Approximation Algorithms for Parallel Machine Scheduling with Speed-Up Resources

Lin Chen[1], Deshi Ye[2], and Guochuan Zhang[3]

1 **Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary**
chenlin198662@gmail.com
2 **Zhejiang University, College of Computer Science, Hangzhou, China**
yedeshi@zju.edu.cn
3 **Zhejiang University, College of Computer Science, Hangzhou, China**
zgc@zju.edu.cn

Abstract

We consider the problem of scheduling with renewable speed-up resources. Given m identical machines, n jobs and c different discrete resources, the task is to schedule each job non-preemptively onto one of the machines so as to minimize the makespan. In our problem, a job has its original processing time, which could be reduced by utilizing one of the resources. As resources are different, the amount of the time reduced for each job is different depending on the resource it uses. Once a resource is being used by one job, it can not be used simultaneously by any other job until this job is finished, hence the scheduler should take into account the job-to-machine assignment together with the resource-to-job assignment.

We observe that, the classical unrelated machine scheduling problem is actually a special case of our problem when $m = c$, i.e., the number of resources equals the number of machines. Extending the techniques for the unrelated machine scheduling, we give a 2-approximation algorithm when both m and c are part of the input. We then consider two special cases for the problem, with m or c being a constant, and derive PTASes (Polynomial Time Approximation Schemes) respectively. We also establish the relationship between the two parameters m and c, through which we are able to transform the PTAS for the case when m is constant to the case when c is a constant. The relationship between the two parameters reveals the structure within the problem, and may be of independent interest.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases approximation algorithms, scheduling, linear programming

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2016.5

1 Introduction

We consider a natural generalization of the classical scheduling problem in which there are multiple different resources available. Each job has an original processing time which may be reduced by utilizing one of the resources. Since resources are different, the amount of the time reduced for each job is different depending on the resource it uses. It is a hard constraint that the usage of the resources does not conflict, that is, once a specific resource is being used by some job, it becomes unavailable to all the other jobs until this job is completed. Consequently a good schedule not only needs to choose the right machine and resource for each job but also needs to sequence jobs on each machine in a proper way such that the usage of each resource does not conflict.

The problem arises naturally in production logistics where a task not only relies on the machine but also on the personnel it is assigned to. It also has its own right from a theoretical point of view. As we will provide details later, this problem is a special case of the general multiprocessor task scheduling problem ($P|set|C_{max}$), which does *not* admit any constant ratio approximation algorithm [2], and meanwhile a generalization of the unrelated machine scheduling problem ($R||C_{max}$), for which a 2-approximation algorithm stands for more than two decades [11].

We give a formal description of our model. There are m parallel identical machines, n jobs and c discrete resources. Each job j has a processing time p_j and has to be processed *non-preemptively* on one of the machines. This processing time might be reduced by utilizing a resource. Specifically, when resource k is allocated to job j then its processing time becomes p_{jk}. At most one resource could be allocated to a job and once a resource, say resource k, is being used by job j, then it can no longer be used by any other job during the time interval where job j is processed. Throughout this paper, we do *not* necessarily require $p_{jk} \leq p_j$. We assume all parameters are taking integral values.

As we have described, in our model jobs could be processed with or without a resource. However, we always assume that each job is processed with a resource unless otherwise specified. Such an assumption causes no loss of generality since we could always introduce m dummy resources (that could not alter the processing time of any job), one for each machine, and jobs scheduled on a machine without a resource could then be viewed as processed with the dummy resource corresponding to this machine. This assumption works for the case that c, the number of resources, is part of the input. For the case that c is a constant, we return to the original assumption that the usage of resources is optional.

Related work. One special case of our problem with $c = 1$ and $m = 2$ is considered in [12], in which an FPTAS (Fully Polynomial Time Approximation Scheme) is derived. Another related problem is considered in [10], in which $c = 1$ again, but the machines are dedicated, i.e., for each job the processing machine is known in advance. For the two-machine case, they prove that the problem is NP-hard but admits an FPTAS. For an arbitrary number of machines, they give a 3/2-approximation algorithm. Moreover, a PTAS is designed for a constant number of machines.

Another closely related model is that a job can be given several resources and yet all resources are identical, so the processing time of each job does not depend on which resource but the number of resources it uses. For this problem on unrelated machines, Grigoriev et al. [3] give a $(3.75 + \varepsilon)$-approximation algorithm. On identical machines, Kellerer [9] gives a $(3.5 + \varepsilon)$-approximation algorithm, which is improved very recently by Jansen, Maack and Rau [4] to an asymptotic PTAS.

Our problem is a generalization of the classical unrelated machine scheduling problem, denoted as $R||C_{max}$, in which each job j has a (machine dependent) processing time p_{ij} if it is processed on machine i. Indeed, if the number of machines is equal to the number of resources, i.e. $m = c$, and $p_j = \infty$ (indeed, it suffices to have $p_j > \sum_{i,j'} p_{ij'}$, as in this case a schedule that does not process job j with any resource is never optimal), then our problem is equivalent to the unrelated machine scheduling problem. To see why, notice that given any feasible solution of our problem, we can rearrange jobs so that all jobs using the same resource, say, k, are scheduled on machine k. By doing so the makespan is not increased, and meanwhile the new solution is a feasible solution of the unrelated machine scheduling problem in which machine k is one of the unrelated machines which processes job j with time p_{jk}. The current best-known approximation ratio for the unrelated scheduling problem

is 2 if m is part of the input, whereas no approximation algorithm could achieve a ratio strictly better than 2, assuming $P \neq NP$ [11]. If m is a constant, an FPTAS exists [7] and its current best running time is $O(n) + (\log m/\varepsilon)^{O(m \log m)}$ [5].

Meanwhile, our problem is also a special case of the general multiprocessor task scheduling problem, denoted as $P|set|C_{max}$, in which each task (job), say, j, could be processed simultaneously on multiple machines, and its processing time is $p_{j,S}$ where S is the set of machines we choose to process it. To see why our problem is a special case, we view each resource as a special machine which we call a resource machine, and each job could either be processed on a normal machine with processing time p_j, or processed simultaneously on a normal machine i and some resource machine k, with $p_{j,\{i,k\}} = p_{jk}$. Thus our problem could be transformed to a multiprocessor task scheduling problem with $m + c$ machines. There is a PTAS for the general multiprocessor task scheduling problem if the number of machines is a constant, and no constant ratio approximation algorithm exists if the number of machines is part of the input [2][6]. This result implies that for our problem, if both the number of resources and the number of machines are constants, then there is a PTAS.

Our contribution. We study the scheduling problem with speed-up resources. As we have mentioned, it is an intermediate model between the general model $P|set|C_{max}$ and the classical unrelated machine scheduling $R||C_{max}$. We hope our research could bridge the study of these two well-known models and leads to a better understanding of them.

In this paper, we give the first 2-approximation algorithm when the number of machines m and resources c are both part of the input. We then consider two special cases with either m or c being a constant, and provide PTASes, respectively.

For the general case, we observe that the natural LP (Linear Programming) formulation of the problem has too many constraints, whereas its extreme point solution may split too many jobs which causes the classical rounding technique from [11] inapplicable. To handle this, the key idea is to iteratively remove constraints from the LP. We will iteratively modify the fractional solution such that either we get a new solution with fewer split jobs (which is the same as the traditional rounding), or we get a new solution for which we need fewer constraints to characterize it.

Given the lower bound of 1.5 for the unrelated machine scheduling problem $R||C_{max}$, and hence also for our problem, PTASes are only possible for special cases. We first consider the case when m is a constant and present a PTAS. To achieve this, we first determine (through enumeration) the scheduling of a constant number of jobs, and then handle the scheduling of remaining jobs by formulating it as an LP. We prove that, the LP we construct has a special structure which enforces that only a constant number among its huge number (non-constant) of constraints could become tight and correspond to an extreme point solution. Using this fact we are able to make use of the classical rounding technique from [11] to derive a PTAS.

We then consider the case when c is a constant. We establish an interesting relationship between this special case and the case when m is a constant. Indeed, we show that it suffices to consider solutions where all jobs using resources are scheduled only on $O(c)$ machines. Thus, this special case is a combination of scheduling with resources on $O(c)$ machines, together with the classical scheduling without resources on the remaining $m - O(c)$ machines.

2 General case

In this section, we consider the problem when the number of machines and resources, i.e., m and c, are both part of the input and give a 2-approximation algorithm. Recall that we can assume every job is processed with one resource.

We start with a natural LP formulation of this problem. Let $x_{ijk} = 1$ denote that job j is processed on machine i with resource k, and $x_{ijk} = 0$ otherwise. We first ignore the disjoint condition, i.e., the usage of each resource is not in conflict, and establish the following LP_r.

$$\sum_{i=1}^{m}\sum_{k=1}^{c} x_{ijk} = 1 \qquad \forall j \tag{1a}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} p_{jk}x_{ijk} \le T \qquad \forall k \tag{1b}$$

$$\sum_{j=1}^{n}\sum_{k=1}^{c} p_{jk}x_{ijk} \le T \qquad \forall i \tag{1c}$$

$$0 \le x_{ijk} \le 1 \tag{1d}$$

$$x_{ijk} = 0 \quad \text{if} \quad p_{ik} > T. \tag{1e}$$

Constraint (1a) ensures that every job is scheduled. Constraint (1b) ensures that the total processing time of jobs processed with the same resource k does not exceed T. Constraint (1c) ensures that the total processing time of jobs on each machine does not exceed T. Through binary search we can find the minimum integer $T = T^*$ such that LP_r admits a feasible solution, which is obviously a lower bound on the optimal solution. We denote by x^* the fractional solution of LP_r for $T = T^*$. Our rounding technique tries to make x^* into an integral solution so that (1b) and (1c) could be violated but not much, and the disjoint condition becomes respected, i.e., the disjoint condition which is not met by the LP_r will be achieved via rounding.

We remark that in the classical unrelated machine scheduling problem, the LP relaxation has only $n + m$ constraints, hence in its extreme point solution only m jobs would get split. By re-assigning these jobs to m machines, one per machine, a 2-approximation solution is derived. However, our LP_r has $n + m + c$ constraints. Its extreme point solution may cause $m + c$ jobs to be split, which is too many for carrying out the subsequent re-assigning procedure. To handle this, the key idea of our rounding procedure is to reduce the number of constraints via *well structured* fractional solutions.

In the following we define well structured solutions as well as its *rank*, both of which are crucial for our rounding procedure.

Given any fractional solution x when $T = T^*$, we can compute the fraction of job j processed with resource k through $x_{jk} = \sum_{i=1}^{m} x_{ijk} \in [0, 1]$. We call $\hat{x} = (x_{jk})$ a semi-solution to LP_r.

Obviously it holds for every resource k that $\sum_{j=1}^{n}\sum_{i=1}^{m} p_{jk}x_{ijk} = \sum_{j=1}^{n} p_{jk}x_{jk} \le T^*$. We say resource k is saturated with respect to $\hat{x}$ (and also x) if the equality holds. The number of saturated resources is called the degree of $\hat{x}$ (and x), and denoted as $d(\hat{x})$ $(= d(x))$.

We call $\sum_{j=1}^{n} p_{jk}x_{jk}$ the load of resource k. A semi-solution is called feasible, if the load of each resource is no greater than T^*, and the total load of all resources is no greater than mT^*. Obviously any feasible solution of LP_r implies a feasible semi-solution. On the other hand, any feasible semi-solution also implies a feasible solution of LP_r through the following *Direct Schedule*. (For simplicity we suppose resource 1 to resource $d = d(\hat{x})$ are saturated.)

Direct schedule

1. For $1 \le k \le d$, put (fractions of) jobs using resource k onto machine k.
2. For $k > d$, put (fractions of) jobs using unsaturated resources arbitrarily onto machine $d + 1$ to machine m such that the load of each machine is no greater than T^*.

Consequently, each solution has its corresponding semi-solution, and vice versa.

A semi-solution $\hat{x}$ (and also its corresponding solution x) is called *well structured*, if every job uses at most one unsaturated resource. We have the following lemma.

▶ **Lemma 1.** *Given a feasible semi-solution $\hat{x}$, a feasible well structured semi-solution $\hat{x'}$ can be constructed such that $d(\hat{x'}) \geq d(\hat{x})$.*

Proof. For each job j, if it uses two or more unsaturated resources, then $x_{j,k_1} > 0$ and $x_{j,k_2} > 0$ for some unsaturated resources k_1 and k_2. For simplicity we assume the total load of jobs using the two resources are L_1 and L_2 respectively.

Suppose without loss of generality $p_{j,k_1} \leq p_{j,k_2}$, we can choose $\delta = min\{x_{j,k_2}, \frac{T^*-L_1}{p_{j,k_1}}\}$ and replace x_{j,k_1} and x_{j,k_2} with $x_{j,k_1} + \delta$ and $x_{j,k_2} - \delta$ respectively. By doing so either resource k_1 becomes saturated or x_{j,k_2} becomes 0. In both cases the number of unsaturated resources used by job j is decreased by one. Notice that by altering $\hat{x}$ in this way, the total processing time of all jobs does not increase and the load of each resource is still no greater than T^*.

We iteratively apply the above procedure until every job uses at most one unsaturated resource, and a feasible well structured semi-solution $\hat{x'}$ with $d(\hat{x'}) \geq d(\hat{x})$ is derived. ◀

Now we are able to define the *rank* of a well structured (semi-)solution.

Again we assume that resource 1 to resource $d(= d(\hat{x}))$ are saturated. A bipartite graph $G(\hat{x}) = (V_1(\hat{x}) \cup V_2(\hat{x}), E(\hat{x}))$ corresponding to $\hat{x}$ is constructed in the following way.

We let $V_1(\hat{x}) = \{J_1, J_2, \cdots, J_n\}$ be the set of job nodes. If $d < m$, then $V_2(\hat{x}) = \{R_0, R_1, R_2, \cdots, R_d\}$ with nodes R_1 to R_d corresponding to the saturated resources, and R_0 corresponding to all the unsaturated resources. Otherwise $d = m$, then there is no unsaturated resources and $V_2(\hat{x}) = \{R_1, R_2, \cdots, R_d\}$. Let $x_{j0} = \sum_{k=d+1}^{c} \sum_{i=1}^{m} x_{ijk} = \sum_{k=d+1}^{c} x_{jk} \in [0, 1]$ if R_0 exists. For $0 \leq k \leq d$, there is an edge $(j, k) \in E(\hat{x})$ if and only if $0 < x_{jk} < 1$.

Additionally, if there are any isolated nodes, we simply remove them (from $V_1(\hat{x})$). This completes the construction of $G(\hat{x})$ for $\hat{x}$.

The rank of a well structured semi-solution $\hat{x}$ is defined as $r(\hat{x}) = |E(\hat{x})| + m - d(\hat{x})$.

The rank will serve as a potential function which allows us to iteratively round an initial feasible solution until a certain criterion is satisfied. Indeed, we have the following.

▶ **Lemma 2.** *Given a well structured semi-solution $\hat{x}$ and its corresponding graph $G(\hat{x}) = (V_1(\hat{x}) \cup V_2(\hat{x}), E(\hat{x}))$, let $G^i(\hat{x}) = (V_1^i(\hat{x}) \cup V_2^i(\hat{x}), E^i(\hat{x}))$ be any of its connected component. If $|E^i(\hat{x})| > |V_1^i(\hat{x})| + |V_2^i(\hat{x})|$, then a well structured solution $\hat{x'}$ of LP_r with a lower rank (i.e. $r(\hat{x'}) \leq r(\hat{x}) - 1$) can be constructed in polynomial time.*

Given the above lemma, we are able to show the following key theorem, which directly implies a 2-approximation algorithm.

▶ **Theorem 3.** *Let x^* be the fractional solution of LP_r as we define before. Then an integer solution $x^{IP} = (x_{ijk}^{IP})$ for the following Integer Programming can be derived in polynomial time.*

$$\sum_{i=1}^{m} \sum_{k=1}^{c} x_{ijk} = 1 \qquad \forall j$$

$$\sum_{j=1}^{n} \sum_{i=1}^{m} p_{jk} x_{ijk} \leq T^* + p_{max} \qquad \forall k$$

$$\sum_{j=1}^{n} \sum_{k=1}^{c} p_{jk} x_{ijk} \leq T^* + p_{max} \qquad \forall i$$

$$x_{ijk} \in \{0, 1\}$$

Here $p_{max} = \max_{j,k}\{p_{jk}|x^*_{ijk} \neq 0\}$. *And moreover, we could schedule jobs in a proper sequence on each machine so that the disjoint condition is also satisfied. Hence the makespan of the generated schedule is at most twice the optimal solution.*

3 The special case with a constant number of machines

In this section, we show that the problem admits a PTAS if the number of machines m is a given constant. Again, we assume that every job is processed with one resource.

Let $\bar{p}_j = \min\{p_{j1}, \cdots, p_{jc}\}$ be the shortest possible processing time of job j and we call it the critical processing time. The resource with which the processing time of job j achieves $\bar{p}_j$ is then called the critical resource of j (if there are multiple such resources, we choose arbitrarily one). We sort jobs so that $\bar{p}_1 \geq \bar{p}_2 \geq \cdots \geq \bar{p}_n$. Consider the first q jobs where q is some constant to be fixed later, we call them critical jobs, and others non-critical jobs.

Notice that we have a 2-approximation algorithm for the general case, thus we can compute some value T such that the makespan of the optimum solution (i.e., OPT) falls in $[T/2, T]$. We provide an algorithm such that given any $t \in [T/2, T]$ and a small positive $\epsilon > 0$, it either determines that $OPT > t$, or produces a feasible schedule with makespan bounded by $t + O(\epsilon T)$.

The basic idea of the algorithm is simple. We first determine (through enumeration) the scheduling of all the critical jobs. For each possible scheduling of the critical jobs, we set up an LP (Linear Programming) for the remaining jobs. If such an LP does not admit a feasible solution, then $OPT > t$. Otherwise we compute its extreme point solution and show that in such a solution only a constant number (depending on q and ϵ) of jobs get split. Finally we show how to construct a feasible schedule based on such a solution.

Configuration schedules. Let $\lambda = 1/\epsilon$ be an integer. Let $ST = \{0, T\epsilon/q, 2T\epsilon/q, \cdots, T + 2T\epsilon\}$ be the set of scaled time points (and hence $|ST| = \lambda q + 2q + 1$). Given a schedule, the processing interval of job j is defined to be the interval (u_j, v_j) such that the processing of j starts at time u_j and ends at time v_j. We say two jobs overlap if they use the same resource and the intersection of their processing interval is nonempty.

A container for a critical job, say, j, is a four-tuple $\vec{v}_j = (i, k_j, a_j, b_j)$ where $1 \leq i \leq m$, $1 \leq k_j \leq c$, $a_j, b_j \in ST$. It implies that job j is processed with resource k_j on machine i during the time window (a_j, b_j) (i.e., its processing interval (u_j, v_j) is a subset of (a_j, b_j)), and furthermore, no other jobs are processed during (a_j, b_j) on machine i.

Obviously there are $mc(\lambda q + 2q + 1)^2$ different kinds of containers. A configuration is then a list of containers for all the critical jobs, namely $(\vec{v}_1, \vec{v}_2, \cdots, \vec{v}_q)$. It can be easily seen that there are at most $m^q c^q (\lambda q + 2q + 1)^{2q}$ different configurations.

A feasible schedule is called a configuration schedule if we can compute a container for each critical job. Notice that this is not always the case since $a_j, b_j \in ST$, and it is possible that any interval (a_j, b_j) during which the critical job j is processed contains some other jobs. Nevertheless, with $O(\epsilon)$-loss we can focus on configuration schedules, as is implied by the following lemma.

▶ **Lemma 4.** *Given a feasible schedule of makespan t, there exists a feasible configuration schedule with makespan no more than $t + 2T\epsilon$.*

Linear Programming for non-critical jobs. Lemma 4 ensures the existence of a configuration schedule whose makespan is bounded by $OPT + 2T\epsilon$. Thus for any $t \in [T/2, T]$, if $t \geq OPT$ then there exists a configuration schedule whose makespan is bounded by $t + 2T\epsilon$.

Recall that there are $\eta \leq m^q c^q (\lambda q + 2q + 1)^{2q}$ different configurations. Let them be CF_1, CF_2, $\cdots$, CF_η. For each configuration, say, CF_κ, the scheduling of critical jobs are fixed. In the following we set up a linear programming $LP_m(CF_\kappa)$ for the remaining jobs.

Suppose according to CF_κ there are $\zeta \leq 2q$ different container points (i.e., the time point when a container starts or ends). We sort them in increasing order as $t_1 < t_2 < \cdots < t_\zeta$. We plug in $t_{\zeta+1} = t + 2T\epsilon$ and $t_0 = 0$.

During each interval (t_i, t_{i+1}) $(0 \leq i \leq \zeta)$, if there is a critical job being processed on a machine, then this machine is called occupied. Otherwise we call it a free machine. Let M_i be the set of free machines during (t_i, t_{i+1}). Similarly during each interval (t_i, t_{i+1}), each resource is either used by a critical job or is not used. Let R_i be the set of resources that are not used during (t_i, t_{i+1}).

Recall that we sort jobs such that $\bar{p}_1 \geq \bar{p}_2 \geq \cdots \geq \bar{p}_n$, and the remaining non-critical jobs are job $q+1$ to job n. We set up a linear programming $LP_m(CF_\kappa)$ as follows.

$$\sum_{i=0}^{\zeta} \sum_{k \in R_i} x_{ijk} = 1, \qquad q+1 \leq j \leq n \tag{2a}$$

$$\sum_{j=q+1}^{n} \sum_{k \in R_i} p_{jk} x_{ijk} \leq (t_{i+1} - t_i)|M_i|, \qquad 0 \leq i \leq \zeta \tag{2b}$$

$$\sum_{j=q+1}^{n} p_{jk} x_{ijk} \leq t_{i+1} - t_i, \qquad 0 \leq i \leq \zeta, k \in R_i \tag{2c}$$

$$x_{ijk} \geq 0, \qquad 0 \leq i \leq \zeta, q+1 \leq j \leq n, k \in R_i \tag{2d}$$

Here x_{ijk} denotes the fraction of job j processed during (t_i, t_{i+1}) with resource k. Constraint (2a) ensures that each non-critical job is scheduled. Since during time interval (t_i, t_{i+1}), only $|M_i|$ machines are free, thus the total load (processing time) of non-critical jobs should not exceed $(t_{i+1} - t_i)|M_i|$, which is implied by (2b). Furthermore, during this interval, the total load of non-critical jobs using any resource $k \in R_i$ is no greater than $t_{i+1} - t_i$ (otherwise the disjoint condition is violated), as is implied by (2c).

As long as $t \geq OPT$, among all the configurations there exists some CF_κ such that $LP_m(CF_\kappa)$ admits a feasible solution. If there is no such a configuration, then we conclude that $t < OPT$. Otherwise, we show a feasible schedule with makespan $t + 3t\epsilon$ could be generated.

▶ **Lemma 5.** *Let x be an extreme point solution of $LP_m(CF_\kappa)$ for some κ, then we have $|\{j | 0 < x_{ijk} < 1 \text{ for some } i, k\}| \leq (m+1)(2q+1)$, i.e., at most $(m+1)(2q+1)$ jobs are split.*

Proof. Suppose there are $\psi \geq n - q$ non-zero variables in the extreme point solution, then they correspond to exact ψ tight constraints among constraints (1), (2) and (3).

Notice that constraints (1) and (2) are composed of $n - q$ and $\zeta + 1$ different inequalities respectively, while constraint (3) is made up of $(\zeta + 1)c$ inequalities. We show that, among the ψ equalities (tight constraints), at most $m(\zeta + 1)$ ones could be from (3). To see why, consider each $0 \leq i \leq \zeta$. For any i, there are at most $|M_i| \leq m$ equalities from (3) since otherwise, the constraint $\sum_{j=q+1}^{n} \sum_{k \in R_i} p_{jk} x_{ijk} \leq (t_{i+1} - t_i)|M_i|$ is violated. Thus $\psi \leq n - q + (m+1)(\zeta+1) \leq n - q + (m+1)(2q+1)$.

Now using a similar argument as [11], we denote μ as the number of jobs getting split (i.e., $x_{ijk} \in (0,1)$ for some i and k), then $2\mu + n - q - \mu \leq \psi \leq n - q + (m+1)(2q+1)$, which completes the proof. ◀

Based on the solution satisfying the above lemma, we show how to generate a near optimal feasible schedule.

First, all the critical jobs are fixed according to CF_κ and we do not need to consider them. Let D be the set of (at most $(m+1)(2q+1)$) split jobs. Temporarily we do not consider them. For each of the remaining non-critical jobs, say, j, there exist some i and k such that $x_{ijk} = 1$, implying that job j should be scheduled during (t_i, t_{i+1}) with resource k. Let U_i be the set of all non-critical jobs (excluding jobs in D) to be scheduled during (t_i, t_{i+1}).

Now we aim to schedule jobs of U_i onto $|M_i|$ free machines during (t_i, t_{i+1}). A *preemptive* schedule satisfying the disjoint condition could be constructed as follows: We order jobs in U_i such that jobs using the same resource are adjacent. We pick a free machine of M_i and put jobs one by one onto it according to the job sequence until their total processing time exceeds $t_{i+1} - t_i$. Then the last job is split and on the next machine we start with its remaining fraction, followed by next jobs in the sequence.

Notice that Constraints (2b) and (2c) ensure that any job of U_i has a processing time no more than $t_{i+1} - t_i$, and their total processing time is no more than $|M_i|(t_{i+1} - t_i)$, thus the above method returns a preemptive schedule where at most $|M_i| \leq m$ jobs are split. Furthermore, the disjoint condition is satisfied. To see why, consider any resource k. All jobs using this resource are adjacent in the job sequence and their total processing time is no more than $t_{i+1} - t_i$, hence they are scheduled either on one machine or on two machines. If they are on one machine then certainly there is no overlap, otherwise on one machine they are started from t_i and on the other machine they are finished until t_{i+1}, and if there is overlap then their total processing time becomes strictly larger than $t_{i+1} - t_i$, which is a contradiction.

Carrying out the above procedure for each (t_i, t_{i+1}), we derive a preemptive schedule in which at most $m(2q+1)$ jobs get split. We take them out and add them to D. Now it can be easily seen that except for jobs in D, all the other jobs are scheduled integrally during $(0, t + 2T\epsilon)$ and the disjoint condition is satisfied.

There are at most $(2m+1)(2q+1)$ jobs in D. Consider the sum of their critical processing times. It remains to show that, there exists a constant q (depending on m and $1/\epsilon$), such that this value is bounded by $T\epsilon$. If this claim holds, then we simply put jobs in D on machine 1 during interval $(t + 2T\epsilon, t + 3T\epsilon)$ and let each job be processed with its critical resource. A feasible schedule with makespan no more than $t + 3T\epsilon$ is derived.

The following lemma from [7] ensures the existence of such a q.

▶ **Lemma 6** ([7]). *Suppose $d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$ is a sequence of real numbers and $D = \sum_{j=1}^{n} d_j$. Let u, v be nonnegative integers, $\alpha > 0$, and assume that n is sufficiently large (i.e., $n > (\lceil \frac{1}{\alpha} \rceil u + 1)(v+1)^{\lceil \frac{1}{\alpha} \rceil}$ suffices). Then, there exists an integer $q = q(u, v, \alpha)$ such that*

$$d_q + d_{q+1} + \cdots + d_{q+u+vq-1} \leq \alpha D,$$

$$q \leq (v+1)^{\lceil \frac{1}{\alpha} \rceil - 1} + u[1 + (v+1) + \cdots + (v+1)^{\lceil \frac{1}{\alpha} \rceil - 2}].$$

In our problem, $\frac{\sum_{j=1}^{n} \bar{p}_j}{m} \leq OPT \leq T$, thus we choose $\alpha = \frac{\epsilon}{m}$, $u = 2m+1$ and $v = 4m+2$, and derive that $q \leq (6m+3)(4m+2)^{\lceil \frac{m}{\epsilon} \rceil}$, which is a constant. Thus we have the following.

▶ **Theorem 7.** *There exists a PTAS for the scheduling with speed-up resources problem when m is a constant.*

4 The special case with a constant number of resources

In this section we assume that each job could be processed with or without a resource. We show that the problem when c is a constant admits a PTAS. The following lemma, which characterize the relationship between the two parameters m and c, is the key to the algorithm.

▶ **Lemma 8.** *Given any positive integer $\lambda = 1/\epsilon$, if there is a feasible solution with makespan T and $m > 3c\lambda$, then there exists a feasible solution with makespan $T(1+\epsilon)$ and all the jobs processed with resources are distributed only on $3c\lambda$ machines.*

Proof idea. The proof is constructive. We only give the main idea here and the reader may refer to the full version of this paper for details. We start with the feasible solution of makespan T and modify it iteratively into the solution satisfying the lemma. During the modification, we only move jobs and do not change the resource each job uses. For simplicity, given a solution, a job processed with resource is called a resource job, and otherwise it is called a non-resource job.

We postpone all jobs by $T\epsilon$ and then divide the time horizon $[0, T(1+\epsilon)]$ equally into $\lambda + 1 = 1/\epsilon + 1$ sub-intervals, each of length $T\epsilon$. Consider each time point $T\epsilon\eta$ for $1 \le \eta \le \lambda$. On each machine, if there is any resource job whose processing interval contains one of these time points, this machine becomes a good machine. It is not difficult to see there are at most $2c\lambda$ good machines. We consider the remaining bad machines. We additionally select $c\lambda$ machines out of them and move all resource jobs of bad machines onto them. This procedure is carried out iteratively. For $1 \le \eta \le \lambda$, suppose we have modified the solution so that the following is true: Among all bad machines, there exist $c(\eta - 1)$ special machines (called as semi-good machines) such that if the processing of a resource job is finished earlier or at the time $T\epsilon\eta$, then it is either on a good machine or on a semi-good machine. Notice that when $\eta = 1$ this condition is trivially true since we postpone all jobs by $T\epsilon$ and none of them could finish before $T\epsilon$. In step η, we try to additionally select c machines out of the remaining machines (not good or semi-good) and try to move onto them all resource jobs scheduled within $(T\epsilon\eta, T\epsilon(\eta+1))$. Assume for simplicity that there is no job crossing time points $T\epsilon\eta$ and $T\epsilon(\eta+1)$ on the c machines we have selected. The crucial observation is that these c additional machines are neither good nor semi-good, hence *no* resource jobs are scheduled on them *before* $T\epsilon\eta$. Given that we have postponed all jobs by $T\epsilon$, on these c additional machines we could shift back by $T\epsilon$ all the non-resource jobs before $T\epsilon(\eta+1)$, whereas enforcing that during $(T\epsilon\eta, T\epsilon(\eta+1))$ only resource jobs are left on these c machines. Now we could simply take out all the resource jobs scheduled within $(T\epsilon\eta, T\epsilon(\eta+1))$, and let all jobs using the same resource be scheduled on one of the c machines. By doing so the disjoint condition is respected and by adding these additional c machines to semi-good machines, we can continue the above procedure for $\eta + 1$. ◀

Let $p_{j0} = p_j$, τ be some constant to be fixed later and $\Lambda = 3c\tau\lambda(\lambda+1)$. Again $\bar{p}_j = \min\{p_{j0}, p_{j1}, \cdots, p_{jc}\}$ is called the critical processing time of job j. We sort all jobs in non-increasing order of their critical processing times. Let T be some integer such that $T/2 \le OPT \le T$. A job is called big if $\bar{p}_j > T\epsilon/\tau$, and small otherwise. With $O(\epsilon)$-loss we could round (down) the processing times of big jobs such that p_{jk} is a multiple of $T\epsilon/\Lambda$ (if $p_{jk} > T$ we simply round it to ∞). It is easy to verify that there are $\phi \le (\lambda\Lambda)^{c+1}$ different kinds of big jobs.

According to Lemma 8, with additional $O(\epsilon)$-loss we may assume that all jobs processed with resources are on the first $3c\lambda$ machines. We call them critical machines and others non-critical machines. With additional $O(\epsilon)$-loss we could further assume that every (rounded)

big job j on critical machines has starting and ending times multiples of $T\epsilon/\Lambda$. Let Sol be the solution of makespan $OPT + T \cdot O(\epsilon)$ satisfying all above requirements (the reader may refer to the full version of this paper for a formal proof). In the following we give an algorithm such that given $t \in [T/2, T]$, it either returns a feasible solution of makespan $t + O(T\epsilon)$, or concludes there is no feasible solution of makespan no more than t.

Consider non-critical machines in Sol. We first classify jobs into groups according to p_{j0}. Let $G_l = \{j | (l-1)T\epsilon^2 < p_{j0} \leq lT\epsilon^2, 1 \leq j \leq n\}$ for $\lambda + 1 \leq l \leq \lambda^2$ and $G_\lambda = \{j | p_{j0} \leq T\epsilon\}$. Notice that now we do not round the processing times but only classify jobs into groups. Similar as the traditional parallel machine scheduling problem [1], we use a $(\lambda^2 - \lambda + 2)$-tuple $(\nu_\lambda, \nu_{\lambda+1}, \cdots, \nu_{\lambda^2})$ to represent the jobs scheduled on a non-critical machine. Here ν_l $(\lambda + 1 \leq l \leq \lambda^2)$ is the number of jobs from G_l on this machine. Furthermore, ν_λ is computed in the following way: we first compute the total processing time of jobs from G_λ and let it be ξ, then $\nu_\lambda = \lfloor \frac{\xi}{T\epsilon} \rfloor$. It is easy to verify that there are at most $\lambda^{O(\lambda^2)}$ different kinds of tuples. We list all the tuples as $(\nu_\lambda(i), \cdots, \nu_{\lambda^2}(i))$ for $1 \leq i \leq \gamma = \lambda^{O(\lambda^2)}$. We say a non-critical machine is of type i, if the jobs on it correspond to the tuple $(\nu_\lambda(i), \cdots, \nu_{\lambda^2}(i))$.

Now we define an outline of a feasible schedule. It indicates which big jobs are scheduled on critical machines. Indeed, given a schedule, an outline for it is a ϕ-tuple $\omega = (\omega_1, \omega_2, \cdots, \omega_\phi)$, where ω_i the number of the i-th kind of big jobs that are scheduled on critical machines. Recall that there are at most Λ big jobs on critical machines, there are $(\Lambda + 1)^\phi$ different possible outlines and we could guess out the outline for Sol. Let the outline be O_ι. Let J_b be the set of all big jobs and CR be the set of big jobs on critical machines according to O_ι.

Similar as we did in Section 3, we define a container (i, k_j, a_j, b_j) for a big job j on critical machines, where k_j is its resource, and a_j, b_j are the starting and ending times, which is a multiples of $T\epsilon/\Lambda$. We also define configurations in a similar way. Let $q' \leq |CR|$ be some constant to be determined later. We take out q' jobs in CR with the largest critical processing times and let $W \subset CR$ be the set of them. A configuration is a list of q' containers for the q' jobs in W. Simple calculations show that there are $(\lambda\Lambda)^{O(q')}$ different configurations.

Suppose we guess the correct outline O_ι and configuration CF_κ. According to the configuration, we sort all different container points as $t_1 < t_2 < \cdots t_\zeta$ with $\zeta \leq 2q'$. Again we plug in $t_0 = 0$ and $t_{\zeta+1} = t$ and set up a mixed integer linear programming $MILP(O_\iota, CF_\kappa)$.

$$\sum_{i=0}^{\zeta} \sum_{k \in R_i} x_{ijk} = 1, \quad j \in CR \setminus W \tag{3a}$$

$$x_{j0} = 1, \quad j \in J_b \setminus CR \tag{3b}$$

$$x_{j0} + \sum_{i=0}^{\zeta} \sum_{k \in R_i} x_{ijk} = 1, \quad j \notin W \tag{3c}$$

$$\sum_{j \notin W} \sum_{k \in R_i} p_{jk} x_{ijk} \leq (t_{i+1} - t_i)|M_i|, \quad 0 \leq i \leq \zeta \tag{3d}$$

$$\sum_{j \notin W} p_{jk} x_{ijk} \leq t_{i+1} - t_i, \quad 0 \leq i \leq \zeta, k \in R_i \setminus \{0\} \tag{3e}$$

$$z_i = 0 \quad \text{if} \quad \sum_{l=\lambda}^{\lambda^2} \nu_l(i)(l-1)T\epsilon^2 \geq t \tag{3f}$$

$$\sum_{i=1}^{\gamma} z_i = m - 3c\lambda \tag{3g}$$

$$\sum_{j\in G_l} x_{j0} = \sum_{i=1}^{\gamma} z_i \nu_l(i), \qquad \lambda+1 \le l \le \lambda^2 \tag{3h}$$

$$\sum_{j\in G_0} x_{j0} p_{j0} \le \sum_{i=1}^{\gamma} z_i \nu_l(i) l T\epsilon, \qquad \lambda \le l \le \lambda^2 \tag{3i}$$

$$x_{ijk} \ge 0, x_{j0} \ge 0 \qquad 0 \le i \le \zeta, j \notin W, k \in R_i \tag{3j}$$

$$z_i \ge 0, z_i \in \mathrm{Z}, \qquad 1 \le i \le \gamma \tag{3k}$$

Here we use similar notations as that of Section 3. Note that the positions of jobs in W are already fixed by CF_κ and we do not need to consider them. R_i is the set of resources that are not used by jobs of W during (t_i, t_{i+1}). Specifically, 0 is taken as a special resource such that if job j is processed without any resource, then it is taken as processed with resource 0. Thus resource 0 is always available and $0 \in R_i$ for any $0 \le i \le \zeta$. M_i is the set of critical machines that are not occupied by jobs of W during (t_i, t_{i+1}) and again we call them as free machines.

We explain the variables used. x_{ijk} is the fraction of job j scheduled during (t_i, t_{i+1}) with resource k. Since during this interval only resources of R_i are available, thus it is only defined for $k \in R_i$. Furthermore, x_{ij0} denotes the fraction of job j scheduled without any resource and as we mention before, it is viewed as processed with resource 0. x_{j0} is the fraction of job j scheduled on non-critical machines. z_i is the number of non-critical machines of type i.

We explain the constraints. Notice that a big job (of J_b) is either on critical machines or on non-critical machines, and this is determined beforehand by O_ι. For $j \in CR$, it should be on critical machines and there are two cases. One is that $j \in W$, then the position of this job is further determined through CF_κ and we do not need to consider it. The other case is $j \in CR \setminus W$, then it should be on critical machines, just as (3a) implies. For big jobs that are not on critical machines, they are on non-critical machines, which is implied by (3b). Constraint (3c) implies that each job should be scheduled either on critical machines or on non-critical machines, and this holds for both big and small jobs.

Constraints (3d) and (3e) are the same with the constraints in LP_m we derive in Section 3. (3d) means the total processing time of jobs scheduled during (t_i, t_{i+1}) on critical machines should not exceed the available times provided by free machines. This is straightforward since the other $3c\lambda - |M_i|$ critical machines are occupied by jobs of W and we can not put jobs on it. (3e) means the total processing time of jobs using resource $k \in R_i$ during (t_i, t_{i+1}) should not exceed $t_{i+1} - t_i$. Notice that 0 should be excluded since it is not a real resource, i.e., jobs processed without resource could be processed at the same time if they are on different machines.

Constraints (3f),(3g),(3h),(3i) are standard constraints. (3f) excludes tuples that are infeasible. (3g) holds as each non-critical machine is of a certain type. Both sides of (3h) equal to the number of jobs in G_l that are scheduled on non-critical machines. Notice that here G_λ is not taken account of since such jobs can be split, just as in the classical scheduling problem. The left side of (3i) calculate the total processing time of jobs in G_l on non-critical machines and the right side is obviously its upper bound.

It can be easily seen that the in the above $MILP$ there is only a constant number of integer variables which is bounded by $\gamma = (\lambda+4)^{\lambda^2-\lambda+1}$, i.e., $2^{O(1/\epsilon^2 \log(1/\epsilon))}$, thus it could be solved in $f(1/\epsilon)poly(n, \log P)$ time using Kannan's algorithm [8]. Here $P = \sum_{j=1}^{n} \bar{p}_j$ is a natural upper bounded for T and $f(1/\epsilon)$ only depends on $1/\epsilon$. Given a feasible solution of the $MILP(O_\iota, CF_\kappa)$, we can show that it could be rounded into an integer solution with an additive loss of $T\epsilon \cdot O(\lambda^2 + cq')$. This is again by observing that once we fix the value of

integer variables z_i, there are only a limited number of constraints for the fractional variable x_{ijk}, whereas we get at most $O(\lambda^2 + cq')$ split (small) jobs. Choosing proper q' and τ allows us to bound the overall increase by $O(\epsilon)OPT$. The reader is referred to the full version of this paper for details.

▶ **Theorem 9.** *There is a PTAS for the scheduling with speed-up resources problem when c is a constant.*

Acknowledgements. We thank Janer Chen for pointing out the relationship between the problem we consider and $P|set|C_{max}$ and other useful communications.

References

1 N. Alon, Y. Azar, G.J. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, pages 493–500, 1997. doi:10.1109/SFCS.1975.23.

2 J. Chen and A. Miranda. A polynomial time approximation scheme for general multiprocessor job scheduling. *SIAM journal on computing*, 31(1):1–17, 2001. doi:10.1145/361604.361612.

3 A. Grigoriev, M. Sviridenko, and M. Uetz. Machine scheduling with resource dependent processing times. *Mathematical programming*, 110(1):209–228, 2007. doi:10.1145/361604.361612.

4 K. Jansen, M. Maack, and M. Rau. Approximation schemes for machine scheduling with resource (in-)dependent processing times. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1526–1542, 2016. doi:10.1109/SFCS.1975.23.

5 K. Jansen and M. Mastrolilli. Scheduling unrelated parallelmachines: linear programming strikes back. Technical report, University of Kiel, 2010. Technical Report Bericht-Nr. 1004. doi:10.1109/SFCS.1975.23.

6 K. Jansen and L. Porkolab. General multiprocessor task scheduling: Approximate solutions in linear time. In *Workshop on Algorithms and Data Structures (WADS'99)*, pages 110–121, 1999. doi:10.1109/SFCS.1975.23.

7 K. Jansen and L. Porkolab. Improved Approximation Schemes for Scheduling Unrelated Parallel Machines. *Mathematics of Operations Research*, 26(2):324–338, 2001. doi:10.1145/361604.361612.

8 R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12:415–440, 1987. doi:10.1145/361604.361612.

9 H. Kellerer. An approximation algorithm for identical parallel machine scheduling with resource dependent processing times. *Operations Research Letters*, 36(2):157–159, 2008. doi:10.1145/361604.361612.

10 H. Kellerer and V.A. Strusevich. Scheduling parallel dedicated machines with the speeding-up resource. *Naval Research Logistics*, 55(5):377–389, 2008. doi:10.1145/361604.361612.

11 J. K. Lenstra, D. B. Shmoys, and Eva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programing*, 46:259–271, 1990. doi:10.1145/361604.361612.

12 H. Xu, L. Chen, D. Ye, and G. Zhang. Scheduling on two identical machines with a speed-up resource. *Information Processing Letters*, 111(7):831–835, 2011. doi:10.1145/361604.361612.