


How Unsplittable-Flow-Covering Helps Scheduling with Job-Dependent Cost Functions

Wiebke Höhn¹ · Julián Mestre² ·
Andreas Wiese³ 

Received: 11 May 2016 / Accepted: 3 March 2017
© The Author(s) 2017. This article is an open access publication

Abstract Generalizing many well-known and natural scheduling problems, scheduling with job-specific cost functions has gained a lot of attention recently. In this setting, each job incurs a cost depending on its completion time, given by a private cost function, and one seeks to schedule the jobs to minimize the total sum of these costs. The framework captures many important scheduling objectives such as weighted flow time or weighted tardiness. Still, the general case as well as the mentioned special cases are far from being very well understood yet, even for only one machine. Aiming for better general understanding of this problem, in this paper we focus on the case of uniform job release dates on one machine for which the state of the art is a 4-approximation algorithm. This is true even for a special case that is equivalent to the covering version of the well-studied and prominent unsplittable flow on a path problem, which is interesting in its own right. For that covering problem, we present a quasi-polynomial time $(1 + \varepsilon)$ -approximation algorithm that yields an $(e + \varepsilon)$ -approximation for the above scheduling problem. Moreover, for the latter we devise the best possible resource

Funded by the Go8-DAAD joint research cooperation scheme. An extended abstract of this paper appeared in the proceedings of ICALP 2014.

✉ Andreas Wiese
awiese@mpi-inf.mpg.de
Wiebke Höhn
hoehn@math.tu-berlin.de
Julián Mestre
mestre@it.usyd.edu.au

- ¹ Technische Universität Berlin, Berlin, Germany
- ² The University of Sydney, Sydney, Australia
- ³ Max-Planck-Institut für Informatik, Saarbücken, Germany

augmentation result regarding speed: a polynomial time algorithm which computes a solution with *optimal* cost at $1 + \varepsilon$ speedup. Finally, we present an elegant QPTAS for the special case where the cost functions of the jobs fall into at most $\log n$ many classes. This algorithm allows the jobs even to have up to $\log n$ many distinct release dates. All proposed quasi-polynomial time algorithms require the input data to be quasi-polynomially bounded.

Keywords Approximation algorithms · Scheduling · Job-dependent cost functions · Unsplittable flow

Mathematics Subject Classification Approximation algorithms (68W25)

1 Introduction

In scheduling, a natural way to evaluate the quality of a solution is to assign a cost to each job which depends on its completion time. The goal is then to minimize the sum of these costs. The function describing this dependence may be completely different for each job.

There are many well-studied and important scheduling objectives which can be cast in this framework. Some of them are already very well understood, for instance weighted sum of completion times $\sum_j w_j C_j$ for which there are polynomial time approximation schemes (PTASs) [1], even for multiple machines and very general machine models. On the other hand, for natural and important objectives such as weighted flow time or weighted tardiness, not even a constant factor polynomial time approximation algorithm is known, even on a single machine. In a recent break-through result, Bansal and Pruhs presented a $O(\log \log P)$ -approximation algorithm [6] for the single machine case where every job has its private cost function, denoting by P the range of the processing times. Formally, they study the General Scheduling Problem (GSP) where the input consists of a set of jobs J where each job $j \in J$ is specified by a processing time p_j , a release date r_j , and a non-decreasing cost function f_j . The goal is to compute a preemptive schedule on one machine which minimizes $\sum_j f_j(C_j)$ where C_j denotes the completion time of job j in the schedule. Interestingly, even though this problem is very general, subsuming all the objectives listed above, the best known complexity result for it is only strong NP-hardness. Thus, there might even be a polynomial time $(1 + \varepsilon)$ -approximation.

Aiming to better understand GSP, in this paper we investigate the special case that all jobs are released at time 0. This version is still strongly NP-hard [19], even in the restricted case where the individual cost functions are scaled versions of an underlying common function [17]. The currently best known approximation algorithm for GSP without release dates is a $(4 + \varepsilon)$ -approximation algorithm [16]. As observed by Bansal and Verschae [7], this problem is a generalization of the covering-version of the well-studied Unsplittable Flow on a Path problem (UFP) [2, 3, 5, 9, 12, 15], which we refer to as *UFP cover problem*. The input of this problem consists of a path, each edge e having a demand u_e , and a set of tasks T . Each task i is specified by a start vertex s_i and an end vertex t_i on the path, defining a sub-path P_i , a size p_i , and a

cost c_i . In the UFP cover problem, the goal is to select a subset of the tasks $T' \subseteq T$ which covers the demand profile, i.e., $\sum_{i \in T' \cap T_e} p_i \geq u_e$ where T_e denotes the set of all tasks $i \in T$ such that $e \in P_i$. The objective is to minimize the total cost $\sum_{i \in T'} c_i$.

The UFP cover problem is a generalization of the knapsack cover problem [10] and corresponds to instances of GSP without release dates where the cost function of each job attains only the values 0, some job-dependent value c_i , and ∞ . The UFP cover problem has applications to resource allocation settings such as workforce and energy management, making it an interesting problem in its own right. For example, one can think of the tasks as representing time intervals when employees are available, and one aims at providing certain service level that changes over the day. The best known approximation algorithm for UFP cover is a 4-approximation [8, 11]. This essentially matches the best known result for GSP without release dates.

1.1 Our Contribution

In this paper we present several new approximation results for GSP without release dates and some of its special cases. Since these results are based on approximations for UFP cover problem, we state these auxiliary related results first.

First, we give a $(1 + \varepsilon)$ -approximation algorithm for the UFP cover problem with quasi-polynomial running time. Our algorithm uses some ideas from the QPTAS for UFP (packing) of Bansal et al. [3]. In UPF (packing), each edge has a capacity value analogous to the demand value in UFP cover; the goal is to select a maximum profit subset of tasks such that every edge the aggregate size of tasks using that edge does not exceed its capacity. The high-level idea behind the QPTAS of Bansal et al. [3] is to start with an edge in the middle of the path and to consider the tasks using it. One divides these tasks into groups, all tasks in a group having roughly the same size and cost. For each group, one guesses an approximation of the capacity profile used by an optimal solution using the tasks in that group. In UPF (packing), one can show that by slightly underestimating the true profile one still obtains almost the same profit as the optimum.

A natural adaptation of this idea to the UFP cover problem would be to guess an approximate coverage profile that *overestimates* the profile covered by an optimal solution. Unfortunately, it might happen that the tasks in a group may not suffice to cover certain approximate profiles. When considering only a polynomial number of approximate profiles, this could lead to a situation where the coverable approximate profiles are much more expensive to cover than the optimal solution.

We remedy this problem in a maybe counterintuitive fashion. Instead of guessing an approximate upper bound of the true profile, we first guess a *lower* bound of it. Then we select tasks that cover this lower bound, and finally add a small number of “maximally long” additional tasks. Using this procedure, we cannot guarantee how much our selected tasks exceed the guessed profile on each edge. However, we can guarantee that for the correctly guessed profile, we cover at least as much as the optimum and pay only slightly more. Together with the recursive framework from [3], we obtain a QPTAS. As an application, we use this algorithm to get a quasi-polynomial time $(e + \varepsilon)$ -approximation algorithm for GSP with uniform release dates,

improving the approximation ratio of the best known polynomial time 4-approximation algorithm [16]. This algorithm, as well as the QPTAS mentioned below, requires the input data to be quasi-polynomially bounded.

In addition, we consider a different way to relax the problem. Rather than sacrificing a $1 + \varepsilon$ factor in the objective value, we present a polynomial time algorithm that computes a solution with *optimal* cost but requires a speedup of $1 + \varepsilon$. Such a result can be easily obtained for job-independent, scalable cost functions i.e., functions f for which there exist a function ϕ satisfying $f(c \cdot t) = \phi(c) \cdot f(t)$ for any $c, t \geq 0$. In this case, the result is immediately implied by the PTAS in [21] and the observation that for scalable cost functions s -speed c -approximate algorithms translate into $(s \cdot c)$ -speed optimal ones. In our case, however, the cost functions of the jobs can be much more complicated and, even worse, they can be different for each job. Our algorithm first imposes some simplification on the solutions under consideration, at the cost of a $(1 + \varepsilon)$ -speedup. Then, we use a recently introduced technique to by Sviridenko and Wiese [23]. They first guess a set of discrete intervals representing slots for large jobs and the placement of big jobs into discrete intervals. Then they use a linear program to simultaneously assign large jobs into these slots and small jobs into the remaining idle times. Like in the latter paper, for the case that the processing times of the jobs are not polynomially bounded, we employ a technically involved dynamic program which moves on the time axis from left to right and considers groups of $O(\log n)$ intervals at a time.

An interesting open question is to design a (Q)PTAS for GSP without release dates. As a first step towards this goal, recently Megow and Verschae [21] presented a PTAS for minimizing the objective function $\sum_j w_j g(C_j)$ where each job j has a private weight w_j but the function g is identical for all jobs. In Sect. 4 we present a QPTAS for a generalization of this setting. Instead of only one function g for all jobs, we allow up to $(\log n)^{O(1)}$ such functions, each job using one of them, and we even allow the jobs to have up to $(\log n)^{O(1)}$ distinct release dates. We note that our algorithm requires the weights of the jobs to be in a quasi-polynomial range. Despite the fact that this setting is much more general, our algorithm is very clean and easy to analyze.

1.2 Related Work

As mentioned above, Bansal and Pruhs present a $O(\log \log P)$ -approximation algorithm for GSP [6]. Even for some well-studied special cases, this is now the best known polynomial time approximation result. For instance, for the important weighted flow time objective, previously the best known approximation factors were $O(\log^2 P)$, $O(\log W)$ and $O(\log nP)$ [4, 14], where P and W denote the ranges of the job processing times and weights, respectively. A QPTAS with running time $n^{O_\varepsilon(\log P \log W)}$ is also known [13]. For the objective of minimizing the weighted sum of completion times, PTASs are known, even for an arbitrary number of identical and a constant number of unrelated machines [1].

For the case of GSP with identical release dates, Bansal and Pruhs [6] give a 16-approximation algorithm. Later, Cheung et al. [16] gave a pseudo-polynomial primal-

dual 4-approximation, which can be adapted to run in polynomial time at the expense of increasing the approximation factor to $(4 + \varepsilon)$.

As mentioned above, a special case of GSP with uniform release dates is a generalization for the UFP cover problem. For this special case, a 4-approximation algorithm is known [8, 11]. The packing version is very well studied. After a series of papers on the problem and its special cases [5, 9, 12, 15], the currently best known approximation results are a QPTAS [3] and a $(2 + \varepsilon)$ -approximation in polynomial time [2].

2 Quasi-PTAS for UFP Cover

In this section, we present a quasi-polynomial time $(1 + \varepsilon)$ -approximation algorithm for the UFP cover problem. Subsequently, we show how it can be used to obtain an approximation algorithm with approximation ratio $e + \varepsilon \approx 2.718 + \varepsilon$ and quasi-polynomial running time for GSP with uniform release dates. Throughout this section, we assume that the sizes of the tasks are quasi-polynomially bounded. Our algorithm follows the structure from the QPTAS for the packing version of the UFP cover problem due to Bansal et al. [3]. First, we describe a recursive exact algorithm with exponential running time. Subsequently, we describe how to turn this routine into an algorithm with only quasi-polynomial running time and an approximation ratio of $1 + \varepsilon$.

To compute the exact solution (in exponential time) one can use the following recursive algorithm: Given the path $G = (V, E)$, denote by e_M the edge in the middle of G and let T_M denote the set of tasks that use e_M , i.e., the set of all tasks i such that $e_M \in P_i$. Our strategy is to “guess” which tasks in T_M are contained in OPT, an (unknown) optimal solution.

Throughout this paper, whenever we use the notion of *guessing* a set of tasks (or some other entity), we mean that we enumerate all possibilities for this set of tasks (or the entity) and continue the algorithm for each enumerated option. One of them will correspond to the respective choice in an optimal solution or in a suitably chosen near-optimal solution. In order to analyze the resulting algorithm, we can therefore assume that we know the corresponding choice in the mentioned (near-)optimal solution. This motivates the notion of *guessing*. Note that if we enumerate K possibilities for the set of tasks (or the entity) then this increases the running time of the remaining algorithm by a factor of K .

Once we have chosen the tasks from T_M that we want to include in our solution, the remaining problem splits into the two independent subproblems given by the edges on the left and on the right of e_M , respectively, and the tasks whose paths are fully contained in them. Therefore, we enumerate all subsets of $T'_M \subseteq T_M$. Denote by \mathcal{T}_M the resulting set of sets. For each set $T'_M \in \mathcal{T}_M$ we recursively compute the optimal solution for the subpaths $\{e_1, \dots, e_{M-1}\}$ and $\{e_{M+1}, \dots, e_{|E|}\}$, subject to the tasks in T'_M being already chosen and that no more tasks from T_M are allowed to be chosen. The leaf subproblems are given when the path in the recursive call has only one edge. Since $|E| = O(n)$ this procedure has a recursion depth of $O(\log n)$ which is helpful when aiming at quasi-polynomial running time. However, since in each recursive step we try each set $T'_M \in \mathcal{T}_M$, the running time is exponential (even in a single step of the recursion). To remedy this issue, we will show that there is a set of task sets

$\tilde{T}_M \subseteq T_M$ which is of small size and which approximates T_M well. More precisely, we can compute \tilde{T}_M in quasi-polynomial time (and it thus has only quasi-polynomial size) and there is a set $T_M^* \in \tilde{T}_M$ such that $c(T_M^*) \leq (1 + \varepsilon) \cdot c(T_M \cap \text{OPT})$ and T_M^* dominates $T_M \cap \text{OPT}$. In this context, for any set of tasks T , its cost is denoted by $c(T) := \sum_{i \in T} c_i$. We modify the above procedure such that we do recurse on each set in \tilde{T}_M , instead of recursing on each set in T_M . The set of task sets \tilde{T}_M has quasi-polynomial size and \tilde{T}_M contains the mentioned set T_M^* . When we continue in the same manner, the recursion depth becomes $O(\log n)$ and the resulting algorithm is a QPTAS. In the sequel, we describe the above algorithm in detail and show in particular how to obtain the set of task sets \tilde{T}_M .

2.1 Formal Description of the Algorithm

We assume that we know the value of the optimal objective, which we denote by B ; if we do not know the value of B , we can use binary search and the algorithm to estimate it within a $1 + \varepsilon$ factor. In a preprocessing step we reject all tasks i whose cost is larger than B and select all tasks i whose cost is at most $\varepsilon B/n$. The latter cost at most $n \cdot \varepsilon B/n \leq \varepsilon B$ and thus only a factor $1 + \varepsilon$ in the approximation ratio. We update the demand profile accordingly.

We define a recursive procedure $\text{UFPcover}(E', T')$, which gets as input a subpath $E' \subseteq E$ of G and a set of already chosen tasks T' . Denote by \tilde{T} the set of all tasks $i \in T \setminus T'$ such that the path of i uses only edges in E' . The output of $\text{UFPcover}(E', T')$ is a $(1 + \varepsilon)$ -approximation to the minimum cost solution for the subproblem of selecting a set of tasks $T'' \subseteq \tilde{T}$ such that $T' \cup T''$ satisfy all demands of the edges in E' , i.e., $\sum_{i \in (T' \cup T'') \cap T_e} p_i \geq u_e$ for each edge $e \in E'$. Note that there might be no feasible solution for this subproblem in which case we output ∞ . Let e_M be an edge in the “middle” of E' , such that the number of edges on the left and on the right of e_M differ by at most one. Denote by $T_M \subseteq \tilde{T}$ all tasks in \tilde{T} whose path uses e_M . As described in the previous subsection, the key is now to construct the set of task sets \tilde{T}_M such that (i) each task set in \tilde{T}_M is a subset of T_M , (ii) one of them (the set T_M^* in the previous discussion) dominates $T_M \cap \text{OPT}$ while $c(T_M^*) \leq (1 + \varepsilon) \cdot c(T_M \cap \text{OPT})$, and (iii) \tilde{T}_M contains only a quasi-polynomial number of task sets. Given \tilde{T}_M , we compute $\text{UFPcover}(E'_L, T' \cup T'_M)$ and $\text{UFPcover}(E'_R, T' \cup T'_M)$ for each set $T'_M \in \tilde{T}_M$, where E'_L and E'_R denote the subpaths of E' on the left and on the right of e_M , respectively. We output

$$\min_{T'_M \in \tilde{T}_M} c(T'_M) + \text{UFPcover}(E'_L, T' \cup T'_M) + \text{UFPcover}(E'_R, T' \cup T'_M).$$

For computing the set of task sets \tilde{T}_M , we first group the tasks in T_M into $(\log n)^{O(1)}$ many groups, all tasks in a group having roughly the same costs and sizes. Formally, for each pair (k, ℓ) , denoting (approximately) cost $(1 + \varepsilon)^k$ and size $(1 + \varepsilon)^\ell$, we define

$$T_{(k,\ell)} := \left\{ i \in T_M : (1 + \varepsilon)^k \leq c_i < (1 + \varepsilon)^{k+1} \wedge (1 + \varepsilon)^\ell \leq p_i < (1 + \varepsilon)^{\ell+1} \right\}.$$

Since the sizes of the tasks are quasi-polynomially bounded and we preprocessed the weights of the tasks, we have $(\log n)^{O(1)}$ non-empty groups. As we will show in the lemma below, for each group $T_{(k,\ell)}$, we can compute a polynomial-size set of task sets $\tilde{T}_{(k,\ell)}$ containing at least one set that approximates $\text{OPT}_{(k,\ell)} := \text{OPT} \cap T_{(k,\ell)}$ sufficiently well.

In order to prove the lemma, we formally introduce the notion of a *profile*. A profile $Q : E' \rightarrow \mathbb{R}_{\geq 0}$ assigns a *height* $Q(e)$ to each edge $e \in E'$, and a profile Q dominates a profile Q' if $Q(e) \geq Q'(e)$ holds for all $e \in E'$. The profile $Q_{\tilde{T}}$ induced by a set of tasks \tilde{T} is defined by the heights $Q_{\tilde{T}}(e) := \sum_{i \in T_e \in \tilde{T}} p_i$ (recall that T_e denotes all tasks in T whose path P_i contains the edge e). Finally, a set of tasks \tilde{T} dominates a set of tasks \tilde{T}' if $Q_{\tilde{T}}$ dominates $Q_{\tilde{T}'}$.

Lemma 1 *Given a group $T_{(k,\ell)}$, there is a polynomial time algorithm which computes a set of task sets $\tilde{T}_{(k,\ell)}$ that contains a set $T_{(k,\ell)}^* \in \tilde{T}_{(k,\ell)}$ such that*

$$c(T_{(k,\ell)}^*) \leq (1 + \varepsilon) \cdot c(\text{OPT}_{(k,\ell)})$$

and $T_{(k,\ell)}^*$ dominates $\text{OPT}_{(k,\ell)}$.

Proof First, we guess the number of tasks in $\text{OPT}_{(k,\ell)}$. If $|\text{OPT}_{(k,\ell)}|$ is smaller than $\frac{1}{\varepsilon^2}$ then we can guess an optimal set $\text{OPT}_{(k,\ell)}$. Otherwise, we will consider a polynomial number of approximate profiles, one of them underestimates the unknown true profile induced by $\text{OPT}_{(k,\ell)}$ by at most $O(\varepsilon) \cdot |\text{OPT}_{(k,\ell)}|$ units on each edge. For each approximate profile we will compute a cover of cost no more than $1 + O(\varepsilon)$ times the optimum cost of covering this profile, and if the profile is close to the true profile induced by $\text{OPT}_{(k,\ell)}$, we can extend this solution to a dominate the latter profile by adding only $O(\varepsilon) \cdot |\text{OPT}_{(k,\ell)}|$ more tasks.

Several arguments in the remaining proof are based on the structure of $T_{(k,\ell)}$ and the structure of the true profile $Q_{\text{OPT}_{(k,\ell)}}$. Since all tasks in $T_{(k,\ell)}$ contain the edge e_M and span a subpath of E' , the height of the profile $Q_{\text{OPT}_{(k,\ell)}}$ is unimodal: It is non-decreasing until e_M and non-increasing after that; see Fig. 1. In particular, a task that covers a certain edge e covers all edges in between e and e_M as well.

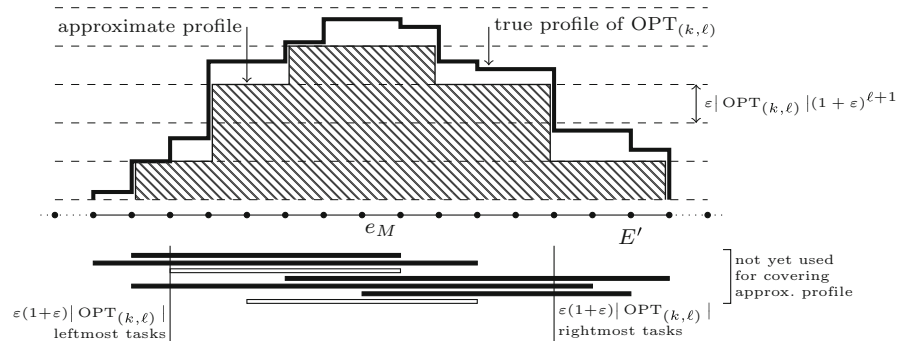


Fig. 1 Construction from Lemma 1

For the approximate profiles, we restrict ourselves to heights from

$$\mathcal{H} := \left\{ j \cdot \varepsilon \cdot |\text{OPT}_{(k,\ell)}| \cdot (1 + \varepsilon)^{\ell+1} \mid j \in \left\{ 0, 1, \dots, \frac{1}{\varepsilon} \right\} \right\}.$$

Moreover, aiming to approximate the true profile, we only take into account profiles in which the edges have non-decreasing and non-increasing height before and after e_M on the path, respectively. Utilizing the natural ordering of the edges on the path, we formally define the set \mathcal{Q} of approximate profiles as follows

$$\mathcal{Q} := \left\{ Q \mid \begin{array}{l} Q(e) \in \mathcal{H} \ \forall e \in E' \ \wedge \ Q(e) \leq Q(e') \ \forall e < e' \leq e_M \\ \wedge \ Q(e) \geq Q(e') \ \forall e_M \leq e < e' \end{array} \right\}.$$

Since $|\text{OPT}_{(k,\ell)}| \cdot (1 + \varepsilon)^{\ell+1}$ is an upper bound on the height of $Q_{\text{OPT}_{(k,\ell)}}$, there is a profile $Q^* \in \mathcal{Q}$ which is dominated by $Q_{\text{OPT}_{(k,\ell)}}$ and for which the gap $Q_{\text{OPT}_{(k,\ell)}}(e) - Q(e)$ does not exceed $\varepsilon \cdot |\text{OPT}_{(k,\ell)}| \cdot (1 + \varepsilon)^{\ell+1}$ for all $e \in E'$. Observe that by construction, an approximate profile can have at most $|\mathcal{H}|$ edges at which it jumps from one height to a larger one, and analogously, it can have at most $|\mathcal{H}|$ edges where it can jump down to some smaller height. Hence, \mathcal{Q} contains at most $n^{2|\mathcal{H}|} = n^{2/\varepsilon}$ profiles.

For each approximate profile $Q \in \mathcal{Q}$, we compute a cover based on LP rounding. To this end, we denote by $e_L(h)$ and $e_R(h)$ the first and last edge $e \in E'$ for which $Q(e) \geq h$, respectively. Note that by the structure of the paths of tasks in $T_{(k,\ell)}$, in fact every set of tasks covering $e_L(h)$ also covers all edges between e_M and $e_L(h)$ by at least the same amount, and analogously for $e_R(h)$. Regarding the LP-formulation, this allows us to only require a sufficient covering of the edges $e_L(h)$ and $e_R(h)$ rather than of all edges. Denoting by x_i the decision variable representing its selection for the cover, we formulate the LP as follows

$$\begin{array}{ll} \min & \sum_{i \in T_{(k,\ell)}} c_i \cdot x_i \\ & \sum_{i \in T_{(k,\ell):e_L(h) \in P_i}} x_i \cdot p_i \geq h & \forall h \in \mathcal{H} \\ & \sum_{i \in T_{(k,\ell):e_R(h) \in P_i}} x_i \cdot p_i \geq h & \forall h \in \mathcal{H} \\ & 0 \leq x_i \leq 1 & \forall i \in T_{(k,\ell)}. \end{array}$$

If there exists a feasible solution to the LP, we round up all fractional values x_i^* (i.e., values $x_i^* \in (0, 1)$) of some optimal extreme point solution x^* , and we choose the corresponding tasks as a cover for Q and denote them by T^* . Since the LP has only $2|\mathcal{H}| = \frac{2}{\varepsilon}$ more constraints than variables, its optimal extreme point solutions contain at most $\frac{2}{\varepsilon}$ fractional variables. Hence, the additional cost incurred by the rounding does not exceed $\frac{2}{\varepsilon}(1 + \varepsilon)^{\ell+1}$, where the latter term is the maximum task cost in $T_{(k,\ell)}$. For the matter of calculating the cost of the computed solution, let us assume that $Q = Q^*$ (recall that Q^* is a profile dominated by $Q_{\text{OPT}_{(k,\ell)}}$ for which the gap

$Q_{OPT_{(k,\ell)}}(e) - Q(e)$ is bounded by $\varepsilon \cdot |OPT_{(k,\ell)}| \cdot (1 + \varepsilon)^{\ell+1}$ on each edge $e \in E'$. Then, the cost of the selected tasks is at most

$$\begin{aligned} \sum_{i \in T_{(k,\ell)}} c_i \cdot x_i^* + \frac{2}{\varepsilon} (1 + \varepsilon)^{k+1} &\leq c(OPT_{(k,\ell)}) + 2\varepsilon \cdot |OPT_{(k,\ell)}| \cdot (1 + \varepsilon)^{k+1} \\ &\leq (1 + 2\varepsilon (1 + \varepsilon)) \cdot c(OPT_{(k,\ell)}), \end{aligned}$$

where the first and second inequality follows from $|OPT_{(k,\ell)}| \geq \frac{1}{\varepsilon^2}$ and from the minimum task weight in $T_{(k,\ell)}$, respectively, and moreover, the first inequality uses that $Q = Q^*$ is dominated by $Q_{OPT_{(k,\ell)}}$.

After covering some $Q \in \mathcal{Q}$ with tasks T^* in the first step, in the second step we extend this cover by additional edges $A^* \subseteq T_{(k,\ell)} \setminus T^*$. We define the set A^* to contain $\varepsilon (1 + \varepsilon) \cdot |OPT_{(k,\ell)}|$ tasks $T_{(k,\ell)} \setminus T^*$ with the leftmost start vertices and $\varepsilon (1 + \varepsilon) \cdot |OPT_{(k,\ell)}|$ tasks in $T_{(k,\ell)} \setminus T^*$ with the rightmost end vertices. We add $T^* \cup A^*$ to the set of task sets $\tilde{T}_{(k,\ell)}$.

Assume that $Q = Q^*$. Then the above LP has a feasible solution and in particular the resulting set $T^* \cup A^*$ was added to $\tilde{T}_{(k,\ell)}$. We claim that the computed tasks $T^* \cup A^*$ dominate $OPT_{(k,\ell)}$. Firstly, observe that any set of $\varepsilon (1 + \varepsilon) \cdot |OPT_{(k,\ell)}|$ tasks from $T_{(k,\ell)}$ has a total size of at least the gap between two height steps from \mathcal{H} . Hence, if an edge e is covered by that many edges from A^* and $Q = Q^*$ then we know that $Q_{T^* \cup A^*}(e) \geq Q_{OPT_{(k,\ell)}}(e)$.

On the other hand, if an edge e is covered by less than $\varepsilon (1 + \varepsilon) \cdot |OPT_{(k,\ell)}|$ tasks from A^* , we know that there exists no further task in $T_{(k,\ell)} \setminus (T^* \cup A^*)$ whose path contains e . Otherwise, this would be a contradiction to the choice of the tasks A^* being the $\varepsilon (1 + \varepsilon) \cdot |OPT_{(k,\ell)}|$ ones with the leftmost start and rightmost end vertices, respectively. Thus, since in this second case $T^* \cup A^*$ contains all tasks that cover e , we have that $Q_{T^* \cup A^*}(e) \geq Q_{OPT_{(k,\ell)}}(e)$.

Finally, the total cost of A^* does not exceed

$$2\varepsilon (1 + \varepsilon) \cdot |OPT_{(k,\ell)}| \cdot (1 + \varepsilon)^{k+1} \leq 2\varepsilon (1 + \varepsilon)^2 \cdot c(OPT_{(k,\ell)}),$$

and so $T^* \cup A^*$ has cost of at most $(1 + \varepsilon') \cdot c(OPT_{(k,\ell)})$ for $\varepsilon' := 2\varepsilon(1 + \varepsilon) (2 + \varepsilon)$. □

We define the set of tasks sets \tilde{T}_M as follows: we consider all combinations of taking exactly one set $\tilde{T}_{(k,\ell)} \in \tilde{T}_{(k,\ell)}$ from each set of task sets $\tilde{T}_{(k,\ell)}$ (there is one such set for each group $T_{(k,\ell)}$). For each such combination we take the union of the respective sets $\tilde{T}_{(k,\ell)}$ and add the resulting union to \tilde{T}_M . Since there are $(\log n)^{O(1)}$ groups, by Lemma 1 the set \tilde{T}_M contains only a quasi-polynomial number of task sets and it contains one set T_M^* which is a good approximation to $T_M \cap OPT$, i.e., the set T_M^* dominates $T_M \cap OPT$ and it is at most by a factor $1 + O(\varepsilon)$ more expensive. Now each node in the recursion tree has at most $n^{(\log n)^{O(1)}}$ children and, as argued above, the recursion depth is $O(\log n)$. Thus, a call to $UFPcover(E, \emptyset)$ has quasi-polynomial running time and yields a $(1 + O(\varepsilon))$ -approximation for the overall problem.

Theorem 1 For any $\varepsilon > 0$ there is a quasi-polynomial $(1 + \varepsilon)$ -approximation algorithm for UFP cover if the sizes of the tasks are in a quasi-polynomial range.

2.2 $(e + \varepsilon)$ -Approximation for GSP with Uniform Release Dates

Bansal and Pruhs [6] give a 4-approximation-preserving reduction from GSP with uniform release dates to UFP cover using geometric rounding. Here we observe that if instead we use *randomized geometric rounding* [18], then one can obtain an e -approximation-preserving reduction. Together with our QPTAS for UFP cover, we get the following result.

Theorem 2 For any $\varepsilon > 0$ there is a quasi-polynomial time $(e + \varepsilon)$ -approximation algorithm for GSP with uniform release dates.

Proof The heart of the proof is an e -approximation-preserving reduction from GSP with uniform release dates to UFP cover. Although here we develop a randomized algorithm, we note that the reduction can be de-randomized using standard techniques.

Given an instance of the scheduling problem we construct an instance of UFP cover as follows. For ease of presentation, we take our path $G = (V, E)$ to have vertices $0, 1, \dots, P$; towards the end, we explain how to obtain an equivalent and more succinct instance. For each $i = 1, \dots, P$, edge $e = (i - 1, i)$ has demand $u_e = P - i$. Finally, we assume that for each job j and time slot t we have $f_j(t) = 0$ or $f_j(t) \geq 1$; otherwise, we can always scale the functions so that this property holds.

The reduction has two parameters, $\gamma > 1$ and $\alpha \in [0, 1]$, which will be chosen later to minimize the approximation guarantee. For each job j , we define a sequence of times $t_0^j, t_1^j, t_2^j, \dots, t_k^j$ starting from 0 and ending with $P + 1$ such that the cost of finishing a job in between two consecutive times differs by at most a factor of γ . Formally, $t_0^j = 0, t_k^j = P + 1$ and t_i^j is the first time step such that $f(t_i^j) > \gamma^{i-1+\alpha}$. For each $i > 0$ such that $t_{i-1}^j < t_i^j$, we create a task covering the interval $[t_{i-1}^j, t_i^j - 1]$ having demand p_j and costing $f_j(t_i^j - 1)$.

Given a feasible solution of the UFP cover instance, we claim that we can construct a feasible schedule of no greater cost. For each job j , we consider the right-most task chosen (we need to pick at least one task from each job to be feasible) in the UFP cover solution and assign to j a due date equal to the right endpoint of the task. Notice that the cost of finishing the jobs by their due date equals the total cost of these right-most tasks. By the feasibility of the UFP cover solution, it must be the case that for each time t , the total processing volume of jobs with a due date of t or greater is at least $T - t + 1$. Therefore, scheduling the jobs according to earliest due date first, yields a schedule that meets all the due date. Therefore, the cost of the schedule is at most the cost of the UFP cover instance.

Conversely, given a feasible schedule, we claim that, if α is chosen uniformly at random and γ is set to e , then there is a solution of the UFP cover instance whose expected cost is at most e times more expensive than the cost of the schedule. For each job j , we pick all the tasks whose left endpoint is less than or equal to C_j , the completion time of j . It follows that the UFP cover solution is feasible. Let $f_j(C_j)$

be the cost incurred by j . For a fixed α , let the most expensive task induced by j cost $f_j(C_j)\gamma^\beta$. Notice that β is also uniformly distributed in $[0, 1]$. The combined expected cost of all the tasks induced by j is therefore

$$\int_0^1 f_j(C_j) (\gamma^\beta + \gamma^{\beta-1} + \dots) d\beta = f_j(C_j) \frac{\gamma}{\ln \gamma},$$

which is minimum at $\gamma = e$. By linearity of expectation, we get that the total cost of the UFP cover solution is at most an e factor larger than the cost of the schedule.

To de-randomize the reduction, and at the expense of adding another ε' to the approximation factor, one can discretize the random variable α , solve several instances, and return the one producing the best solution. Finally, we mention that it is not necessary to construct the full path from 0 to P . It is enough to keep the vertices where tasks start or end. Stretches where no task begins or end can be summarized by an edge having demand equal to the largest demand in that stretch.

Applying the ε -approximation-preserving reduction and then running the $(1 + \varepsilon)$ -approximation of Theorem 2 finishes the proof. \square

3 General Cost Functions Under Speedup

We present a polynomial time algorithm that computes a solution for an instance of GSP with uniform release dates whose cost is optimal and that is feasible if the machine runs with speed $1 + \varepsilon$ (rather than unit speed).

Let $0 < \varepsilon < 1$ be a constant and assume for simplicity that $\frac{1}{\varepsilon} \in \mathbb{N}$. For our algorithm, we first prove different properties that we can assume “at $1 + \varepsilon$ speedup”; by this, we mean that there is a schedule whose cost is at most the optimal cost (without enforcing these restricting properties) and that is feasible if we increase the speed of the machine by a factor $1 + \varepsilon$. Many statements are similar to properties that are used in [1] for constructing PTASs for the problem of minimizing the weighted sum of completion times.

For a given schedule denote by S_j and C_j the start and completion times of job j (recall that we consider only non-preemptive schedules). We define $C_j^{(1+\varepsilon)}$ to be the smallest power of $1 + \varepsilon$ which is not smaller than C_j , i.e.,

$$C_j^{(1+\varepsilon)} := (1 + \varepsilon)^{\lceil \log_{1+\varepsilon} C_j \rceil},$$

and we adjust the objective function as given in the next lemma. Also, we impose that jobs that are relatively large are not processed too early (the speedup will compensate for the delay of the start time). Formally, we enforce this by introducing artificial release dates $r(j)$ for each job j , and we disallow to start job j before time $r(j)$.

In a given schedule, we call a job j *large* if $S_j \leq \frac{1}{\varepsilon^3} \cdot p_j$ and *small* otherwise. For the large jobs, we do not allow arbitrary starting times but we discretize the time axis such that it contains only a constant number of starting times for large jobs (for constant ε). For this purpose, we define intervals and subintervals of the form

$$\begin{aligned}
 I_t &:= [R_t, R_{t+1}) && \text{with } R_t := (1 + \varepsilon)^t && \forall t \in \mathbb{N} \\
 I_{t,k} &:= [R_{t,k}, R_{t,k+1}) && \text{with } R_{t,k} := \left(1 + k \cdot \frac{1}{6} \cdot \frac{\varepsilon^4}{1+\varepsilon}\right) \cdot R_t && \forall t \in \mathbb{N}, \\
 &&&&& k \in \left\{0, \dots, 6 \lceil \frac{1+\varepsilon}{\varepsilon^3} \rceil\right\}.
 \end{aligned}$$

Note that for any fixed t all intervals $I_{t,k}$ have equal length. For the small jobs, we do not want them to overlap over interval boundaries and we want that all small jobs scheduled in an interval I_t are scheduled during one (connected) subinterval $I_t^s \subseteq I_t$.

Lemma 2 *At $1 + O(\varepsilon)$ speedup we can make the following assumptions:*

1. *The objective function is $\sum_j f_j(C_j^{(1+\varepsilon)})$, instead of $\sum_j f_j(C_j)$.*
2. *For each job j it holds $S_j \geq (1 + \varepsilon) \lceil \log_{1+\varepsilon} \left(\frac{\varepsilon}{1+\varepsilon} \cdot p_j\right) \rceil =: r(j)$.*
3. *Any small job starting during an interval I_t finishes in I_t .*
4. *Each large job starts at some point in time $R_{t,k}$ and every interval $I_{t,k}$ is used by either only small jobs or by one large job or it is empty.*
5. *For each interval I_t there is a time interval $I_{t,k,\ell} := [R_{t,k}, R_{t,\ell})$ with $0 \leq k \leq \ell \leq 6 \frac{1+\varepsilon}{\varepsilon^3}$ during which no large jobs are scheduled, and no small jobs are scheduled during $I_t \setminus I_{t,k,\ell}$.*

Proof Each property of the lemma will require us to increase the speed of the machine by a factor of $1 + \varepsilon$, apart from the last property. Compared to the initial unit speed, the final speed will be some power of $1 + \varepsilon$. Technically, we consolidate the resulting polynomial in ε to some $\varepsilon' = O(\varepsilon)$, achieving all properties at speed $1 + \varepsilon'$.

Throughout the proof we make use of the observation that at speedup $1 + \varepsilon$, a processing time p reduces to $\frac{1}{1+\varepsilon} \cdot p$, and hence, we gain idle time of length $\frac{\varepsilon}{1+\varepsilon} \cdot p$.

To observe the first point of the lemma, consider some job j with completion time C_j in an arbitrary schedule at unit speed. After increasing of the speed by a factor of $1 + \varepsilon$, time $C_j^{(1+\varepsilon)}$ corresponds to

$$(1 + \varepsilon)^{\lceil \log_{1+\varepsilon} \frac{C_j}{1+\varepsilon} \rceil} = (1 + \varepsilon)^{\lceil \log_{1+\varepsilon} C_j \rceil - 1} \leq C_j,$$

and hence, the ensued cost never exceeds the original cost.

Regarding the second point of the lemma, the above observation implies that at speedup $1 + \varepsilon$ a job j of processing time p_j allows for an additional idle time of length $\varepsilon / (1 + \varepsilon) \cdot p_j$. Hence, if $S_j < (1 + \varepsilon) \lceil \log_{1+\varepsilon} (\varepsilon \cdot p_j / (1+\varepsilon)) \rceil = r(j)$ we can set its start time to $r(j)$ without exceeding its unit speed completion time.

For showing the third point, consider a small job that starts in I_t and that finishes in some later interval. By definition, its length is at most $\varepsilon^3 \cdot R_{t+1}$. From the above observations and the interval length $|I_t| = \varepsilon \cdot R_t$, it follows that at speed $1 + \varepsilon$, the interval I_t provides an additional idle time of length $\varepsilon^2 / (1 + \varepsilon) \cdot R_t$, and the length of the small job reduces to at most $\varepsilon^3 \cdot R_t$. Since for sufficiently small ε it holds that $\frac{\varepsilon^2}{1+\varepsilon} \geq \varepsilon^3$, the small job can be scheduled during the new idle time, and hence, it finishes in I_t .

Regarding the second to last point of the lemma, we consider a large job i , the large job j that is scheduled after i , and all small jobs scheduled between i and j (any of the three may also not exist). Let I_t be the interval during which i finishes, and let $I_t(i, j) \subseteq I_t$ be the subinterval of I_t that is used by i and j and all small jobs in between. If i or j is fully scheduled during I_t then the length of $I_t(i, j)$ is at least the minimum processing time of a large job $\varepsilon^3 \cdot R_t$. Otherwise, if i starts before I_t and j finishes after I_t , it holds $I_t(i, j) = I_t$, and thus, $|I_t(i, j)| = \varepsilon \cdot R_t$. Hence, in any case we know that

$$|I_t(i, j)| \geq \min\{\varepsilon^3 \cdot R_t, \varepsilon \cdot R_t\} = \varepsilon^3 \cdot R_t.$$

By the above observation, at speedup $1 + \varepsilon$ the interval $I_t(i, j)$ provides additional idle time of $\varepsilon^4 / (1 + \varepsilon) \cdot R_t$, i.e., 6 free intervals $I_{t,k}$. We use two of these intervals to separate i from the small jobs and the small jobs from j , respectively. The remaining idle intervals $I_{t,k}$ ensure that we can proceed analogously for the overlapping intervals $I_t(*, i)$ and $I_t(j, *)$.

Note that by starting jobs from I_t earlier in the same interval we do not violate the release dates assumed by Lemma 2.2 since jobs are only released at the beginning of an interval I_t . This completes the proof of the fourth part of the lemma.

The proof of the last point of the lemma is a straight-forward implication of its fourth point. By this we can assume that all small jobs are contained in intervals $I_{t,k}$ that contain no large jobs. We change the order of the subintervals which either belong to small jobs or to large jobs that are fully scheduled in I_t . We proceed in such a way that all intervals of small jobs occur consecutively. Since all jobs still finish in I_t this modification does not increase the cost of the schedule.

Overall, we can make the assumptions of the lemma at a total speedup of $(1 + \varepsilon)^4$, which is $1 + O(\varepsilon)$ under our assumption that $\varepsilon < 1$, so the lemma follows. \square

3.1 Special Case of Polynomial Processing Times

For the moment, let us assume that the processing times of the instance are polynomially bounded. We will give a generalization to arbitrary instances later.

Our strategy is the following: Since the processing times are bounded, the whole schedule finishes within $\log_{1+\varepsilon}(\sum_j p_j) \leq O(\frac{1}{\varepsilon} \log n)$ intervals, (i.e., $O(\log n)$ for constant ε).¹ Ideally, we would like to guess the placement of all large jobs in the schedule and then use a linear program to fill in the remaining small jobs. However, this would result in $n^{O(\frac{1}{\varepsilon} \log n)}$ possibilities for the large jobs, which is quasi-polynomial but not polynomial. Instead, we only guess the *pattern* of large-job usage for each interval. A pattern P for an interval is a set of $O(\frac{1}{\varepsilon})$ integers which defines the start and end times of the *slots* during which large jobs are executed in I_t .

Proposition 1 *For each interval I_t there are only $N \in O_\varepsilon(1)$ many possible patterns, i.e., constantly many for constant ε . The value N is independent of t .*

¹ We write $O_\varepsilon(f(n))$ for an expression which is $O(f(n))$ for constant ε .

We first guess all patterns for all intervals at once. Since there are only $O\left(\frac{1}{\varepsilon} \log n\right)$ intervals, this yields only $N^{O\left(\frac{1}{\varepsilon} \log n\right)} \in n^{O_\varepsilon(1)}$ possible combinations for all patterns for all intervals. Suppose now that we guessed the pattern corresponding to the optimal solution correctly. Next, we solve a linear program that in parallel

- assigns large jobs to the slots specified by the pattern,
- assigns small jobs into the remaining idle times on the intervals.

Formally, we solve the following LP. We denote by Q the set of all slots for large jobs, $\text{size}(s)$ denotes the length of a slot s , $\text{begin}(s)$ its start time, and $t(s)$ denotes the index of the interval I_t that contains s . For each interval I_t denote by $\text{rem}(t)$ the remaining idle time for small jobs, and consider these idle times as slots for small jobs, which we refer to by their interval indices $I := \{1, \dots, \log_{1+\varepsilon}(\sum_j p_j)\}$. For each pair of slot $s \in Q$ and job $j \in J$, we introduce a variable $x_{s,j}$ corresponding to assigning j to s . Analogously, we use variables $y_{t,j}$ for the slots in I .

$$\min \sum_{j \in J} \left(\sum_{s \in Q} f_j(R_{t(s)+1}) \cdot x_{s,j} + \sum_{t \in I} f_j(R_{t+1}) \cdot y_{t,j} \right) \tag{1}$$

$$\sum_{s \in Q} x_{s,j} + \sum_{t \in I} y_{t,j} = 1 \quad \forall j \in J \tag{2}$$

$$\sum_{j \in J} x_{s,j} \leq 1 \quad \forall s \in Q \tag{3}$$

$$\sum_{j \in J} p_j \cdot y_{t,j} \leq \text{rem}(t) \quad \forall t \in I \tag{4}$$

$$x_{s,j} = 0 \quad \forall s \in Q, \forall j \in J : r(j) > \text{begin}(s) \vee p_j > \text{size}(s) \tag{5}$$

$$y_{t,j} = 0 \quad \forall t \in I, \forall j \in J : r(j) > R_t \vee p_j > \varepsilon \cdot |I_t|. \tag{6}$$

$$x_{s,j}, y_{t,j} \geq 0 \quad \forall s \in Q, \forall t \in I, \forall j \in J \tag{7}$$

Denote the above LP by sLP. It has polynomial size and thus we can solve it efficiently. Borrowing ideas from [22] we round it to a solution that is not more costly and which can be made feasible using additional speedup of $1 + \varepsilon$.

Lemma 3 *Given a fractional solution (x, y) to sLP. In polynomial time, we can compute a non-negative integral solution (x', y') whose cost is not larger than the cost of (x, y) and which fulfills the constraints (2), (3), (5), (6), (7) and*

$$\sum_{j \in J} p_j \cdot y_{t,j} \leq \text{rem}(t) + \varepsilon \cdot |I_t| \quad \forall t \in I. \tag{4a}$$

Proof The proof follows the general idea of [22]. Given some fractional solution (x, y) to the sLP (2)–(7), we construct a fractional matching M in a bipartite graph $G = (V \cup W, E)$. For each job $j \in J$ and for each large slot $s \in Q$, we introduce vertices $v_j \in V$ and $w_s \in W$, respectively. Moreover, for each slot of small jobs $t \in I$, we add $k_t := \lceil \sum_{j \in J} y_{t,j} \rceil$ vertices $w_{t,1}, \dots, w_{t,k_t} \in W$. We introduce an edge $(v_j, w_s) \in E$ with cost $f_j(R_{t(s)+1})$ for all job-slot pairs for which $x_{s,j} > 0$, and we choose it to an extent of $x_{s,j}$ for M . Regarding the vertices $w_{t,1}, \dots, w_{t,k_t}$, we add edges in the

following way. We first sort all jobs j with $y_{t,j} > 0$ in non-increasing order of their length p_j , and we assign them greedily to $w_{t,1}, \dots, w_{t,k_t}$; that is, we choose the first vertex $w_{t,\ell}$ which has not yet been assigned one unit of fractional jobs, we assign as much as possible of $y_{t,j}$ to it, and if necessary, we assign the remaining part to the next vertex $w_{t,\ell+1}$. Analogously to the above edges, we define the cost of an edge $(v_j, w_{t,\ell})$ to be $f_j(R_{t+1})$, and we add it fractionally to M according to the fraction $y_{t,\ell,j}$ of $y_{t,j}$ the job was assigned to $w_{t,\ell}$ by the greedy assignment. Note that $p_{t,\ell}^{\min} \geq p_{t,\ell+1}^{\max}$ for $\ell = 1, \dots, k_t - 1$ where $p_{t,\ell}^{\min}$ and $p_{t,\ell}^{\max}$ are the minimum and maximum length of all jobs (fractionally) assigned to $w_{t,\ell}$, respectively.

By construction, M is in fact a fractional matching, i.e., for every vertex $v_j \in V$ the set M contains edges whose chosen fractions add up to exactly 1. Moreover, the total cost of M equals the cost of the solution (x, y) . Due to standard matching theory, we know that there also exists an integral matching M' in G whose cost does not exceed the cost of M , and since G is bipartite, we can compute such a matching in polynomial time, see e.g., [20]. We translate M back into an integral solution (x', y') of the LP where we set $y_{t,j} = 1$ for every edge $(v_j, w_{t,\ell})$ in M . It remains to show that (x', y') satisfies (2), (3), (4a), (5), (6) and (7). All constraints but (4a) are immediately satisfied by construction. In order to show that (4a) is satisfied observe that

$$\begin{aligned} \sum_{j \in J} p_j \cdot y'_{t,j} &\leq \sum_{\ell=1}^{k_t} p_{t,\ell}^{\max} \leq p_{t,1}^{\max} + \sum_{\ell=2}^{k_t} p_{t,\ell}^{\max} \leq \varepsilon \cdot |I_t| + \sum_{\ell=1}^{k_t-1} p_{t,\ell}^{\min} \\ &\leq \varepsilon \cdot |I_t| + \sum_{\ell=1}^{k_t-1} \sum_{\substack{j \in J: \\ (v_j, w_{t,\ell}) \in E}} p_j \cdot y_{t,\ell,j} \leq \varepsilon \cdot |I_t| + \sum_{\ell=1}^{k_t} \sum_{\substack{j \in J: \\ (v_j, w_{t,\ell}) \in E}} p_j \cdot y_{t,\ell,j} \\ &= \varepsilon \cdot |I_t| + \sum_{j \in J} p_j \cdot y_{t,j} \leq \varepsilon \cdot |I_t| + \text{rem}(t), \end{aligned}$$

where the third inequality follows from (6). □

In particular, the cost of the computed solution is no more than the cost of the integral optimum and it is feasible under $1 + O(\varepsilon)$ speedup according to Lemma 2. We remark that the technique of guessing patterns and filling them in by a linear program was first used in [23].

3.2 General Processing Times

For the general case, i.e., for arbitrary processing times, we first show that at $1 + \varepsilon$ speedup, we can assume that for each job j there are only $O(\log n)$ intervals between $r(j)$ (the artificial release date of j) and C_j . Then we devise a dynamic program which moves from left to right on the time axis and considers sets of $O(\log n)$ intervals at a time, using the technique from Sect. 3.1.

Lemma 4 *At $1 + \varepsilon$ speedup we can assume that*

$$\frac{C_j}{r(j)} \leq q(n) := \frac{1}{\varepsilon^3} n + (1 + \varepsilon)^5.$$

Thus, $[r(j), C_j]$ has non-empty intersection with at most $K \leq O_\varepsilon(\log n)$ intervals I_t .

Proof Consider some interval I_t and some job j that has been released not later than the beginning of the interval, i.e., $r(j) \leq R_t$. According to the definition of $r(j)$ in Lemma 2.2 this is equivalent to $t \geq \left\lceil \log_{1+\varepsilon} \left(\frac{\varepsilon}{1+\varepsilon} \cdot p_j \right) \right\rceil$, which can be further bounded as follows

$$\left\lceil \log_{1+\varepsilon} \left(\frac{\varepsilon}{1+\varepsilon} \cdot p_j \right) \right\rceil \geq \log_{1+\varepsilon} \left(\frac{\varepsilon}{1+\varepsilon} \cdot p_j \right) - 1 = \log_{1+\varepsilon} (\varepsilon \cdot p_j) - 2.$$

This implies an upper bound on the total processing time of all jobs released before I_t

$$\sum_{j:r(j) \leq R_t} p_j \leq \sum_{j:r(j) \leq R_t} \frac{1}{\varepsilon} (1 + \varepsilon)^{t+2} \leq n \cdot \frac{1}{\varepsilon} (1 + \varepsilon)^{t+2} \leq \varepsilon^2 (1 + \varepsilon)^{t+2 + \left\lceil \log_{1+\varepsilon} \left(\frac{n}{\varepsilon^3} \right) \right\rceil}. \tag{8}$$

Now, consider the idle time that we gain in I_t at speedup $1 + \varepsilon$, which has a length of $\varepsilon / (1 + \varepsilon) \cdot |I_t| = \varepsilon^2 (1 + \varepsilon)^{t-1}$. From (8) it follows that at speedup $1 + \varepsilon$ the interval I_{t+s} with $s := \left\lceil \log_{1+\varepsilon} \left(\frac{n}{\varepsilon^3} \right) \right\rceil + 3$ provides enough space to hold all jobs released before R_t . Hence, we can assume those jobs to finish not later than R_{t+s+1} .

Since all jobs i with $r(i) \leq R_{t-1}$ can be scheduled in the idle time of some earlier interval if necessary, we can assume $R_{t-1} < r(j) \leq R_t$, and hence,

$$\frac{C_j}{r(j)} \leq \frac{R_{t+s+1}}{R_{t-1}} = (1 + \varepsilon)^{s+2} = \frac{1}{\varepsilon^3} \cdot n + (1 + \varepsilon)^5.$$

In particular, it is sufficient to consider $s + 2 = O_\varepsilon(\log n)$ intervals for processing a job. □

Throughout the remainder of this section let $K := \left\lceil \log_{1+\varepsilon}(q(n)) \right\rceil \in O_\varepsilon(\log n)$ where $q(n)$ is the polynomial from Lemma 4. Thus, K denotes the number of intervals between the time $r(j)$ and the completion time C_j of each job j .

If after the assumption of Lemma 4 there is a point in time τ that will *not* schedule any job, i.e., there is no job j with $\tau \in [r(j), r(j) \cdot q(n)]$, then we divide the instance into two independent pieces.

Proposition 2 *Without loss of generality we can assume that the union of all intervals $\bigcup_j [r(j), r(j) \cdot q(n)]$ is a (connected) interval.*

For our dynamic program (DP) we subdivide the time axis into *blocks*. Each block B_i consists of the intervals $I_{i,K}, \dots, I_{(i+1),K-1}$. The idea is that in each iteration the DP schedules the jobs released during a block B_i in the intervals corresponding to the

blocks B_i and B_{i+1} . So in the end, the intervals of each block B_{i+1} contain jobs released during B_i and B_{i+1} . To separate the jobs from both blocks we prove the following lemma.

Lemma 5 *At $1 + \varepsilon$ speedup we can assume that during each interval I_t in a block B_{i+1} there are two subintervals $[a_t, b_t), [b_t, c_t) \subseteq I_t$ such that*

- during $[a_t, b_t)$ only small jobs from block B_i are scheduled and during $I_t \setminus [a_t, b_t)$ no small jobs from block B_i are scheduled,
- during $[b_t, c_t)$ only small jobs from block B_{i+1} are scheduled and during $I_t \setminus [b_t, c_t)$ no small jobs from block B_{i+1} are scheduled,
- the interval boundaries a_t, b_t, c_t are of the form $(1 + z \cdot \frac{\varepsilon^4}{4(1+\varepsilon)^2}) \cdot R_t$ for $x \in \mathbb{N}$ and $z \in \{0, 1, \dots, \frac{4(1+\varepsilon)^2}{\varepsilon^3}\}$ (so possibly $[a_t, b_t) = \emptyset$ or $[b_t, c_t) = \emptyset$).

Proof Based on Lemma 2.3 we can assume that all small jobs that are started within I_t also finish in I_t ; moreover, they are processed in some interval $I_{t,k,\ell} \subseteq I_t$ which contains no large jobs (see Lemma 2.5 for the notation). By Lemma 4, the interval I_t can be assumed to contain only small jobs with release date in B_i and B_{i+1} , and by Lemma 2.1 we know that we can rearrange the jobs in I_t without changing the cost. Hence, for proving the lemma it is sufficient to show that we can split $I_{t,k,\ell}$ at some of the discrete points given in the lemma, such that the small jobs released in B_i and B_{i+1} are scheduled before and after this point, respectively.

The interval $I_{t,k,\ell}$ starts at $(1 + \frac{1}{4}k \cdot \varepsilon^4 / (1 + \varepsilon)) \cdot R_t$ and its length is some integral multiple of $\frac{1}{4} \varepsilon^4 / (1 + \varepsilon) \cdot R_t$. At a speedup of $1 + \varepsilon$, the interval $I_{t,k,\ell}$ provides additional idle time of length at least $\frac{1}{4} \varepsilon^4 / (1 + \varepsilon)^2 \cdot R_t$ (if $I_{t,k,\ell}$ is not empty), which equals the step width of the discrete interval end points required in the lemma. Hence, by scheduling all small jobs released in B_i and B_{i+1} at the very beginning and very end of $I_{t,k,\ell}$, there must be a point in time $\tau := (1 + z \cdot \varepsilon^4 / (4(1 + \varepsilon)^2)) \cdot R_t$ with $z \in \{0, 1, \dots, 4(1 + \varepsilon)^2 / \varepsilon^3\}$ which lies in the idle interval between the two groups of small jobs. Finally, if setting a_t and c_t to the start and end of $I_{t,k,\ell}$, respectively, and if choosing $b_t := \tau$, we obtain intervals as claimed in the lemma. \square

Using Lemma 4 we devise a dynamic program. We work again with patterns for the intervals. Here a pattern for an interval I_t in a block B_i denotes $O(\varepsilon)$ integers which define

- the start and end times of the large jobs from B_{i-1} which are executed during I_t ,
- the start and end times of the large jobs from B_i which are executed during I_t ,
- a_t, b_t, c_t according to Lemma 5, implying slots for small jobs.

Denote by \bar{N} the number of possible patterns for an interval I_t according to this definition. Similarly as in Proposition 1 we have that $\bar{N} \in O_\varepsilon(1)$ and \bar{N} is independent of t .

Each dynamic programming cell is characterized by a tuple (B_i, P_i) where B_i is a block during which at least one job is released or during the block thereafter, and P_i denotes a pattern for all intervals of block B_i . For a pattern P_i , we denote by $Q_i(P_i)$ and $Q_{i-1}(P_i)$ the set of slots in B_i which are reserved for large jobs released in B_{i-1}

and B_i , respectively. Moreover, for some interval I_t in B_i let $D_{i-1,t}(P_i)$ and $D_{i,t}(P_i)$ be the two slots for small jobs from B_{i-1} and B_i , respectively. The number of DP-cells is polynomially bounded as there are only n blocks during which at least one job is released and, as in Sect. 3.1, the number of patterns for a block is bounded by $\bar{N}^{O_\varepsilon(\log n)} \in n^{O_\varepsilon(1)}$.

The subproblem encoded in a cell (B_i, P_i) is to schedule all jobs j with $r(j) \geq I_{i,K}$ during $[R_{i,K}, \infty)$ while obeying the pattern P_i for the intervals $I_{i,K}, \dots, I_{(i+1) \cdot K-1}$. To solve this subproblem we first enumerate all possible patterns P_{i+1} for all intervals of block B_{i+1} . Suppose that we guessed the pattern P_{i+1} corresponding to the optimal solution of the subproblem given by the cell (B_i, P_i) . Like in Sect. 3.1 we solve the problem of scheduling the jobs of block B_i according to the patterns P_i and P_{i+1} by solving and rounding a linear program of the same type as sLP. Denote by $\text{opt}(B_i, P_i, P_{i+1})$ the optimal solution to this subproblem.

Lemma 6 *Given a DP-cell (B_i, P_i) and a pattern P_{i+1} , there is a polynomial time algorithm that computes a solution to the problem of scheduling all jobs released during B_i according to the patterns P_i, P_{i+1} such that*

- the cost is bounded by $\text{opt}(B_i, P_i, P_{i+1})$ and
- the schedule is feasible if during B_i and B_{i+1} the speed of the machine is increased by a factor of $1 + \varepsilon$.

Proof The proof works analogously to the proof of Lemma 3. We formulate the following LP for (fractionally) solving the problem

$$\min \sum_{j \in J_i} \left(\sum_{\substack{s \in Q_i(P_i) \\ \cup Q_i(P_{i+1})}} f_j(R_{t(s)+1}) \cdot x_{s,j} + \sum_{t=i \cdot K}^{(i+2) \cdot K-1} f_j(R_{t+1}) \cdot y_{t,j} \right) \tag{9}$$

$$\sum_{\substack{s \in Q_i(P_i) \\ \cup Q_i(P_{i+1})}} x_{s,j} + \sum_{t=i \cdot K}^{(i+2) \cdot K-1} y_{t,j} = 1 \quad \forall j \in J_i \tag{10}$$

$$\sum_{j \in J_i} x_{s,j} \leq 1 \quad \forall s \in Q_i(P_i) \cup Q_i(P_{i+1}) \tag{11}$$

$$\sum_{j \in J_i} p_j \cdot y_{t,j} \leq |D_{i,t}(P_{i(t)})| \quad \forall t \in \{i \cdot K, \dots, (i+2) \cdot K - 1\} \tag{12}$$

$$x_{s,j} = 0 \quad \forall j \in J_i, \forall s \in Q : r(j) > \text{begin}(s) \vee p_j > \text{size}(s) \tag{13}$$

$$y_{t,j} = 0 \quad \forall t \in I, \forall j \in J_i : r(j) > R_t \vee p_j > \varepsilon \cdot |I_t| \tag{14}$$

$$x_{s,j}, y_{t,j} \geq 0 \quad \forall j \in J_i, \forall s \in Q_i(P_i) \cup Q_i(P_{i+1}), \forall t \in \{i \cdot K, \dots, (i+2) \cdot K - 1\}, \tag{15}$$

where $J_i \subseteq J$ denotes the set of all jobs j with $r(j) \in B_i$, and $i(t)$ is the index of the block that contains the interval I_t .

This LP has exactly the same structure as sLP (1)–(7) and hence, we obtain an analogous result to Lemma 3. This means that given a fractional solution (x, y) to the above LP, we can construct an integral solution (x', y') which is not more costly than (x, y) , and which fulfills all constraints (10)–(15) with (12) being replaced by the relaxed constraint

$$\sum_{j \in J_i} p_j \cdot y_{t,j} \leq |D_{i,t}(P_{i(t)})| + \varepsilon \cdot |I_t| \quad \forall t \in \{i \cdot K, \dots, (i + 2) \cdot K - 1\}.$$

We increase the speed by a factor of $1 + \frac{\varepsilon}{1-\varepsilon} \in 1 + O(\varepsilon)$ and thus each interval provides an additional idle time of $\varepsilon \cdot |I_t|$. Therefore, in each interval I_t we can schedule all jobs that are assigned to I_t by the integral solution (x', y') . Due to Lemma 2.1, this does not increase the cost of the schedule which concludes the proof. \square

By definition of the patterns, an optimal solution $\text{OPT}(B_{i+1}, P_{i+1})$ is independent of the patterns that have been chosen for earlier blocks. This is simply due to the separately reserved slots for jobs from different blocks within each pattern, i.e., a slot in B_{i+1} which is reserved for jobs from B_i cannot be used by jobs from B_{i+1} in any case. Hence, $\text{OPT}(B_i, P_i)$ decomposes into $\text{OPT}(B_{i+1}, P_{i+1})$ and $\text{opt}(B_i, P_i, P_{i+1})$ for a pattern $P_{i+1} \in \mathcal{P}_{i+1}$ which leads to the lowest cost, where \mathcal{P}_{i+1} denotes the set of all possible patterns for block B_{i+1} . Thus, formally it holds that

$$\text{OPT}(B_i, P_i) = \min_{P_{i+1} \in \mathcal{P}_{i+1}} \text{OPT}(B_{i+1}, P_{i+1}) + \text{opt}(B_i, P_i, P_{i+1}). \quad (16)$$

This observation of an optimal substructure allows to easily formulate a DP. We interpret each cell (B_i, P_i) as a node in a graph, and we add an edge between cells (B_i, P_i) and (B_{i+1}, P_{i+1}) for all $P_i \in \mathcal{P}_i$ and $P_{i+1} \in \mathcal{P}_{i+1}$. For each triple B_i, P_i, P_{i+1} we compute a solution using Lemma 6, and we assign the cost of this solution to the edge $((B_i, P_i), (B_{i+1}, P_{i+1}))$. Due to (16), a minimum cost path in this $O(\text{poly}(n))$ size graph corresponds to a scheduling solution whose cost, at speed $1 + \varepsilon$, does not exceed the optimal cost at unit speed.

Overall, in our argumentation above we needed to increase the speed of the machine by a factor $1 + O(\varepsilon) \leq 1 + \alpha \cdot \varepsilon$ for some constant α and we obtained a polynomial time algorithm, assuming that ε is constant. Therefore, for any given constant $\varepsilon' > 0$ we can define $\varepsilon := \varepsilon'/\alpha$ and construct our algorithm above for this value of ε . This yields a polynomial time algorithm that needs to increase the speed of the machine only by a factor $1 + \varepsilon'$. The main theorem of this section follows.

Theorem 3 *Let $\varepsilon > 0$. There is a polynomial time algorithm for GSP with uniform release dates which computes a solution with optimal cost and which is feasible if the machine runs with speed $1 + \varepsilon$.*

4 Few Classes of Cost Functions

In this section, we study the following special case of GSP with release dates. We assume that each cost function f_j can be expressed as $f_j = w_j \cdot g_{u(j)}$ for a job-

dependent weight $w_j \in \mathbb{N}$, k global functions g_1, \dots, g_k , and an assignment $u : J \rightarrow [k]$ of cost functions to jobs. We present a QPTAS for this problem, assuming that $k = (\log n)^{O(1)}$ and that the jobs have at most $(\log n)^{O(1)}$ distinct release dates. We assume that the job weights are in a quasi-polynomial range, i.e., we assume that there is an upper bound $W = 2^{(\log n)^{O(1)}}$ for the job weights.

In our algorithm, we first round the values of the functions g_i so that they attain only few values, $(\log n)^{O(1)}$ many. Then we guess the $(\log n)^{O(1)}/\varepsilon$ most expensive jobs and their costs. For the remaining problem, we use a linear program. Since we rounded the functions g_i , our LP is sparse, and by rounding an extreme point solution we increase the cost by at most an ε -fraction of the cost of the previously guessed jobs, which yields an $(1 + \varepsilon)$ -approximation overall.

Formally, we use a binary search framework to estimate the optimal value B . Having this estimate, we adjust the functions g_i such that each of them is a step function with at most $(\log n)^{O(1)}$ steps, all being powers of $1 + \varepsilon$ or 0.

Lemma 7 *At $1 + \varepsilon$ loss we can assume that for each $i \in [k]$ and each t it holds that $g_i(t)$ is either 0 or a power of $1 + \varepsilon$ in $[\frac{\varepsilon}{n} \cdot \frac{B}{W}, B)$.*

Proof Denote by $g_i^{(1+\varepsilon)}$ the rounded cost functions for $i \in [k]$, i.e., formally we define

$$g_i^{(1+\varepsilon)}(t) := \begin{cases} \min \left\{ (1 + \varepsilon)^{\lceil \log_{1+\varepsilon}(g_i(t)) \rceil}, B \right\}, & \text{if } g_i(t) > \frac{\varepsilon}{n} \cdot \frac{B}{W} \\ \frac{\varepsilon}{n} \cdot \frac{B}{W}, & \text{if } 0 < g_i(t) \leq \frac{\varepsilon}{n} \cdot \frac{B}{W} \\ 0, & \text{if } g_i(t) = 0. \end{cases}$$

Consider some optimal schedule with completion time C_j for $j \in J$. Then it holds that

$$\begin{aligned} \sum_{j \in J} w_j \cdot g_{u(j)}^{(1+\varepsilon)}(C_j) &\leq \sum_{\substack{j \in J: 0 < g_{u(j)}(C_j) \\ \leq (\varepsilon \cdot B)/(n \cdot W)}} w_j \cdot \frac{\varepsilon \cdot B}{n \cdot W} + (1 + \varepsilon) \cdot \sum_{\substack{j \in J: g_{u(j)}(C_j) \\ > (\varepsilon \cdot B)/(n \cdot W)}} w_j \cdot g_{u(j)}(C_j) \\ &\leq \varepsilon \cdot B \cdot \frac{1}{n} \sum_{\substack{j \in J: 0 < g_{u(j)}(C_j) \\ \leq (\varepsilon \cdot B)/(n \cdot W)}} \frac{w_j}{W} + (1 + \varepsilon) \cdot B \\ &\leq \varepsilon \cdot B + (1 + \varepsilon) \cdot B = (1 + 2\varepsilon) \cdot B. \end{aligned}$$

The lemma follows by redefining ε . □

Our problem is in fact equivalent to assigning a due date d_j to each job such that the due dates are *feasible*, meaning that there is a preemptive schedule where every job finishes no later than its due date, and the objective being $\sum_j f_j(d_j)$ (see also [6]). The following lemma characterizes when a set of due dates is feasible.

Lemma 8 ([6]) *Given a set of jobs and a set of due dates. The due dates are feasible if and only if for every interval $I = [r_j, d_{j'}]$ for any two jobs j, j' , the jobs in*

$X(I) := \{\bar{j} : r_{\bar{j}} \in I\}$ that are assigned a deadline after I have a total size of at least $\text{ex}(I) := \max(\sum_{\bar{j} \in X(I)} p_{\bar{j}} - |I|, 0)$. That is,

$$\sum_{\bar{j} \in X(I): d_{\bar{j}} > d_{I'}} p_{\bar{j}} \geq \text{ex}(I) \quad \forall I = [r_j, d_{j'}].$$

Denote by D all points in time where at least one cost function $g_i^{(1+\varepsilon)}$ increases. It suffices to consider only those values as possible due dates.

Proposition 3 *There is an optimal due date assignment such that $d_j \in D$ for each job j .*

Denote by R the set of all release dates of the jobs. Recall that $|R| \leq (\log n)^{O(1)}$. Now, we guess the $|D| \cdot |R| / \varepsilon$ most expensive jobs of the optimal solution and their respective costs. Due to the rounding in Lemma 7 we have that $|D| \leq k \cdot \log_{1+\varepsilon}(W \cdot n / \varepsilon) = (\log n)^{O(1)}$ and thus there are only $O(n^{|D| \cdot |R| / \varepsilon}) = n^{(\log n)^{O(1)} / \varepsilon}$ many guesses.

Suppose we guess this information correctly. Let J_E denote the guessed jobs and for each job $j \in J_E$ denote by d_j the latest time where it attains the guessed cost, i.e., its *due date*. Denote by c_{thres} the minimum cost of a job in J_E , according to the guessed costs. The remaining problem consists of assigning a due date $d_j \in D$ to each job $J \setminus J_E$ such that none of these jobs costs more than c_{thres} , all due dates together are feasible, and the overall cost is minimized. We express this as a linear program.

In the LP, we have a variable $x_{j,t}$ for each pair of a job $j \in J \setminus J_E$ and a due date $t \in D$ such that j does not cost more than c_{thres} when finishing at time t . We add the constraint $\sum_{t \in D} x_{j,t} = 1$ for each job j , modeling that the job has a due date, and one constraint for each interval $[r, t]$ with $r \in R$ and $t \in D$ to model the condition given by Lemma 8.

$$\min \sum_{j \in J \setminus J_E} \sum_{t \in D} x_{j,t} \cdot f_j(t) \tag{17}$$

$$\sum_{\substack{j \in (J \setminus J_E) \\ \cap X([r,t])}} \sum_{\substack{t' \in D: \\ t' > t}} p_j \cdot x_{j,t'} + \sum_{\substack{j \in J_E \cap X([r,t]): \\ d_j > t}} p_j \geq \text{ex}([r, t]) \quad \forall r \in R \quad \forall t \in D \tag{18}$$

$$\sum_{t \in D} x_{j,t} = 1 \quad \forall j \in J \setminus J_E \tag{19}$$

$$x_{j,t} = 0 \quad \forall j \in J \setminus J_E \quad \forall t \in D : \\ r_j + p_j > t \vee w_j g_{u(j)}(t) > c_{\text{thres}} \tag{20}$$

$$x_{j,t} \geq 0 \quad \forall j \in J \setminus J_E \quad \forall t \in D \tag{21}$$

In polynomial time, we compute an extreme point solution x^* for the LP. The next lemma shows that by appropriate rounding, we can turn x^* into an integral solution at a small increase of cost.

Lemma 9 Denote by $c(x^*)$ the cost of an extreme point solution x^* . In polynomial time we can compute a set of feasible due dates $\{d_j\}_{j \in J}$ such that

$$\sum_{j \in J \setminus J_E} f_j(d_j) \leq c(x^*) + \varepsilon \cdot \sum_{j \in J_E} f_j(d_j).$$

Proof The solution x^* has at most $|D| \cdot |R| + |J \setminus J_E|$ many non-zeros. Each job j needs at least one non-zero variable $x_{j,t}^*$, due to the constraint $\sum_{t \in D} x_{j,t} = 1$. Thus, there are at most $|D| \cdot |R|$ fractionally assigned jobs, i.e., jobs j having a variable $x_{j,t}^*$ with $0 < x_{j,t}^* < 1$. We define an integral solution by rounding x^* as follows: For each job j we set d_j to be the maximum value t such that $x_{j,t}^* > 0$. Since the solution x^* has at most $|D| \cdot |R|$ fractional entries, the rounding affects at most $|D| \cdot |R|$ variables whose corresponding cost $x_{j,t} \cdot f_j(t)$ do not exceed c_{thres} after the rounding. Thus, in the resulting schedule we have

$$\begin{aligned} \sum_{j \in J \setminus J_E} f_j(d_j) &\leq c(x^*) + |D| \cdot |R| \cdot c_{\text{thres}} \leq c(x^*) + \varepsilon \cdot \frac{1}{\varepsilon} \cdot |D| \cdot |R| \cdot c_{\text{thres}} \\ &\leq c(x^*) + \varepsilon \cdot \sum_{j \in J_E} f_j(d_j). \end{aligned}$$

This completes the proof. □

Since $c(x^*) + \sum_{J_E} f_j(d_j)$ is a lower bound on the optimum, we obtain a $(1 + \varepsilon)$ -approximation. As there are quasi-polynomially many guesses for the expensive jobs and the remainder can be done in polynomial time, we obtain a QPTAS.

Theorem 4 There is a QPTAS for GSP, assuming that each cost function f_j can be expressed as $f_j = w_j \cdot g_{u(j)}$ for some job-dependent weight w_j and at most $k = (\log n)^{O(1)}$ global functions g_1, \dots, g_k , and that the jobs have at most $(\log n)^{O(1)}$ distinct release dates.

Acknowledgements Open access funding provided by Max Planck Society. We would like to thank the anonymous reviewers for many helpful comments.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Afrati, F., Bampis, E., Chekuri, C., Karger, D., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., Sviridenko, M.: Approximation schemes for minimizing average weighted completion time with release dates. In: Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS '99), pp. 32–44 (1999)
2. Anagnostopoulos, A., Grandoni, F., Leonardi, S., Wiese, A.: A mazing $2+\varepsilon$ approximation for unsplittable flow on a path. In: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '14), pp. 26–41 (2014)

3. Bansal, N., Chakrabarti, A., Epstein, A., Schieber, B.: A quasi-PTAS for unsplittable flow on line graphs. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06), pp. 721–729 (2006)
4. Bansal, N., Dhamdhere, K.: Minimizing weighted flow time. *ACM Trans. Algorithms* **3**(4), 39 (2007)
5. Bansal, N., Friggstad, Z., Khandekar, R., Salavatipour, R.: A logarithmic approximation for unsplittable flow on line graphs. In: Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '09), pp. 702–709 (2009)
6. Bansal, N., Pruhs, K.: The geometry of scheduling. *SIAM J. Comput.* **43**(5), 1684–1698 (2014)
7. Bansal, N., Verschae, J.: Personal communication
8. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. *J. ACM* **48**(5), 1069–1090 (2001)
9. Bonsma, P., Schulz, J., Wiese, A.: A constant factor approximation algorithm for unsplittable flow on paths. In: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '11), pp. 47–56 (2011)
10. Carr, R.D., Fleischer, L.K., Leung, V.J., Phillips, C.A.: Strengthening integrality gaps for capacitated network design and covering problems. In: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00), pp. 106–115 (2000)
11. Chakaravarthy, V.T., Kumar, A., Roy, S., Sabharwal, Y.: Resource allocation for covering time varying demands. In: Proceedings of the 19th European Symposium on Algorithms (ESA '11), volume 6942 of LNCS, pp. 543–554. Springer (2011)
12. Chakrabarti, A., Chekuri, C., Gupta, A., Kumar, A.: Approximation algorithms for the unsplittable flow problem. In: Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX '02), volume 2462 of LNCS, pp. 51–66. Springer (2002)
13. Chekuri, C., Khanna, S.: Approximation schemes for preemptive weighted flow time. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02), pp. 297–305 (2002)
14. Chekuri, C., Khanna, S., Zhu, A.: Algorithms for minimizing weighted flow time. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC '01), pp. 84–93 (2001)
15. Chekuri, C., Mydlarz, M., Shepherd, F.: Multicommodity demand flow in a tree and packing integer programs. *ACM Trans. Algorithms* **3**(3), 27 (2007)
16. Cheung, M., Mestre, J., Shmoys, D.B., Verschae, J.: A primal-dual approximation algorithm for min-sum single-machine scheduling problems. [arXiv:1612.03339](https://arxiv.org/abs/1612.03339)
17. Höhn, W., Jacobs, T.: On the performance of Smith's rule in single-machine scheduling with nonlinear cost. *ACM Trans. Algorithms* **11**(4), 25 (2015)
18. Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. *Inf. Comput.* **131**(1), 63–79 (1996)
19. Lawler, E.L.: A “pseudopolynomial” algorithm for sequencing jobs to minimize total tardiness. In: Studies in Integer Programming, volume 1 of Annals of Discrete Mathematics, pp. 331–342. North-Holland, Amsterdam (1977)
20. Lovász, L., Plummer, M.: Matching Theory, volume 29 of Annals of Discrete Mathematics. North-Holland, Amsterdam (1986)
21. Megow, N., Verschae, J.: Dual techniques for scheduling on a machine with varying speed. In: Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP '13), volume 7965 of LNCS, pp. 745–756. Springer (2013)
22. Shmoys, D.B., Tardos, É.: An approximation algorithm for the generalized assignment problem. *Math. Program.* **62**(1–3), 461–474 (1993)
23. Sviridenko, M., Wiese, A.: Approximating the configuration-LP for minimizing weighted sum of completion times on unrelated machines. In: Proceedings of the 16th Conference on Integer Programming and Combinatorial Optimization (IPCO '13), volume 7801 of LNCS, pp. 387–398. Springer (2013)