also the k-th part). Denote that set as $\bar{J}_M^{(k,\gamma,\lambda,i)}$. We define the function $\bar{f}_{k,i}(t)$ that models how much the k-th and the $(k+1)$-th parts of the jobs in $\bar{J}_M^{(k,\gamma,\lambda,i)} \cap OPT$ cover. Formally, $\bar{f}_{k,i}(t) := \sum_{j \in \bar{J}_M^{(k,\gamma,\lambda,i)} : t \in I_j^k \cup I_j^{k+1}} p_j$. We guess an underestimating function $\tilde{f}_{k,i}$ with the same properties as the function $\tilde{f}_k$ as given in Lemma 4, i.e., $\tilde{f}_{k,i}$ has $O(1/\delta)$ many steps, $\tilde{f}_{k,i}(t) \leq \bar{f}_{k,i}(t) \leq \tilde{f}_{k,i}(t) + O(\delta) \cdot |\bar{J}_M^{(k,\gamma,\lambda,i)}| \cdot (1+\epsilon)^{\lambda+1}$, and $\tilde{f}_{k,i}$ is non-increasing. Like before, the steps of $\tilde{f}_{k,i}$ yield a partition of $[t_M, T)$ into $O(1/\delta)$ many subintervals $A_{i,1}, A_{i,2}, \ldots$ such that $\tilde{f}_{k,i}$ is constant in each of them. Each such subinterval $A_{i,i'}$ yields a subgroup $J_M^{(k,\gamma,\lambda,i,i')}$ that contains all jobs $j \in J_M^{(k,\gamma,\lambda,i)}$ whose $(k+1)$-th part ends in $A_{i,i'}$, i.e., $(t_j^{(k+1)} - 1) \in A_{i,i'}$.

We continue recursively for K levels, expanding the tree accordingly. Analogous to before, we obtain for each node v in level k' of the tree (that is not a leaf), a subprofile function $\bar{f}_v$ and an approximate version $\tilde{f}_v$ such that $\tilde{f}_v(t) \leq \bar{f}_v(t) \leq \tilde{f}_v(t) + O(\delta) \cdot |\bar{J}_M^{(k,\gamma,\lambda,v)}| \cdot (1+\epsilon)^{\lambda+1}$, where $\bar{J}_M^{(k,\gamma,\lambda,v)}$ is the set of jobs in J_v that the optimum extends up to its $(k+k')$-th part. The leafs of the tree yield a partition of the job set, and the total number of nodes is $(1/\delta)^K$.

We can guess the whole partition in time $n^{(1/\delta)^{O(K)}}$ which will eventually be bounded by $n^{O_\epsilon(1)}$ (note that there are only $T \leq \text{poly}(n)$ options for each endpoint of an interval A_i or $A_{i,i'}$, etc.). In the same running time, we can guess for each arising group and subgroup the total number of jobs that OPT selects from these groups. More precisely, let $G^{(k,\gamma,\lambda)}$ be the resulting tree and for each node v denote by J_v the corresponding job group. For a node v on level k' we guess the value $N(v)$, the number of jobs in J_v that the optimum extends at least up to their respective $(k+k')$-th part.

We now bound the total demand deficit made by the underestimating functions. Let $f(t) = \sum_{j \in J_M^{(k,\gamma,\lambda)} : j \text{ active at } t \text{ in OPT}} p_j$ be the total size of jobs in $J_M^{(k,\gamma,\lambda)}$ that cover t in the optimal solution. We say that a solution is *concordant* with the tree $G^{(k,\gamma,\lambda)}$ and numbers $N(v)$ if, for each node v of each level k', the solution selects the $(k+k')$-th part of at least $(1+\epsilon)N(v)$ jobs in J_v, or of all jobs in J_v in case that $|J_v| < (1+\epsilon)N(v)$. As the next lemma shows, any tree concordant solution covers the demand at any point t almost as good as the optimal solution. The gap is bounded by $K \cdot \delta \cdot |\bar{J}_M^{(k,\gamma,\lambda)}| \cdot (1+\epsilon)^{\lambda+1}$ which is an upper bound on the sum of the deficits of all subprofiles relevant for a time point t. Here $\bar{J}_M^{(k,\gamma,\lambda)}$ is the subset of jobs in $J_M^{(k,\gamma,\lambda)}$ that OPT extends at least to the k-th part.

▶ **Lemma 5.** *Consider any solution concordant with tree $G^{(k,\gamma,\lambda)}$. The demand covered by such a solution at any time $t \in [t_M, T)$ is at least $f(t) - K \cdot \delta \cdot |\bar{J}_M^{(k,\gamma,\lambda)}| \cdot (1+\epsilon)^{\lambda+1}$.*

2.2 Fixing the demand deficit

We would like to recurse on the left and on the right subproblem, i.e., on $[0, t_M)$ and $[t_M + 1, T)$. We have guessed the correct number of jobs in each group but we have not decided yet which exact jobs from each group we want to select.

We deal with these issues as follows. Let us fix a tree $G^{(k,\gamma,\lambda)}$. We first consider any solution ALG that is concordant with the tree. By Lemma 5, this solution already covers almost all necessary demand, having a deficit of at most $\delta \cdot |\bar{J}_M^{(k,\gamma,\lambda)}| \cdot (1+\epsilon)^{\lambda+1}$ for every

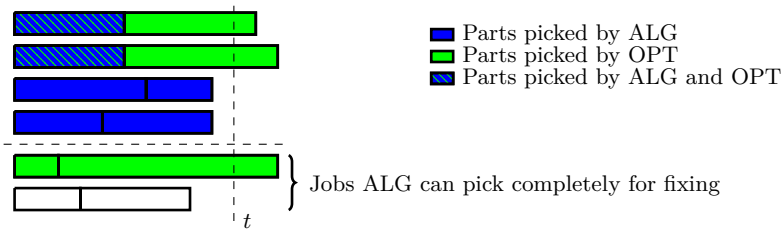


Figure 4 In contrast to the regular UFP-Cover Problem where selecting new jos is always sufficient, here this is not the case: even selecting all the new (bottom) jobs does not suffice to cover t! Instead, extending previously selected jobs is necessary.

time point in $[t_M, T]$. Even if we can fix this demand by adding more jobs (and we will, essentially, do so), picking an arbitrary concordant solution at this point will create issues for the left subproblem: nothing guarantees that the chosen solution we pick allows to cover the remaining demand within $[0, t_M)$ at a reasonable cost. To avoid this problem, we call the left subproblem recursively, giving the trees $G^{(k,\gamma,\lambda)}$ and numbers $N(v)$ for each node as input. We will require this problem to give us a solution ALG that is concordant with the tree for each group $J_M^{(k,\gamma,\lambda)}$ and that satisfies all demand at $[0, t_M)$. The exact way of solving this left subproblem is given in the next subsection. We call a solution constructed this way a *left-feasible* solution.

Consider now a left-feasible solution ALG and fix a tree $G^{(k,\gamma,\lambda)}$. The idea is to fix the deficit in $[t_M, T)$ by adding jobs picked greedily. As a first approach we could consider the following method: within all jobs in $J_M^{(k,\gamma,\lambda)}$ not active at t_M, pick the $\delta|\bar{J}_M^{(k,\gamma,\lambda)}|(1+\epsilon)$ ones that extend furthest to the right when all of their K parts are chosen. We denote by $H^{(k,\gamma,\lambda)}$ the set of these jobs. For any timepoint t that is covered by all these jobs, we will cover the whole deficit. Also, we can show that total incurred cost is at most an ϵ-fraction of the cost of $\mathrm{OPT} \cap \bar{J}_M^{(k,\gamma,\lambda)}$. One might be tempted to conclude that we are done: since we picked the jobs greedily, a time point t that is not covered by all jobs in $H^{(k,\gamma,\lambda)}$ cannot be covered by any other job that we did not make active at t_M. This is indeed enough to argue in the regular UFP-Cover problem [12]. However, the argument fails in our setting as we might still be able to further extend some jobs that our solution picks to cover t_M but not are not extended to cover t; see Figure 4.

To solve this issue we truncate ALG by removing for each group $J_M^{(k,\gamma,\lambda)}$ and each job $j \in J_M^{(k,\gamma,\lambda)}$ all parts that do not cover any point $t \in [0, t_M + 1)$. Let ALG_M be the truncated solution. We show that ALG_M plus all parts of all jobs in $H^{(k,\gamma,\lambda)}$ can be extended (by adding more parts, not necessarily like ALG) to a solution that covers all required demand and costs at most a $1+\epsilon$ factor more than OPT. The constructed solution covers all demand at times $[0, t_M + 1)$ and we will solve the remaining problem in the right subproblem.

To make this idea formal, denote by OPT_M the solution obtained by taking OPT and removing from it all parts I_j^k such that $I_j^k \subseteq [t_M + 1, T)$. For any left-feasible solution S we say that a solution S' is an *extension* of S if for each job j the solution S' extends j up to at least as many parts as S.

▶ **Lemma 6.** *Assume that $1/\delta = K \cdot \epsilon(1+\epsilon)^{O(K)}$. Suppose we are given the left-feasible truncated solution ALG_M. Then we can compute in polynomial time a set of jobs $H \subseteq J_M$ such that*

- *if we select all parts of each job in H this yields a total cost of at most $O(\epsilon) \cdot c(\mathrm{OPT}_M)$, and*
- *for the solution $ALG_M \cup H$ there is an extension ALG' such that $c(ALG') - c(ALG_M \cup H) \leq c(\mathrm{OPT}) - c(\mathrm{OPT}_M)$.*

Proof Sketch. Consider a set $J_M^{(k,\gamma,\lambda)}$. We consider all jobs of this set that are not covering t_M in ALG and sort them non-increasingly with respect to the right endpoint of I_j^K. Let $H^{(k,\gamma,\lambda)}$ be the set of the first $K \cdot \delta |\bar{J}_M^{(k,\gamma,\lambda)}|(1+\epsilon)$ such jobs and define $H = \cup_{k,\gamma,\lambda} H^{(k,\gamma,\lambda)}$. Notice that extending all parts of jobs in $H^{(k,\gamma,\lambda)}$ incurs a cost of at most $K \cdot \delta(1+\epsilon)^{K+1}(1+\epsilon)^{\gamma}|\bar{J}_M^{(k,\gamma,\lambda)}|$. By choosing the constants in the definition of δ appropriately we obtain that the cost is $O(\epsilon) \cdot c(\bar{J}_M^{(k,\gamma,\lambda)})$. Summing over all triplets k, γ, λ yields the desired bound on the total cost of H.

For a given set $H^{(k,\gamma,\lambda)}$, out of all right endpoints of jobs in the set, call τ_R the one most to the left. Inside the interval $[t_M, \tau_R)$ the jobs in $H^{(k,\gamma,\lambda)}$ cover at least $K(1+\epsilon)^{\lambda+1}\delta|\bar{J}_M^{(k,\gamma,\lambda)}|$, and thus they cover all deficit left by the solution ALG (or any other tree concordant solution). On the other hand, for any $t > \tau_R$ our greedy choice for $H^{(k,\gamma,\lambda)}$ guarantees that *all* jobs in $J_M^{(k,\gamma,\lambda)}$ that can be extended to cover t are taken at least up to their k-th part in $ALG_M \cup H$. This allows us to construct the claimed extension ALG' of $ALG_M \cup H$: we start with ALG and transform it step by step to make it resemble the optimal solution. Note that this is a purely existential result since we need to know the optimal solution for this procedure. ◀

2.3 Left subproblem

Suppose that via recursion we have computed a left-feasible solution ALG. Then, using Lemma 6 we compute the jobs H such that $c(H) \leq O(\epsilon) \cdot c(OPT_M)$ and such that the extension ALG' is guaranteed to exist. In order to compute (an approximation to) ALG' we recurse on the right subproblem, given by the interval $[t_M + 1, T)$.

For each $t \in [t_M + 1, T)$ we update the demand D_t to take into account that we already selected some job parts crossing t_M and the jobs in H. Formally, we define the new demands as $D'_t := D_t - \sum_{j:t\in\tilde{J}(t)} p_j$ where $\tilde{J}(t)$ denotes the set of jobs j such that $ALG_M \cup H$ contains a part of j that covers t. For each job j such that ALG_M selected the part I_j^k covering t_M, our subproblem only has the parts of j that lie completely within $[t_M + 1, T)$. We update their cost, taking into account that the left subproblem has already paid c_j^k for it, i.e., the cost value $\bar{c}_j^{\ell-k}$ for each new part $\ell - k$ is set to $\bar{c}_j^{\ell-k} = c_j^{\ell} - c_j^k$. This yields an instance of our problem on the interval $[t_M + 1, T)$ whose size is only half the size of the original interval $[0, T)$. Strictly speaking, the new costs might no longer be a power of $1 + \epsilon$. However, note that the adjustment of costs means that $\bar{c}_j^1 = c_j^{k+1} - c_j^k = \epsilon c_j^1$. Therefore, the costs of any two parts of a job still differ by at most a constant factor and the new cost values come from a set of size $O(\text{poly}(\log n))$ (which is important to bound the number of job groups $J_M^{(k,\gamma,\lambda)}$) . Moreover, this factor does not increase further in the recursion and hence we can recurse one the right subproblem with essentially the same routine as above.

It remains to describe how to recurse on the left subproblem for the interval $[0, t_M)$. Formally, this subproblem is defined as follows: we are given the interval $[0, t_M)$ together with the demand D'_t for each point $t \in [0, t_M)$ (the updated demand). Also, for each tree $G^{(k,\gamma,\lambda)}$ and each vertex v we are given a corresponding group of jobs J_v. Additionally we have to consider the set of all input jobs j such that no part of j crosses t_M - we refer to this set of jobs as J_L. Finally, for each group J_v we are given a value $N(v)$ that indicates that for at least $(1 + \epsilon)N(v)$ jobs in J_v we have to select the respective part that crosses t_M.

Our objective is to find a solution for jobs in $J_L \cup \bigcup_{k,\gamma,\lambda} J^{(k,\gamma,\lambda)}$ that covers all demand in $[0, t_M)$ and that is concordant for each tree. To have a cleaner subproblem, we observe that the leaves of $G^{(k,\gamma,\lambda)}$ imply a *partition* of the jobs of $J^{(k,\gamma,\lambda)}$ into subgroups. For each of them we guess how many jobs the optimal solution selects from the subset corresponding to that leaf. Then for each of them we require that the left subproblem selects either a factor

$1+\epsilon$ more jobs or all jobs from that subset. The resulting solution can easily be transformed into a concordant solution.

We consider the point $t'_M := \lceil \frac{t_M}{2} \rceil$. Like above, we partition the jobs into groups according to which part of them crosses t'_M. However, we do this separately for J_L and each subgroup of jobs crossing t_M. For each resulting separate group, we guess the profiles and recursive subprofiles as before. Once we have guessed this partition of the jobs together with the required number of jobs of each group, we recurse on the left-left problem, i.e., on the problem for the interval $[0, t'_M)$. When we obtained a solution for the left-left subproblem in the interval $[0, t'_M)$ we recurse further on the interval $[t'_M, t_M)$. For this left-right subproblem, we update the cost of the jobs whose respective parts crossing t'_M were selected by the left-left subproblem (like we did when we defined the right subproblem of the interval $[t_M + 1, T)$) and additionally impose the constraint that from each group J_v (as defined by the main subproblem for the interval $[0, T)$) for at least $(1+\epsilon)N(v)$ jobs we select the respective part crossing t_M.

Number of groups. We continue recursively in the same fashion. In the recursion, the number of job groups we pass to each subproblem increases since from the main subproblem for the interval $[0, T)$ we are given a partition into subgroups and whenever we recurse on a left subproblem these subgroups are partitioned further and also new subgroups are defined. However, we can show that in each step of the recursion the total number of arising subgroups is bounded by $(\frac{1}{\epsilon^2} \log n)^{O(K)}$.

▶ **Lemma 7.** *In the input of each subproblem arising in the recursion, the jobs are partitioned into at most* $(\frac{1}{\epsilon^2} \log n)^{O(K)}$ *different groups.*

Whenever we are given a subproblem on some interval I' then we guess subgroups and certain values with a quasi-polynomial number of options in total and we recurse on two subproblems, given by subintervals of I' whose size is half the size of I'. Thus, the recursion tree has a depth of $O(\log T) = O(\log n)$ and each internal node of the tree has at most quasi-polynomially many children. Hence, our algorithm has quasi-polynomial running time overall.

▶ **Theorem 8.** *There are quasi-polynomial time* $(1+\epsilon)$*-approximation algorithms for the general scheduling problem and for the generalized UFP-Cover problem, assuming that all input values are quasi-polynomially bounded integers.*

3 Agreeable Instances

In this section we present our pseudopolynomial-time $(1+\epsilon)$-approximation algorithm for the UFP-Cover problem on agreeable instances. We first show how to partition a given instance into smaller subinstances (Section 3.1). Then we then present our algorithm for a special type of subinstances (Section 3.2).

For simplicity of presentation, we will assume throughout this section w.l.o.g. that each integer timepoint t is associated with a demand D_t and that we only need to cover the demands at such timepoints. Furthermore, we assume w.l.o.g. that the the intervals defined by the release-time and deadline of each job are closed, i.e., have the form $[r_j, d_j]$. We also assume that all elements of the set $U := \cup_j \{r_j, d_j\}$ are disjoint. To simplify the presentation, we further assume w.l.o.g. that all elements of U are even integers.

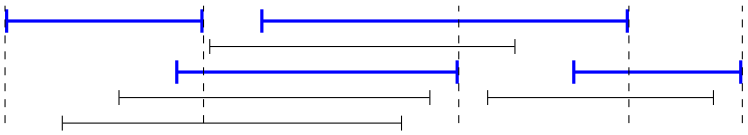


Figure 5 The thick blue jobs are the pivotal jobs, and the dashed vertical lines define the intervals. It is helpful to think of r_j for the first pivotal job j as the start point of the first interval.

3.1 Preprocessing & Preliminary Observations

We partition the time-horizon into intervals. We may assume that there is no timepoint throughout $[0, T]$ that is not covered by any job, since then we could easily seperate the instance at this timepoint into two independent subinstances. For our partitioning we inductively introduce a set of *pivotal jobs* P.

▶ **Definition 9.** The first pivotal job is the job with earliest start time r_j. We define the other pivotal jobs by induction. Assume that we have defined the first k pivotal jobs $j_1, ..., j_k$. Then the $(k+1)$-th pivotal job is the job with latest start time among all jobs j with $r_j \leq d_{j_k}$.

We use the end points of the pivotal jobs in order to partition the time horizon into intervals $\mathcal{I}$. More formally, we partition the time horizon into intervals at timepoints $X := \{d_j : j \in P\} \cup \{0\}$. Let $T_0, T_1, \cdots$ be the timepoints in X in increasing order. Then each interval in $\mathcal{I}$ is of the form $[T_k, T_{k+1}]$ for some $k \in \mathbb{N}$. See Figure 5 for an example.

▶ **Lemma 10.** *The $[r_j, d_j]$-interval of any non-pivotal job intersects at most one timepoint of X. The $[r_j, d_j]$-interval of any pivotal job intersects at most two timepoints of X.*

Next, we cut the instance into subinstances so that each subinstance contains at most q many intervals (in our final algorithm we will set $q = O(1/\epsilon)$). We do this in a randomized fashion but the procedure can be easily derandomized, similar to, e.g., [2]. Let x be a random variable that takes its value uniformly at random among the integers $\{0, 1, 2, \ldots, q-1\}$. We "cut" the instance into subinstances at timepoints $W := \{T_x, T_{x+q}, T_{x+2q}, \ldots\}$. Let each subinstance $I_i := [T_{x+iq}, T_{x+(i+1)q}]$ contain all jobs j whose interval $[r_j, d_j]$ intersects I_i. Jobs j whose intervals $[r_j, d_j]$ span two consecutive subinstances I_i and I_{i+1} are split into two jobs: a job j' with $r_{j'} := r_j, d_{j'} := T_{x+iq}, p_{j'} := p_j, c_{j'} := c_j$, and a job j'' with $r_{j''} := T_{x+iq}, d_{j''} := d_j, p_{j''} := p_j, c_{j''} := c_j$.

Note that the choice of x can be derandomized by trying out all q possible choices for x and selecting the best one. For the obtained division into subinstances we prove the following lemma.

▶ **Lemma 11.** *An exact algorithm with running time $O(f(n))$ for a subinstance containing at most q consecutive intervals from $\mathcal{I}$ yields a $(1 + 2/q)$-approximation algorithm for the original instance, with a running time of $O(n \cdot f(n))$.*

We give a pseudopolynomial-time exact algorithm for the problem on instances with $q = O(1/\epsilon)$ many consecutive intervals. The algorithm is based on dynamic programming. Due to space constraints in the main body of the paper we only present a simpler version of our DP for the case $q = 1$ in Subsection 3.2.

3.2 Solving a subinstance with only one interval

Assume that we are given a subinstance consisting of only one interval I_i. We form a partition $J_L \dot{\cup} J_R$ for the jobs whose $[r_j, d_j]$-interval intersects this interval I_i. The set J_L is the set of

all jobs j such that $d_j \in I_i$, and J_R is the set of such jobs j such that $r_j \in I_i$. By Lemma 10, $J_L \dot\cup J_R$ comprises the whole set of jobs j such that $[r_j, d_j] \cap I \neq \emptyset$. We now define a set of relevant timepoints M for our interval I_i as $M := (U \cap I_i)$, where $U := \{r_j, d_j | j \in J\}$ is the set of all globally relevant timepoints.

Let us consider this set ordered from left to right, so that $M = \{t_1, t_2 \dots t_k\}$. We fill out the table of our dynamic program in a bottom-up fashion by considering these timepoints in reverse order, that is from right to left. Each cell of the dynamic programming table has the form $T[t_z, b, i_L, c_L, c_R]$. Intuitively, it describes the subproblem of covering the demand on the subinterval $[t_z, t_k]$ by a set of jobs $J'_L \subseteq J_L$ having their respective deadline in $[t_z, t_k + 1]$ with $p(J'_L) := \sum_{j \in J'_L} p_j = i_L$ and $c(J'_L) := \sum_{j \in J'_L} c_j = c_L$, and by a set of jobs $J'_R \subseteq J_R$ having their respective release dates in $[t_z, t_k + 1]$ with $c(J'_R) = c_R$. The demand at each point $t \in [t_z, t_k]$ is $D_t - b$, i.e., the reader may imagine that some other routine of the global algorithm selects jobs with a total size of b that cover each point in $[t_z, t_k]$.

Formally, this DP cell is filled out with a "yes" if and only if there exist two sets $J'_L \subseteq J_L$ and $J'_R \subseteq J_R$, such that:

(i) for each job $j \in J'_R$, there holds $r_j \geq t_z$, and for each job $j \in J'_L$ there holds $d_j \geq t_z$,
(ii) $p(J'_L) = \sum_{j \in J'_L} p_j = i_L$ and $c(J'_L) = \sum_{j \in J'_L} c_j = c_L$,
(iii) $c(J'_R) = c_R$, and
(iv) $\forall \ell : z \leq \ell \leq k, \sum_{j \in J'_L \cup J'_R : [r_j, d_j] \ni t_\ell} p_j \geq D_{t_\ell} - b$.

Filling out the table. We fill out the table starting with all entries for the rightmost timepoint t_k. First, we fill in $T[t_k, b, i_L, c_L, c_R]$ for all possible values of $0 \leq i_L \leq \sum_j p_j$, $0 \leq c_L, c_R \leq \sum_j c_j$, and $0 \leq b \leq \sum_{j \in J_R} p_j$. Note that for such a cell only the pivotal job j_p of the interval is relevant since no other job can have its release date or deadline at t_k. For filling in the entry it suffices to consider the two possibilities of selecting j_p and not selecting j_p.

Assume now that we have filled in all cells corresponding to timepoints from t_{z+1} to t_k and we want to fill in the entries for t_z. The timepoint t_z is the start or the end point of a job j that either belongs to J_L or to J_R. The entries for t_z in our dynamic programming table depend on the set to which j belongs to, and on whether j is added to the solution. Formally, if $j \in J_R$, then $T[t_z, b, i_L, c_L, c_R]$ = "yes" if and only if $T[t_{z+1}, b, i_L, c_L, c_R]$ = "yes" and $i_L + b \geq D_{t_z}$ or if $T[t_{z+1}, b + p_j, i_L, c_L, c_R - c_j]$ = "yes" and $i_L + b + p_j \geq D_{t_z}$. So either we do not add j to the solution, and then we need to cover the demand at t_z with the jobs already selected for t_{z+1}, or we add j to the solution, and then we can add its size to the respective b-entry at t_{z+1}. Symmetrically, if $j \in J_L$, then $T[t_z, b, i_L, c_L, c_R]$ = "yes" if and only if we have that $T[t_{z+1}, b, i_L, c_L, c_R]$ = "yes" and $i_L + b \geq D_{t_z}$ or if $T[t_{z+1}, b, i_L - p_j, c_L - c_j, c_R]$ = "yes" and $i_L + b \geq D_{t_z}$.

By keeping track of the respective sets J_L and J_R in each cell we are able to reconstruct our solution starting from the cell of the form $T[t_1, 0, i_L, c_L, c_R]$ that minimizes $c_L + c_R$ among all such cells with a "yes"-entry. Our dynamic program requires pseudopolynomial running time, because the considered possible values for c_L, c_R, i_L and b are pseudopolynomial in the input size. It returns an exact solution to the given problem. We are able to generalize these ideas to subinstances with $O(1/\epsilon)$ many intervals, and thus prove the following theorem.

▶ **Theorem 12.** *There is a pseudopolynomial-time $(1 + \epsilon)$-approximation algorithm for the UFP-Cover problem on agreeable instances.*

References

1. Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A mazing 2+ϵ approximation for unsplittable flow on a path. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 26–41, 2014.
2. Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. Assoc. Comput. Mach.*, 41(1):153–180, January 1994. doi:10.1145/174644.174650.
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. doi:10.1145/1132516.1132617.
4. Nikhil Bansal and Kedar Dhamdhere. Minimizing weighted flow time. *ACM Transactions on Algorithms*, 3(4):Article 39, 2007. doi:10.1145/1290672.1290676.
5. Nikhil Bansal and Kirk Pruhs. The geometry of scheduling. *SIAM Journal on Computing*, 43(5):1684–1698, 2014. doi:10.1137/130911317.
6. Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 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. SIAM, 2015. doi:10.1137/1.9781611973730.5.
8. Venkatesan T. Chakaravarthy, Amit Kumar, Sambuddha Roy, and Yogish Sabharwal. Resource allocation for covering time varying demands. In *Algorithms–ESA 2011*, pages 543–554. Springer, 2011. doi:10.1007/978-3-642-23719-5_46.
9. Chandra Chekuri and Sanjeev Khanna. Approximation schemes for preemptive weighted flow time. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 297–305. ACM, 2002. doi:10.1145/509907.509954.
10. Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC'01)*, pages 84–93, 2001. doi:10.1145/380752.380778.
11. 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. To appear*, 2017.
12. Wiebke Höhn, Julián Mestre, and Andreas Wiese. How unsplittable-flow-covering helps scheduling with job-dependent cost functions. In *International Colloquium on Automata, Languages, and Programming*, pages 625–636. Springer, 2014. doi:10.1007/978-3-662-43948-7_52.
13. Eugene L. Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977. doi:10.1016/S0167-5060(08)70742-8.