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

Wiebke Höhn[1], Julián Mestre[2], and Andreas Wiese[3]

[1] Technische Universität Berlin, Germany
hoehn@math.tu-berlin.de
[2] The University of Sydney, Australia
mestre@it.usyd.edu.au
[3] Max-Planck-Institut für Informatik, Saarbücken, Germany
awiese@mpi-inf.mpg.de

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 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.

1 Introduction

In scheduling, a natural way to evaluate the quality of a computed 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

* Funded by the Go8-DAAD joint research cooperation scheme.
** A full version of the paper can be found at http://arxiv.org/abs/1403.1376.

J. Esparza et al. (Eds.): ICALP 2014, Part I, LNCS 8572, pp. 625–636, 2014.

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 [6,7] 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, and 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 computed 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, so 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 case is still strongly NP-hard [20] and the currently best know approximation algorithm for it is a $(4+\varepsilon)$-approximation algorithm [18,22][1]. As observed by Bansal and Verschae [8], this problem is a generalization of the covering-version of the well-studied Unsplittable Flow on a Path problem (UFP) [2,3,5,11,14,17]. 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, an end vertex t_i, a size p_i, and a cost c_i. In the covering version, 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 all tasks in T whose path uses e. The objective is to minimize the total cost $\sum_{i \in T'} c_i$.

This covering version of UFP 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. UFP-cover is a generalization of the knapsack cover problem [12] 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 best known approximation algorithm for UFP-cover is a 4-approximation [9,13], which essentially matches the best known result for GSP without release dates.

Our Contribution. In this paper we present several new approximation results for GSP without release dates and some of its special cases. First, we give a

[1] In [18] a primal-dual $(2+\varepsilon)$-approximation algorithm was claimed for this problem. However, there is a error in the argumentation: there are instances [22] where the algorithm constructs a dual solution whose value differs from the optimal integral solution by a factor of 4.

$(1+\varepsilon)$-approximation algorithm for the covering version of UFP with quasi-polynomial running time. Our algorithm follows the high-level idea of the known QPTAS for the packing version [3]. Its key concept is to start with an edge in the middle 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, and guesses for each group an approximation of the capacity profile used by the tasks from that group. In the packing version, one can show that by slightly underestimating the true profile one still obtains almost the same profit as the optimum. For the covering version, a natural adjustment would be to use an approximate profile which *over*estimates the true profile. However, when using only a polynomial number of approximate profiles, it can happen that in the instance there are simply not enough tasks from a group available so that one can cover the overestimated profile which approximates the actual profile in the best possible way.

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 (instance-independently) 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 [18,22]. This algorithm, as well as the QPTAS mentioned below, requires the input data to be quasi-polynomially bounded.

Moreover, 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 requiring a speedup of $1+\varepsilon$. Such a result can be easily obtained for job-*independent*, scalable cost functions using the PTAS in [21] (a cost function f is scalable if $f(c\,t) = \phi(c)\,f(t)$ for some suitable function ϕ and all all $c, t \geq 0$). 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 first guess a set of discrete intervals representing slots for large jobs and then use a linear program to simultaneously assign large jobs into these slots and small jobs into the remaining idle times [24].

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 Section 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 algorithms 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.

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 $O(\log nP)$ [4,16], 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 [15]. 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, Shmoys and Cheung claimed a primal-dual $(2+\varepsilon)$-approximation algorithm [18]. However, an instance was later found where the algorithm constructs a dual solution which differs from the best integral solution by a factor 4 [22], suggesting that the primal-dual analysis can show only an approximation ratio of 4. On the other hand, Mestre and Verschae [22] showed that the local-ratio interpretation of that algorithm (recall the close relation between the primal-dual schema and the local-ratio technique [10]) is in fact a pseudopolynomial time 4-approximation, yielding a $(4+\varepsilon)$-approximation in polynomial time.

As mentioned above, a special case of GSP with uniform release dates is a generalization for the covering version of Unsplittable Flow on a Path. For this special case, a 4-approximation algorithm is known [9,13]. The packing version is very well studied. After a series of papers on the problem and its special cases [5,11,14,17], 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 without 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 Unsplittable Flow on a Path 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 $1+\varepsilon$.

For computing 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 tasks that use e_M. Our strategy is to "guess" which tasks in T_M are contained in OPT, the (unknown) optimal solution. Note that once these tasks are chosen, 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 $\{e_1, ..., e_{M-1}\}$ and $\{e_{M+1}, ..., e_{|E|}\}$, 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 one single step of the recursion). To remedy this issue, we will show that for any set $\mathcal{T}_M$ appearing in the recursive procedure there is a set $\bar{\mathcal{T}}_M$ which is of small size and which approximates $\mathcal{T}_M$ well. More precisely, we can compute $\bar{\mathcal{T}}_M$ in quasi-polynomial time (and it thus has only quasi-polynomial size) and there is a set $T^*_M \in \bar{\mathcal{T}}_M$ such that $c(T^*_M) \leq (1+\varepsilon) \cdot c(T_M \cap \mathrm{OPT})$ and T^*_M dominates $T_M \cap \mathrm{OPT}$. For any set of tasks T' we write $c(T') := \sum_{i \in T'} c_i$, and for two sets of tasks T_1, T_2, we say that T_1 *dominates* T_2 if $\sum_{i \in T_1 \cap T_e} p_i \geq \sum_{i \in T_2 \cap T_e} p_i$ for each edge e. We modify the above procedure such that we do recurse on sets in $\bar{\mathcal{T}}_M$ instead of $\mathcal{T}_M$. Since $\bar{\mathcal{T}}_M$ has quasi-polynomial size, $\bar{\mathcal{T}}_M$ contains the mentioned set T^*_M, and the recursion depth is $O(\log n)$, 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 $\bar{\mathcal{T}}_M$.

2.1 Formal Description of the Algorithm

We use a binary search procedure to guess the optimal objective value B. First, 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 UFPcover(E', T') which gets as input a subpath $E' \subseteq E$ of G and a set of already chosen tasks T'. Denote by $\bar{T}$ the set of all tasks $i \in T \setminus T'$ such that the path of i uses only edges in E'. The output of UFPcover(E', T') is a $(1+\varepsilon)$-approximation to the minimum cost solution for the subproblem of selecting a set of tasks $T'' \subseteq \bar{T}$ 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 the edge in the middle of E', i.e., at most $|E'|/2$ edges are on the left and on the right of e_M, respectively. Denote by $T_M \subseteq \bar{T}$ all tasks in $\bar{T}$ whose path uses e_M. As described above, the key is now to construct the set $\bar{\mathcal{T}}_M$ with the above properties. Given this set, we compute UFPcover$(E'_L, T' \cup T'_M)$

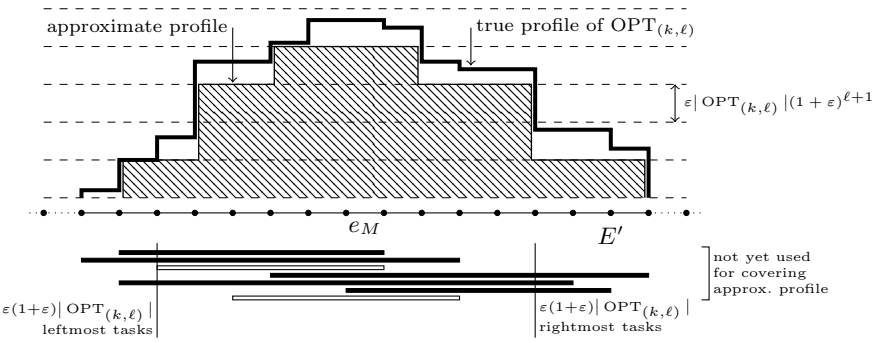

Fig. 1. Construction from Lemma 1

and $\mathrm{UFPcover}(E'_R, T' \cup T'_M)$ for each set $T'_M \in \bar{\mathcal{T}}_M$, where E'_L and E'_R denote the subpaths of E' on the left and on the right of e_M, respectivley. We output

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

For computing the set $\bar{\mathcal{T}}_M$, 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

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

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.

For each group $T_{(k,\ell)}$, we compute a set $\bar{\mathcal{T}}_{(k,\ell)}$ containing at least one set which is not much more expensive than $\mathrm{OPT}_{(k,\ell)} := \mathrm{OPT} \cap T_{(k,\ell)}$ and which dominates $\mathrm{OPT}_{(k,\ell)}$. To this end, observe that the sizes of the tasks in $\mathrm{OPT}_{(k,\ell)}$ cover a certain profile (see Figure 1). Initially, we guess the number of tasks in $\mathrm{OPT}_{(k,\ell)}$, and if $|\,\mathrm{OPT}_{(k,\ell)}\,| \le \frac{1}{\varepsilon^2}$ then we simply enumerate all subsets of $T_{(k,\ell)}$ with at most $\frac{1}{\varepsilon^2}$ tasks. Otherwise, we consider a polynomial number of profiles that are potential approximations of the true profile covered by $\mathrm{OPT}_{(k,\ell)}$. To this end, we subdivide the (implicitly) guessed height of the true profile evenly into $\frac{1}{\varepsilon}$ steps of uniform height, and we allow the approximate profiles to use only those heights while being monotonously increasing and decreasing before and after e_M, respectively (observe that also $\mathrm{OPT}_{(k,\ell)}$ has this property since all its tasks use e_M). This leads to at most $n^{O(1/\varepsilon)}$ different approximate profiles in total.

For each approximate profile we compute a set of tasks covering it using LP-rounding. The path of any task in $T_{(k,\ell)}$ contains the edge e_M, and hence, a task covering an edge e always covers all edges inbetween e and e_M as well. Thus, when formulating the problem as an LP, it suffices to introduce one constraint for the leftmost and one constraint for the rightmost edge of each height in

the approximated profile. We compute an extreme point solution of the LP and round up each of the at most $\frac{2}{\varepsilon}$ fractional variables. Since $|\operatorname{OPT}_{(k,\ell)}| \geq \frac{1}{\varepsilon^2}$ this increases the cost at most a factor $1 + O(\varepsilon)$ compared to the cost of the LP.

It is clear that the LP has a solution if the approximate profile is dominated by the true profile. Among such approximate profiles, consider the one that is closest to the latter. On each edge it would be sufficient to add $O(\varepsilon \cdot \left|\operatorname{OPT}_{(k,\ell)}\right|)$ tasks from $T_{(k,\ell)}$ in order to close the remaining gap. This is due to our choice of the step size of the approximate profile and the fact that all tasks in $T_{(k,\ell)}$ have roughly the same size. To this end, from the not yet selected tasks in $T_{(k,\ell)}$ we add the $O(\varepsilon \cdot |\operatorname{OPT}_{(k,\ell)}|)$ tasks with the leftmost start vertex and the $O(\varepsilon \cdot |\operatorname{OPT}_{(k,\ell)}|)$ tasks with the rightmost end vertex (see Figure 1). This costs again at most an $O(\varepsilon)$-fraction of the cost so far. As a result, on each edge e we have either selected $O(\varepsilon \cdot \left|\operatorname{OPT}_{(k,\ell)}\right|)$ additional tasks using it, thus closing the remaining gap, or we have selected *all* tasks from $T_{(k,\ell)}$ using e. In either case, the selected tasks dominate the tasks in $\operatorname{OPT}_{(k,\ell)}$, i.e., the true profile.

Lemma 1. *Given a group $T_{(k,\ell)}$. There is a polynomial time algorithm which computes a set of task sets $\bar{\mathcal{T}}_{(k,\ell)}$ which contains a set $T^*_{(k,\ell)} \in \bar{\mathcal{T}}_{(k,\ell)}$ such that $c(T^*_{(k,\ell)}) \leq (1+\varepsilon) \cdot c(\operatorname{OPT}_{(k,\ell)})$ and $T^*_{(k,\ell)}$ dominates $\operatorname{OPT}_{(k,\ell)}$.*

We define the set $\bar{\mathcal{T}}_M$ by taking all combinations of selecting exactly one set from the set $\bar{\mathcal{T}}_{(k,\ell)}$ of each group $T_{(k,\ell)}$. Since there are $(\log n)^{O(1)}$ groups, by Lemma 1 the set $\bar{\mathcal{T}}_M$ has only quasi-polynomial size and it contains one set T^*_M which is a a good approximation to $T_M \cap \operatorname{OPT}$, i.e., the set T^*_M dominates $T_M \cap \operatorname{OPT}$ 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.*

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* [19], 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.*

3 General Cost Functions under Speedup

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

Let $1 > \varepsilon > 0$ be a constant and assume for simplicity that $\frac{1}{\varepsilon} \in \mathbb{N}$. For our algorithm, we first prove some 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 which 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 end times of job j in a given schedule (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 adjust the objective function as given in the next lemma. Also, we impose that jobs that are relatively large are not processed too early; formally, they do not run before $(1+\varepsilon)^{\lfloor \log_{1+\varepsilon} \varepsilon \cdot p_j/(1+\varepsilon) \rfloor}$ which is the largest power of $1+\varepsilon$ which is at most $\varepsilon/(1+\varepsilon) \cdot p_j$ (the speedup will compensate for the delay of the start time).

Lemma 2. *At $1+O(\varepsilon)$ speedup we can use the objective function $\sum_j f_j\big(C_j^{(1+\varepsilon)}\big)$, instead of $\sum_j f_j(C_j)$, and assume $S_j \geq (1+\varepsilon)^{\lfloor \log_{1+\varepsilon} \varepsilon \cdot p_j/(1+\varepsilon) \rfloor}$ for each job j.*

Next, we discretize the time axis into intervals of the form $I_t := [R_t, R_{t+1})$ where $R_t := (1+\varepsilon)^t$ for any integer t. Note that $|I_t| = \varepsilon \cdot R_t$. Following Lemma 2, to simplify the problem we want to assign an artificial release date to each job j. For each job j, we define $r(j) := (1+\varepsilon)^{\lfloor \log_{1+\varepsilon} \varepsilon \cdot p_j/(1+\varepsilon) \rfloor}$. Lemma 2 implies then that we can assume $S_j \geq r(j)$ for each job j. Therefore, we interpret the value $r(j)$ as the release date of job j and from now on 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 each interval contains only a constant number of starting times for large jobs (for constant ε). 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 3. *At $1+O(\varepsilon)$ speedup we can assume that*

- *each interval I_t contains only $O(\frac{1}{\varepsilon^3})$ potential start points for large jobs, and*
- *for each interval I_t there is a time interval $I_t^s \subseteq I_t$, ranging from one potential start point for large jobs to another, which fully contains all small jobs scheduled in I_t and no large jobs.*

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. 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 $O(\frac{1}{\varepsilon^3})$ integers which defines the start and end times of the *slots* during which large jobs are executed in I_t. Note that such a job might start before I_t and/or end after 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.*

We first guess all patterns for all intervals at once. Since there are only $O(\frac{1}{\varepsilon}\log n)$ intervals, this yields only $N^{O(\frac{1}{\varepsilon}\log n)} \in n^{O_\varepsilon(1)}$ 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, and also, it 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, $\mathsf{size}(s)$ denotes the length of a slot s, $\mathsf{begin}(s)$ its start time, and $t(s)$ denotes the index of the interval I_t that contains s. For each interval I_t denote by $\mathsf{rem}(t)$ 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.

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

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

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

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

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

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

$$x_{s,j},\, y_{t,j} \ge 0 \qquad \forall\, s \in Q, \forall\, t \in I, \forall j \in J. \tag{7}$$

Denote the above LP by sLP. It has polynomial size and thus we can solve it efficiently. Borrowing ideas from [23] we round it to a solution that is not more costly and which can be made feasible using additional speedup of $1+\varepsilon$.

Lemma 4. *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*

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

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 (accumulating all the speedups from the previous lemmas). We remark that the technique of guessing patterns and filling them in by a linear program was first used in [24].

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 above technique.

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, k global functions $g_1, ..., g_k$, and an assignment $u : J \to [k]$ 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 (integral) 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 5. *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 $\left[\frac{\varepsilon}{n} \cdot \frac{B}{W}, B\right)$.*

Our problem is in fact equivalent to assigning a due date d_j to each job (cf. [6]) 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 $\sum_j f_j(d_j)$. The following lemma characterizes when a set of due dates is feasible.

Lemma 6 ([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 $X(I) := \{j : r_j \in I\}$ that are assigned a deadline after I have a total size of at least $\mathrm{ex}(I) := \max(\sum_{j \in X(I)} p_j - |I|, 0)$. That is, $\sum_{\bar{j} \in X(I): d_{\bar{j}} > d_{j'}} p_{\bar{j}}$ is at least $\mathrm{ex}(I)$ for all intervals $I = [r_j, d_{j'}]$.*

Denote by D all points in time where at least one cost function g_i increases. It suffices to consider only those values as possible due dates.

Proposition 2. *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 $|R| \leq (\log n)^{O(1)}$. We guess now the $|D|\cdot|R|/\varepsilon$ most expensive jobs of the optimal solution and their respective costs. Due to the rounding in Lemma 5 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 in 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 that 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 6.

In polynomial time, we compute an extreme point solution x^* for the LP. It 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 $x^*_{j,t} > 0$. We round up at most $|D| \cdot |R|$ jobs and after the rounding, each of them costs at most c_{thres}. Hence, those jobs cost at most an ε-fraction of the cost of guessed jobs (J_E).

Lemma 7. *Denote by $c(x^*)$ the cost of the solution x^*. We have that*
$$\textstyle\sum_{j\in J\setminus J_E} f_j(d_j) \;\leq\; c(x^*) + \varepsilon \cdot \sum_{j\in J_E} f_j(d_j).$$

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 $g_1, ..., g_k$, and that the jobs have at most $(\log n)^{O(1)}$ distinct release dates.*

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 FOCS 1999, pp. 32–44 (1999)
2. Anagnostopoulos, A., Grandoni, F., Leonardi, S., Wiese, A.: A mazing 2+ε approximation for unsplittable flow on a path. In: Proceedings of SODA 2014, 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 STOC 2006, pp. 721–729 (2006)

4. Bansal, N., Dhamdhere, K.: Minimizing weighted flow time. ACM T. Alg. 3(4), article 39 (2007)
5. Bansal, N., Friggstad, Z., Khandekar, R., Salavatipour, R.: A logarithmic approximation for unsplittable flow on line graphs. In: Proceedings of SODA 2009, pp. 702–709 (2009)
6. Bansal, N., Pruhs, K.: The geometry of scheduling. In: Proceedings of FOCS 2010, pp. 407–414 (2010),
See also `http://www.win.tue.nl/~nikhil/pubs/wflow-journ3.pdf`
7. Bansal, N., Pruhs, K.: Weighted geometric set multi-cover via quasi-uniform sampling. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 145–156. Springer, Heidelberg (2012)
8. Bansal, N., Verschae, J.: Personal communication
9. 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)
10. Bar-Yehuda, R., Rawitz, D.: On the equivalence between the primal-dual schema and the local ratio technique. SIAM J. Discrete Math. 19(3), 762–797 (2005)
11. Bonsma, P., Schulz, J., Wiese, A.: A constant factor approximation algorithm for unsplittable flow on paths. In: Proceedings of FOCS 2011, pp. 47–56 (2011)
12. 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 SODA 2000, pp. 106–115 (2000)
13. Chakaravarthy, V.T., Kumar, A., Roy, S., Sabharwal, Y.: Resource allocation for covering time varying demands. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 543–554. Springer, Heidelberg (2011)
14. Chakrabarti, A., Chekuri, C., Gupta, A., Kumar, A.: Approximation algorithms for the unsplittable flow problem. In: Jansen, K., Leonardi, S., Vazirani, V.V. (eds.) APPROX 2002. LNCS, vol. 2462, pp. 51–66. Springer, Heidelberg (2002)
15. Chekuri, C., Khanna, S.: Approximation schemes for preemptive weighted flow time. In: Proceedings of STOC 2002, pp. 297–305 (2002)
16. Chekuri, C., Khanna, S., Zhu, A.: Algorithms for minimizing weighted flow time. In: Proceedings of STOC 2001, pp. 84–93 (2001)
17. Chekuri, C., Mydlarz, M., Shepherd, F.: Multicommodity demand flow in a tree and packing integer programs. ACM T. Alg. 3(3), article 27 (2007)
18. Cheung, M., Shmoys, D.B.: A primal-dual approximation algorithm for min-sum single-machine scheduling problems. In: Goldberg, L.A., Jansen, K., Ravi, R., Rolim, J.D.P. (eds.) RANDOM 2011 and APPROX 2011. LNCS, vol. 6845, pp. 135–146. Springer, Heidelberg (2011)
19. Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. Inform. Comput. 131(1), 63–79 (1996)
20. Lawler, E.L.: A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. Ann. Discrete Math. 1, 331–342 (1977)
21. Megow, N., Verschae, J.: Dual techniques for scheduling on a machine with varying speed. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 745–756. Springer, Heidelberg (2013)
22. Mestre, J., Verschae, J.: A 4-approximation for scheduling on a single machine with general cost function, `http://arxiv.org/abs/1403.0298`
23. Shmoys, D.B., Tardos, É.: An approximation algorithm for the generalized assignment problem. Math. Program. 62(1-3), 461–474 (1993)
24. Sviridenko, M., Wiese, A.: Approximating the configuration-LP for minimizing weighted sum of completion times on unrelated machines. In: Goemans, M., Correa, J. (eds.) IPCO 2013. LNCS, vol. 7801, pp. 387–398. Springer, Heidelberg (2013)