Scheduling with an Orthogonal Resource Constraint

Martin Niemeier[1] and Andreas Wiese[2]

[1] TU Berlin, Sekretariat MA 5-1, Straße des 17. Juni 136, 10623 Berlin, Germany
martin.niemeier@tu-berlin.de
[2] Università di Roma "La Sapienza", Via Ariosto 25, 00185 Roma, Italy
wiese@dis.uniroma1.it

Abstract. We address a scheduling problem that arises in highly parallelized environments like modern multi-core CPU/GPU computer architectures. Here simultaneously active jobs share a common limited resource, e.g., memory cache. The scheduler must ensure that the demand for the common resource never exceeds the available capacity. This introduces an *orthogonal constraint* to the classical *minimum makespan scheduling* problem. Such a constraint also arises in many other contexts where a common resource is shared across the machines.
We study the non-preemptive case of this problem and give a $(2+\epsilon)$-approximation algorithm which relies on the interplay of several classical and modern techniques in scheduling like grouping, job-classification, and the use of configuration-LPs. This improves upon previous bound of 3 that can be obtained by list scheduling approaches, and gets close to the $(3/2-\epsilon)$ inapproximability bound. If the number of machines or the number of different resource requirements are bounded by a constant we have a polynomial time approximation scheme.

1 Introduction

Highly parallelized processing in modern multi-core CPU/GPU computer architectures poses the following scheduling challenge: Simultaneously active jobs need to share a common limited resource, e.g. a memory cache. The scheduler must ensure that the demand for the common resource never exceeds the available capacities. This introduces an additional "orthogonal" constraint to scheduling problems.

In this paper we address the imposition of such an orthogonal constraint to the classical and well studied *minimum makespan problem*. We are given a set of jobs where each job j has a processing time $p(j)$ and resource requirement $r(j)$. The goal is to schedule the jobs to m identical machines minimizing the makespan, i.e. the largest finishing time of a job. The scheduler must ensure that the *resource constraint* is not violated: At no time t, the total resource consumption (i.e. the sum of resource requirements of jobs active at time t) exceeds 1. (Note that fixing the resource capacity to 1 is without loss of generality.) This problem is called the *resource constrained scheduling problem* (see e.g. [15]).

Note that this model also captures many other settings where the jobs on the machines share a common resource like power supply, workers, etc.

This problem can be seen as a hybrid of two classical problems from two different domains. On the one hand, one obtains minimum makespan scheduling on identical machines if all resource requirements are zero. On the other hand, the well-known Bin-Packing problem is a special case of the resource constrained scheduling problem: Given a Bin-Packing instance, create a job of processing time 1 for each item and set its resource requirement to the item size. The hybrid nature of the problem is also reflected in our algorithms. To tackle it, we combine ideas and techniques from both domains.

1.1 Related Work

Without the resource constraint, one obtains the classic and well studied problem of scheduling on identical machines. Here Graham shows that a natural greedy list scheduling algorithm, where jobs are scheduled in the order of non-increasing processing times, yields an approximation ratio of $\frac{4}{3} - \frac{3}{m}$ [9,10]. After a series of improvements [4,6,18,20] Hochbaum and Shmoys present a a polynomial time approximation scheme (PTAS) [14].

The problem with orthogonal resource constraints, i.e. the resource constrained scheduling problem, was first studied by Garey and Graham [7]. They consider s independent orthogonal resources and show that every list scheduler is a $(s+2-\frac{2s+1}{m})$-approximation algorithm. In the setting of unrelated machines and one resource a 3.75-approximation algorithm follows from a more general result for scheduling with resource dependent processing times [11]. For the latter problem on identical machines also a $(3.5+\epsilon)$-approximation algorithm is known [16]. If preemption is allowed then there is a PTAS if the number of resource requirements is bounded by a constant [15]. If additionally the number of machines is constant, the problem can be solved in polynomial time [2]. There is extensive work in classifying further variants of the problem into polynomial time solvable and NP-hard problems. We refer to [8] and [2] for good overviews. Also, the problem can be seen as a special case of the resource-constrained project scheduling problem (RCPSP), see [1,12] for overviews on this problem.

As already mentioned above, the problem of scheduling jobs to minimize the makespan is closely related to the intensively studied Bin-Packing problem, see [13] for a good survey. For this problem, an asymptotic PTAS is known [5] while it is NP-hard to approximate with a better ratio than 3/2 [13].

Several related, yet clearly different problems are studied in the literature, including multi-dimensional packing problems [3] and geometrical packing problems like strip-packing [17]. For a survey on results for two-dimensional packing problems see [19].

1.2 Our Contribution

We study the problem to minimize the makespan for non-preemptive scheduling with an orthogonal resource constraint. Our main result is a $(2+\epsilon)$-approximation

algorithm which requires a novel and complex combination of modern and classical techniques from scheduling and Bin-Packing, including

- Configuration-LPs and geometric properties of extreme point solutions
- Enumeration by exploiting structure and search-space reduction
- Linear grouping techniques in the spirit of de la Vega and Lueker [5]
- Classification of jobs (large, small, fat, thin jobs).

In particular, our techniques for constructing the schedule go far beyond the list scheduling type methods used in previous work on the problem and its generalizations [7,11,16]. If either the number of machines or the number of different resource requirements is bounded by a constant, these techniques allow us even to obtain a PTAS.

As mentioned above, a simple list scheduler as in [7] achieves an approximation guarantee of 3. In contrast, our $(2+\epsilon)$-approximation algorithm is rather complex, but gets close to the $(\frac{3}{2}-\epsilon)$-inapproximability bound of the problem. The extra effort to achieve an improvement of $1-\epsilon$ in the approximation ratio might not appear to be justified for practical purposes. However, considering that the $(2+\epsilon)$-guarantee is close to the inapproximability bound, we believe that our contribution to the theoretical understanding of the problem is well worth it.

1.3 Formal Definition and Notation

We define the *resource constrained scheduling problem* formally. An instance consists of an integer m and a set of jobs $\mathcal{J}$ where each job $j \in \mathcal{J}$ is characterized by its processing time $p(j)$ and its resource requirement $r(j)$. For notational convenience, we usually denote an instance just by the job set $\mathcal{J}$, implicitly assuming that m, p and r are given as well. For a natural number $\ell \in \mathbb{N}$, we write $[\ell] := \{0, \dots, \ell-1\}$. A *slot* is a time-interval $[t_1, t_2)$ with $t_2 \geq t_1$. A *machine slot* $(k, t_1, t_2) \in [m] \times \mathbb{Q}_{\geq 0} \times \mathbb{Q}_{\geq 0}$ is a slot assigned to a machine k. Let $\mathcal{M}$ denote the set of all machine slots. For notational convenience we often identify machine slots s with the interval they represent: We write $|s|$ instead of $|[t_1, t_2)|$, and $t \in s$ instead of $t \in [t_1, t_2)$. The same applies for unions and intersections of machine slots. A *schedule* is a map $\varphi : \mathcal{J} \to \mathcal{M}$ that assigns a machine slot to each job $j \in \mathcal{J}$. We call a schedule *feasible*, if the length of the assigned machine slot to each job is sufficient, assigned machine slots on the same machine are pairwise non-intersecting, and the resource constraint is satisfied. Formally, a schedule is feasible if $|\varphi(j)| \geq p(j)$ for all $j \in \mathcal{J}$, we have that $\varphi(j) \cap \varphi(j') = \emptyset$ for all $j, j' \in \mathcal{J}$ assigned to the same machine, and $\sum_{j \in \mathcal{J}:\ t \in \varphi(j)} r(j) \leq 1$ for all $t \in \mathbb{Q}_{\geq 0}$. If not noted otherwise, when talking about schedules we always mean feasible schedules. The *makespan* $T(\varphi)$ of a schedule φ is the largest endpoint of an assigned machine slot, i.e. $T(\varphi) := \sup \bigcup_{j \in \mathcal{J}} \varphi(j)$. With $OPT(\mathcal{J})$ we denote the optimal makespan of an instance $\mathcal{J}$. We write $c = O(1)$ to denote that c is some constant. If the constant depends on the parameter ϵ, we write $c = O_\epsilon(1)$ instead.

2 A List Scheduler

Before we describe the main result, we discuss the following simple list scheduling algorithm introduced by Garey and Graham [7]: On input $\mathcal{J}$, we iteratively compute a schedule by adding the jobs one by one as follows. While there are unassigned jobs, determine the smallest time t a not yet assigned job could be placed into the schedule without making it infeasible. Assign it by allocating a suitable machine slot $(k, t, t + p(j))$. If there are several candidates, choose an arbitrary one. Garey and Graham proved an approximation guarantee of $3 - \frac{3}{m}$, and give examples that show that the analysis is tight. Hence this list schedulers is insufficient for our purposes as we are aiming for a $(2 + \epsilon)$-approximation algorithm. Nevertheless, it will prove useful as a subroutine of our main algorithm later, however in a slight variation. Instead of starting with an empty schedule, we start with a partial schedule φ' that schedules a subset of jobs $\mathcal{J}' \subseteq \mathcal{J}$. The list scheduler is then used to complete the schedule by adding the remaining jobs $\mathcal{J} \setminus \mathcal{J}'$ one by one as described above. We will now derive a simple upper bound on the makespan of schedules generated by this algorithm. It depends on three parameters that we define now. Given an instance $\mathcal{J}$, we set

$$\bar{P}(\mathcal{J}) := \sum_{j \in \mathcal{J}} p(j)/m \text{ and } \bar{R}(\mathcal{J}) := \sum_{j \in \mathcal{J}} p(j)r(j).$$

Both values are lower bounds on the optimal makespan (which was also observed by Garey and Graham), a fact that will come in handy later.

Lemma 1. $OPT(\mathcal{J}) \geq \max\{\bar{P}(\mathcal{J}), \bar{R}(\mathcal{J})\}$.

Proof (sketch). The first bound follows from the fact that no schedule can do better than keeping all machines busy at all times. The second bound follows from the resource constraint limiting the total resource consumption to 1. □

While the parameters $\bar{P}(\mathcal{J})$ and $\bar{R}(\mathcal{J})$ only depend on the instance, the third parameter depends on the given partial schedule φ'. An *activation point* of φ' is a time-index t where some machine becomes busy that was idle before, or the resource consumption increases. Let $A(\varphi')$ denote the number of activation points of a schedule φ'. We now derive the following bound on the makespan of the schedule computed by our list scheduler when used to complete a partial schedule φ'.

Lemma 2. *Let $\mathcal{J}' \subseteq \mathcal{J}$ and let φ' be a schedule for $\mathcal{J}'$. Let p and r be such that $p(j) \leq p$ and $r(j) \leq r$ for all $j \in \mathcal{J} \setminus \mathcal{J}'$. In polynomial time we can compute a schedule φ for $\mathcal{J}$ with $T(\varphi) \leq \max\{T(\varphi'), \bar{P}(\mathcal{J}) + \frac{1}{1-r}\bar{R}(\mathcal{J}) + (A(\varphi') + 1) \cdot p\}$.*

Proof. If the makespan of φ is $T(\varphi')$, we are done. Hence assume that the makespan increased when adding the jobs of $\mathcal{J} \setminus \mathcal{J}'$. Let $j^* \in \mathcal{J} \setminus \mathcal{J}'$ be a job that finishes last, i.e. it determines the makespan.

Let $t \leq T(\varphi)$ be a time index. Observe that we are always in (at least) one of the following three cases: (a) All machines are busy at time t, (b) Job j^* is

active, (c) A machine is idle and job j^* is not active. Clearly the total time case (a) can apply is bounded by $\bar{P}(\mathcal{J})$. Trivially, the total time spent in case (b) is at most $p(j^*) \leq p$. Now assume that we are in case (c),and there is *no* activation point in the interval $[t, t+p)$. Then the resource consumption at time t is at least $1-r$ as otherwise the algorithm could and would have scheduled job j^* or some other job at time t on the idle machine. It follows that the total time spent in case (c) with no upcoming activation point in the interval $[t, t+p)$ is at most $\frac{1}{1-r}\bar{R}(\mathcal{J})$. The remaining time spent in case (c) not yet accounted for is then $A(\varphi') \cdot p$. Summing up the individual bounds for all cases, we get the desired running time bound. □

Observe that if both r and p are "very small" and $A(\varphi')$ is "not too large", the second term of the statement from the lemma gets arbitrarily close to $2 \cdot OPT(\mathcal{J})$. This is why the algorithm will be useful as a subprocedure of our $(2+\epsilon)$-approximation algorithm. We will use the algorithm also to schedule some very small sub-instances separately. To do so, we will rely on the following bound.

Corollary 1. *Given an instance $\mathcal{J}$, and let $p := \max\{p(j) \ : j \in \mathcal{J}\}$. In polynomial time we can compute a schedule φ for $\mathcal{J}$ with $T(\varphi) \leq \bar{P}(\mathcal{J}) + 2\bar{R}(\mathcal{J}) + 2p$.*

Proof. Let $\mathcal{J}' := \{j \in \mathcal{J} \ : \ r(j) > \frac{1}{2}\}$. Let φ' be a schedule for $\mathcal{J}'$ that schedules all jobs of that instance sequentially on the first machine, sorted non-increasingly by resource requirement. Observe that the schedule has makespan at most $2\bar{R}(\mathcal{J})$ as the resource usage is at least $\frac{1}{2}$ at all times. Moreover, the schedule has only one activation point. Hence applying Lemma 2 to this schedule proves the claim. □

3 The $(2+\varepsilon)$-Approximation Algorithm

Fix a constant $\epsilon > 0$. We present a polynomial time algorithm with the following property. For any instance $\mathcal{J}$ with $OPT(\mathcal{J}) \leq 1$, it computes a feasible schedule of makespan at most $2 + O(1) \cdot \epsilon$. Such an algorithm can easily be turned into a $(2+\epsilon)$-approximation algorithm using a binary search framework. We simplify the problem even further by considering only instances that are (γ, M)-restricted.

Definition 1. *Let $1 \geq \gamma > 0$, $M > 1$. An instance $\mathcal{J}$ is* (γ, M)-restricted *if for each job j it holds that either $p(j) \geq \gamma$ or $p(j) < \gamma/M$.*

In (γ, M)-restricted instances, we classify jobs j with $p(j) \geq \gamma$ as *large* and jobs j with $p(j) < \gamma/M$ as *small*. Every large job is at least M times as long as any small job. This fact will become handy later. For a (γ, M)-restricted instance $\mathcal{J}$, let $\mathcal{J}_{\text{large}}$ be the set of large and $\mathcal{J}_{\text{small}}$ be the set of small jobs respectively. The following lemma justifies that we can restrict ourselves to restricted instances.

Lemma 3. *For any constants $\epsilon > 0$, $M > 1$, there is a constant $\gamma^*_{\epsilon,M}$ such that for any instance $\mathcal{J}$, in polynomial time we can find a value $\gamma \geq \gamma^*_{\epsilon,M}$ and a partition of the instance $\mathcal{J} = \mathcal{J}_1 \dot{\cup} \mathcal{J}_2$ so that $\mathcal{J}_1$ is (γ, M)-restricted and for J_2 the list scheduler computes a schedule of makespan at most $O(1) \cdot \epsilon \cdot OPT(\mathcal{J})$.*

Proof (sketch). Choose $O_\epsilon(1)$ many disjoint sub-intervals from $(0, O_\epsilon(1)]$ so that each right endpoint is M times larger than its left endpoint. By the pigeonhole-principle, the jobs of one of the intervals have negligible $\bar{P}$ and $\bar{R}$ values. Call them $\mathcal{J}_2$. Using the greedy list scheduler from Section 2, by Lemma 1 and Corollary 1 the list scheduler computes a schedule of makespan at most $O(1) \cdot \epsilon \cdot OPT(\mathcal{J})$ for $\mathcal{J}_2$. □

We set $M := \epsilon^{-3}$. For the remainder of this section, let $\mathcal{J}$ be a (γ, M)-restricted instance with $OPT(\mathcal{J}) \leq 1$. The algorithm to schedule restricted instances is in itself composed of several steps. The rough outline is as follows. We first consider only the large jobs. We compute a schedule of makespan $1 + \epsilon$ for all except a constant number of jobs. Next we compute a set of candidate schedules for the remaining jobs. Each of them has makespan $1 + \epsilon$ as well. We can guarantee that for at least one of the schedules, all small jobs can be added without increasing the makespan (we do not know which one though, so we try them all). As the problem of adding the small jobs is NP-hard on its own, we again rely on approximations by allowing the makespan to increase "a bit". Adding small jobs takes place in two steps. First, for each candidate schedule we add the *fat* jobs, i.e. small jobs of "large" resource requirement. This is successful at least for one candidate. We then concatenate the successful schedule with the separate schedule for only large jobs that we computed in the first step. Finally we complete the schedule by adding the so far not yet scheduled *thin* jobs, i.e., small jobs with "small" resource requirement, using a list scheduler. In total this will result in a schedule of makespan $2 + O(1) \cdot \epsilon$. In summary, we have the following steps.

Step 1a: Compute schedule φ_1 for "almost" all large jobs.
Step 1b: Compute set of candidate schedules for remaining large jobs.
Step 2a: For each candidate, try to add small fat jobs.
Concatenate φ_1 with successful schedule from step 2a.
Step 2b: Add small thin jobs to concatenated schedule using the list scheduler.

Step 1a: Scheduling "almost" all large jobs. We discretize the scheduling decisions for large jobs by setting $\delta := \epsilon \cdot \gamma/2$ and requiring the assigned machine slots to start and end at integer multiples of δ. I.e., for each $j \in \mathcal{J}_{\text{large}}$, its machine slot should be of the form $(k, \ell_1 \cdot \delta, \ell_2 \cdot \delta)$ with $\ell_1, \ell_2 \in \mathbb{N}_0$. We call schedules whose large jobs have this property δ-*atomic*. The following lemma asserts that it is sufficient to consider only δ-atomic schedules.

Lemma 4. *There exists a δ-atomic schedule for $\mathcal{J}$ whose makespan is at most $(1 + \epsilon) \cdot OPT(\mathcal{J}) \leq 1 + \epsilon$.*

Proof. Take an optimal schedule φ. Define another schedule φ' by setting $\varphi'(j) := (k, (1 + \epsilon) \cdot a, (1 + \epsilon) \cdot b)$ for each $\varphi(j) = (k, a, b)$. Observe that φ' is feasible and has a makespan of at most $(1 + \epsilon)T(\varphi) = (1 + \epsilon)OPT(\mathcal{J})$. For every large job $j \in \mathcal{J}_{\text{large}}$, its machine slot $\varphi'(j)$ has length $|\varphi'(j)| = (1 + \epsilon)|\varphi(j)| \geq p(j) + 2\delta$. The last inequality is by the fact that j is large and hence $\epsilon \cdot p(j) \geq \epsilon\gamma = 2\delta$.

Hence, we can round the starting times of large machine slots up and their ending times down to multiples of δ without decreasing their length below $p(j)$. This way we obtain a feasible δ-atomic schedule. □

We call a set of $|\mathcal{J}_{\text{large}}|$ many machine slots (without job assignment) a *template*. It is called *feasible* if it corresponds to a feasible schedule (i.e. there is a feasible schedule that uses the slots from the template). To compute a schedule for the large jobs, we will first find a feasible template, and later assign jobs to its slots. More precisely, we want to find the template corresponding to the schedule of makespan $1+\epsilon$ due to Lemma 4. To do so, we partition the timeline $[0, 1+\epsilon)$ into *frames* $F_\ell := [\delta \cdot (\ell - 1), \delta \cdot \ell)$ for $\ell \in \mathbb{N}$ and set $\mathcal{F} := \left\{F_\ell \ : \ 1 \leq \ell \leq \left\lceil \frac{1+\epsilon}{\delta} \right\rceil \right\}$. Observe that all large machine slots in our δ-atomic schedule are unions of frames from $\mathcal{F}$. Note that since $\gamma \geq \gamma^*_{\epsilon,M}$ (and $\gamma^*_{\epsilon,M}$ is a constant), we have that $|\mathcal{F}| = \left\lceil \frac{1+\epsilon}{\delta} \right\rceil = O_\epsilon(1)$ is constant. This allows us to find a feasible template by enumeration:

Lemma 5. *In polynomial time we can compute a polynomial number of candidate templates. At least one of them is feasible.*

Proof. In a feasible δ-atomic schedule there are at most $|\mathcal{F}|$ many large jobs on a single machine. For each job, start and end times are chosen from $|\mathcal{F}|+1$ many possibilities. Hence, there are at most $Q := \binom{|\mathcal{F}|+1}{2}^{|\mathcal{F}|}$ feasible combinations of machine slots for one machine. Up to permutation of machines, we can describe every template by specifying how many machines use each of the Q possibilities. This results in at most $m^Q = m^{O_\epsilon(1)}$ many candidate templates. □

To compute the schedule, we repeat the following procedure for each of the candidate templates. It will be successful for any feasible template. Let $\mathcal{T}$ be a template. We use a linear program to compute an assignment of large jobs to machine slots from $\mathcal{T}$. Let $\mathcal{I}$ denote the set of all slots from $\mathcal{T}$ (i.e. the time intervals without the machine assignment). For each slot $s \in \mathcal{I}$, let μ_I denote the number of machines slots from the template $\mathcal{T}$ that use it. We model the problem of assigning jobs to intervals with the following linear program.

$$\begin{aligned}
\sum_{I \in \mathcal{I},\ |I| \geq p(j)} x_{j,I} \ &= 1 \quad \forall j \in \mathcal{J}_{\text{large}} \qquad (1)\\
\sum_{j \in \mathcal{J}_{\text{large}}} x_{j,I} \ &\leq \mu_I \quad \forall I \in \mathcal{I}\\
\sum_{I \in \mathcal{I}, F \subseteq I} \sum_{j \in \mathcal{J}_{\text{large}}} r(j) x_{j,I} \ &\leq 1 \quad \forall F \in \mathcal{F}\\
x_{j,I} \ &\geq 0 \quad \forall j \in \mathcal{J}_{\text{large}}\ \forall I \in \mathcal{I}
\end{aligned}$$

The variables $x_{j,I}$ model the assignment of jobs to intervals, where $x_{j,I} = 1$ means that job i is assigned a machine slot with time interval I.

Lemma 6. *Given a feasible template, in polynomial time we can compute a δ-atomic schedule of makespan $(1+\epsilon)$ that schedules all except for $|\mathcal{I}| + |\mathcal{F}|$ many large jobs. The schedule has at most $O(1)\frac{1}{\epsilon\gamma}$ activation points.*

Proof. An extreme point solution of the above linear program has at most $|\mathcal{J}_{\text{large}}| + |\mathcal{I}| + |\mathcal{F}|$ many non-zero entries. Hence, due to Constraints (1) at most $|\mathcal{I}| + |\mathcal{F}|$ jobs are fractionally assigned. We output the schedule obtained for the integrally assigned jobs. Since it is δ-atomic, there are at most $|\mathcal{F}|$ activation points. □

Note that $|\mathcal{I}| = O_\epsilon(1)$ just like $|\mathcal{F}|$. Hence all except for a constant number of jobs are scheduled.

Step 1b: Scheduling the remaining large jobs. If there are only constantly many large jobs to schedule, we can enumerate all possible δ-atomic schedules in polynomial time. Importantly, one of these schedules is *extendable.*

Definition 2. *Let $\mathcal{J}$ be a set of jobs and let φ be a schedule for a subset $\mathcal{J}' \subseteq \mathcal{J}$. The schedule φ is* extendable for $\mathcal{J}$ *if the jobs $\mathcal{J} \setminus \mathcal{J}'$ can be added to φ without increasing the makespan.*

Lemma 7. *Let $\mathcal{J}'_{\text{large}} \subseteq \mathcal{J}$ be a set of large jobs with $|\mathcal{J}'_{\text{large}}| = O_\epsilon(1)$. In polynomial time we can compute a set of δ-atomic schedules of makespan $1+\epsilon$. Each of them is feasible for the sub-instance $\mathcal{J}'_{\text{large}}$. At least one of them is extendable for $\mathcal{J}$. The number of activation points of each schedule is at most $O(1) \cdot \frac{1}{\epsilon\gamma}$.*

Proof (sketch). We can restrict to only using the first $|\mathcal{J}'_{\text{large}}| = O_\epsilon(1)$ machines. Hence there are only constantly many machine slots to consider. □

Step 2a: Adding small fat jobs. We now describe how to add small jobs to a δ-atomic schedule for large jobs. As mentioned previously, there are two kinds of small jobs that need to be treated differently. Define $\beta := \epsilon$. We say that a small job is *fat* if $r(j) \geq \beta$. Otherwise it is *thin.*

We are now in the following situation: We consider a sub-instance $\mathcal{J}' \subseteq \mathcal{J}$ consisting of some large jobs $\mathcal{J}'_{\text{large}}$ and all tiny fat jobs $\mathcal{J}'_{\text{small}}$. Note that $\mathcal{J}'_{\text{small}}$ does *not* contain the small thin jobs. We have a schedule φ_2 for $\mathcal{J}'_{\text{large}}$ of makespan $1 + \epsilon$, and we want to add the small fat jobs $\mathcal{J}'_{\text{small}}$. For simplicity we assume that φ_2 is extendable and show that in this case, the algorithm is successful. The roadmap is as follows: We first round the resource requirements so that only a constant number of different values remain. We then compute an "optimal invalid" schedule for a transformed instance. It is invalid because it allows preemption, migration and parallelization. However, in the end this invalid schedule allows us to compute a "good" feasible schedule for our instance.

The resource rounding is done with a linear grouping technique similar in spirit as the technique employed in [5] for Bin-Packing. We first sort the jobs from $\mathcal{J}'_{\text{small}}$ non-decreasingly by resource requirement. Let $\mathcal{J}'_{\text{small}} = \{j_1, \ldots, j_n\}$ be in this order. For $K := \lceil 1/\epsilon^2 \rceil$, we divide the jobs into K groups as follows: Figuratively, we take a schedule where the jobs from $\mathcal{J}'_{\text{small}}$ are scheduled sequentially in the order from above, slice it into K intervals of equal length and define that $\mathcal{J}_i$ is the group of jobs that are completely contained in interval i. Group $\mathcal{J}_0$ is the set of jobs that are cut when defining the intervals. Formally,

define $\mathcal{J}_i := \{j_k \,:\, (i-1)\cdot p(\mathcal{J}'_{\text{small}})/K \leq \sum_{\ell=1}^{k-1} p(j_\ell) \leq i \cdot p(\mathcal{J}'_{\text{small}})/K - p(j_k)\}$ for $i = 1, \ldots, K$, and $\mathcal{J}_0 := \mathcal{J}'_{\text{small}} \setminus \bigcup_{i=1}^{K} \mathcal{J}_i$. By construction, we obtain the following properties of the set system.

Lemma 8. *We have $|\mathcal{J}_0| \leq K$, $p(\mathcal{J}_i) \leq p(\mathcal{J}'_{\text{small}})/K$ for all $i = 1, \ldots, K$, and for any two jobs $j \in \mathcal{J}_i$ and $j' \in \mathcal{J}_{i'}$ with $1 \leq i < i'$ it holds that $r(j) \leq r(j')$.*

Based on this grouping, we define a set of jobs $\tilde{\mathcal{J}}'_{\text{small}} := \{g_1, ..., g_{K-1}\}$ by setting $p(g_i) := p(\mathcal{J}'_{\text{small}})/K$ and $r(g_i) := \max\{r(j) \,|\, j \in \mathcal{J}_i\}$ for each $i \in \{1, ..., K\}$. For the resulting instance $\tilde{\mathcal{J}}' := \mathcal{J}'_{\text{large}} \cup \tilde{\mathcal{J}}'_{\text{small}}$, we later compute an "invalid" schedule. The jobs g_i are going to act as placeholders to fill in the jobs from $\mathcal{J}_i$ in the final solution later. The groups $\mathcal{J}_0$ and $\mathcal{J}_K$ are treated differently. They can be scheduled separately:

Lemma 9. *The jobs in $\mathcal{J}_0 \cup \mathcal{J}_K$ can be scheduled with makespan $O(1) \cdot \epsilon$.*

Proof. Because the jobs in $\mathcal{J}_0$ are small we have $p(\mathcal{J}_0) \leq K \cdot \frac{\gamma}{M} \leq O(1) \cdot \epsilon$. For $\mathcal{J}_K$, recall the lower bound $\bar{R}$ from Section 2. We get

$$1 \geq OPT(\mathcal{J}') \geq \bar{R}(\mathcal{J}') \geq \bar{R}(\mathcal{J}'_{\text{small}}) \geq \beta \cdot p(\mathcal{J}'_{\text{small}}) \geq \beta \cdot K \cdot p(\mathcal{J}_K) \geq p(\mathcal{J}_K)/\epsilon.$$

We conclude that if we schedule the jobs $\mathcal{J}_0 \cup \mathcal{J}_K$ sequentially on one machine, the makespan is $O(1) \cdot \epsilon$. □

We now discuss how to compute the "invalid" helper schedule. We call it a *relaxed* schedule. In relaxed schedules, we allow jobs to be preempted, migrated, and executed in parallel. However the same rules for feasibility apply as for "real" schedules. For simplicity of presentation, we refrain from a formal definition of relaxed schedules. We first prove the existence of a relaxed schedule for $\tilde{\mathcal{J}}'$.

Lemma 10. *If φ_2 is extendable for $\mathcal{J}'$, then it is extendable (as a relaxed schedule) for $\tilde{\mathcal{J}}'$.*

Proof (sketch). Let $\bar{\varphi}_2$ be an extension of φ_2 for $\mathcal{J}'$. Recall the illustration of slicing a sequential schedule for $\mathcal{J}'_{\text{small}}$ into K equally sized intervals. To construct a relaxed schedule for $\tilde{\mathcal{J}}'$, for each i we use the jobs from interval $i+1$ as a template to fill in the job g_i. Note that this might include some fractions of jobs from that interval which are not included in $\mathcal{J}_i$ but in $\mathcal{J}_0$. □

We now show how to compute such a relaxed schedule for $\tilde{\mathcal{J}}'$. We again resort to linear programming. Note that in order to extend φ_2 we can use only resources "left over" by the large jobs $\mathcal{J}'_{\text{large}}$. Based on φ_2, for each frame $F \in \mathcal{F}$ let $r(F)$ denote the amount of resources available for non-large jobs, and let $m(F)$ denote the number of machines available. Note that these quantities are the same throughout a frame as the schedule φ_2 is δ-atomic. We will use these remaining resources to schedule small jobs of the instance.

Every possibility of small jobs being simultaneously active during frame $F \in \mathcal{F}$ can be described by a characteristic vector $\chi \in [m]^{K-1}$ such that $\sum_{i=1}^{K-1} \chi_i r(g_i) \leq r(F)$ and $\sum_i \chi_i \leq m(F)$. Let $\mathcal{C}(F)$ be the set of all such vectors.

For each job g_i, the entry χ_i specifies the number of machines used for executing job g_i in parallel. We call a vector $\chi \in \mathcal{C}(F)$ a *job configuration*. This allows us to formulate a "configuration-LP" that packs job configurations to frames and ensures that all small jobs are covered. Denote by CONF-LP the following linear program.

$$\begin{aligned} \textstyle\sum_{F\in\mathcal{F}} \sum_{\chi\in\mathcal{C}(F)} \chi_i x_{F\chi} &\geq p(g_i) \quad \forall i = 1,\dots,K-1 && (2)\\ \textstyle\sum_{\chi\in\mathcal{C}(F)} x_{F\chi} &\leq \delta \quad \forall F \in \mathcal{F} && (3)\\ x_{F\chi} &\geq 0 \quad \forall F \in \mathcal{F}\ \forall \chi \in \mathcal{C}(F) \end{aligned}$$

The variable $x_{F\chi}$ models for how much time configuration χ should be used within frame F. Due to Lemma 10 we know that CONF-LP has a solution and an extreme point solution fulfills the properties of the following lemma.

Lemma 11. *There is a polynomial time algorithm which computes a relaxed schedule for $\tilde{\mathcal{J}}'$ which extends φ_2. In particular, in polynomial time we can compute a solution to CONF-LP with at most $K + |\mathcal{F}|$ non-zero variables.*

Based on the solution to CONF-LP we construct a non-relaxed schedule φ_2' for $\mathcal{J}'$. We partition each frame $F \in \mathcal{F}$ into subframes, each subframe corresponds to a positive variable $x_{F\chi}$ and has length $x_{F\chi}$. The packing constraints (3) ensure that we can do that. Now for each subframe and each $i \in [K]$, create χ_i many machine slots (assign them to free machines greedily) and reserve them for jobs of group $\mathcal{J}_i$. The machine slots created in this way act as placeholders and, to avoid confusion, we will refer to them as *placeholder slots*. Our construction ensures that if we pack jobs of group $\mathcal{J}_i$ arbitrarily to its reserved placeholder slots, we will not violate the resource requirement (as by Lemma 8, the resource requirement of all jobs from $\mathcal{J}_i$ is at most $r(g_i)$). As the covering constraints (2) are satisfied, the total amount of execution time reserved for each group $\mathcal{J}_i$ is at least $p(g_i)$ which by definition and Lemma 8 is at least $\sum_{j\in\mathcal{J}_i} p(j)$.

Now for each group $\mathcal{J}_i$, $i = 1 \dots K-1$ and each job $j \in \mathcal{J}_i$, select an arbitrary placeholder slot reserved for group $\mathcal{J}_i$ which has a positive amount of space left and assign j to it. By the observations from above, it is clear that this algorithm manages to assign all jobs. However, it might produce an infeasible solution as some placeholder slots might be over-packed. We can repair this as follows: For each subframe (i.e. for each positive variable in the LP-solution), and for each placeholder slot belonging to this subframe, pick the job added last and remove it. Now the placeholder slots are not over-packed anymore, i.e. the resulting schedule is feasible. For the removed jobs, observe that those that are taken from the same subframe can be scheduled in parallel. Hence, because they are small, we can schedule them separately in a time-frame of γ/M timeunits. As there are at most $|\mathcal{F}| + K$ many nonzeros the LP solution due to Lemma 11, we conclude that the increase of the makespan is at most $\gamma/M \cdot (|\mathcal{F}|+K) = O(1)\cdot\epsilon$. Hence, in summary we get the following result:

Lemma 12. *Given φ_2, in polynomial time we can compute a schedule φ_2' for $\mathcal{J}'$ with $T(\varphi_2') \leq 1 + O(1)\cdot\epsilon$. Moreover we have $A(\varphi_2') \leq O(1)\cdot\frac{1}{\epsilon^2\gamma}$.*

Proof. The makespan increase due to the above procedure, as well as the length of the schedules for $\mathcal{J}_0$ and $\mathcal{J}_K$, is bounded by $O(1) \cdot \epsilon$. To see the bound on the activation points, observe that new activation points can only be introduced for each subframe from the construction above. □

Step 2b: Adding small thin jobs. Let φ_1 and φ_2' be the schedules obtained due to Lemma 6 and Lemma 12, respectively. Observe that if we concatenate φ_1 and φ_2', we obtain a schedule of makespan $2 + O(1) \cdot \epsilon$ that schedules all jobs from $\mathcal{J}$ except for the small thin ones. The number of activation points of this schedule is $O(1) \cdot 1/(\epsilon^2\gamma)$.

We use the list scheduler from Section 2 to complete the schedule. Applying Lemma 2 in this situation, we can set $p := \gamma/M$ and $r := \beta$. Hence we obtain a full schedule of makespan

$$\max\left\{2 + O(1) \cdot \epsilon, \bar{P}(\mathcal{J}) + \frac{1}{1-\beta}\bar{R}(\mathcal{J}) + O(1) \cdot \frac{1}{\epsilon^2\gamma}\gamma/M\right\} = 2 + O(1) \cdot \epsilon.$$

With the discussion from the beginning of this section we conclude:

Theorem 1. *There is a $(2+\epsilon)$-approximation algorithm for the resource constraint scheduling problem.*

4 Complexity and Special Cases

For complexity of the resource constrained scheduling problem, consider the following reduction from Bin-Packing: for each given item $a_j \in \{a_1, ..., a_n\}$ introduce a job j with $p(j) = 1$ and $r(j) = a_j$ and define $m := n$. Then, the inapproximability for Bin-Packing [14] carries over to our problem.

Theorem 2. *The resource constrained scheduling problem is NP-hard to approximate with a factor of $\frac{3}{2} - \epsilon$ for any $\epsilon > 0$.*

In the case where the number of machines is bounded by a constant, one can get a PTAS by adapting the components of the $(2+\epsilon)$-approximation algorithm. The same is true if instead we bound the number of different resource requirements by a constant. Essentially there are two steps of the algorithm that need to be improved. The first one is the way we deal with large jobs: Instead of creating two schedules of makespan $1 + \epsilon$ and concatenating them, we need to treat all large jobs at the same time. The second issue is the use of the greedy list scheduler to schedule the thin/small jobs: No matter how we tweak the parameters, this algorithms performance guarantee will not get better than $2 + \epsilon$. Both issues can be addressed for both special cases, but in different ways.

For space reasons we omit further details.

Theorem 3. *For any constant C, there is a PTAS for the resource constrained scheduling problem for instances $\mathcal{J}$ with $m \leq C$ or with $|\{r(j) \ : \ j \in \mathcal{J}\}| \leq C$.*

Acknowledgements. We would like to thank Marco Di Summa, Friedrich Eisenbrand, Thomas Rothvoß, and José Verschae for helpful discussions.

References

1. C. Artigues, S. Demassey, and E. Néron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications.* ISTE, 2010.
2. J. Blazewicz, J.K. Lenstra, and A.H.G.Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11 – 24, 1983.
3. C. Chekuri and S. Khanna. On multi-dimensional packing problems. In *Proceedings of the 10th annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*, pages 185–194. SIAM, 1999.
4. E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7:1–17, 1978.
5. W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.
6. D. K. Friesen. Tighter bounds for the multifit processor scheduling algorithm. *SIAM Journal on Computing*, 13:170–181, 1984.
7. M. R. Garey and R. L. Grahams. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4:187–200, 1975.
8. M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 1975.
9. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
10. R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:263–269, 1969.
11. A. Grigoriev, M. Sviridenko, and M. Uetz. Machine scheduling with resource dependent processing times. *Mathematical Programming*, 110:209–228, 2007.
12. S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of OR*, 207:1–14, 2010.
13. D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* Thomson, 1996.
14. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.
15. K. Jansen and L. Porkolab. On preemptive resource constrained scheduling: Polynomial-time approximation schemes. In *Integer Programming and Combinatorial Optimization*, volume 2337 of *Lecture Notes in Computer Science*, pages 329–349. Springer Berlin / Heidelberg, 2006.
16. H. Kellerer. An approximation algorithm for identical parallel machine scheduling with resource dependent processing times. *OR Letters*, 36:157–159, 2008.
17. C. Kenyon and E. Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, 2000.
18. M. A. Langston. *Processor scheduling with improved heuristic algorithms.* PhD thesis, Texas A&M University, 1981.
19. A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141:241–252, 2002.
20. S. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23:116–127, 1976.