Minimizing Makespan and Preemption Costs on a System of Uniform Machines

Hadas Shachnai[1,⋆], Tami Tamir[2], and Gerhard J. Woeginger[3]

[1] Bell Laboratories, Lucent Technologies, 600 Mountain Ave. Murray Hill, NJ 07974
hadas@research.bell-labs.com
[2] Department of Computer Science, Technion, Haifa 32000, Israel
tami@cs.technion.ac.il
[3] Faculty of Mathematical Sciences, Univ. of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands. woeginge@math.utwente.nl

Abstract. It is well known that for preemptive scheduling on uniform machines there exist polynomial time exact algorithms, whereas for non-preemptive scheduling there are probably no such algorithms. However, it is not clear how many preemptions (in total, or per job) suffice in order to guarantee an optimal polynomial time algorithm. In this paper we investigate exactly this hardness gap, formalized as two variants of the classic preemptive scheduling problem. In *generalized multiprocessor scheduling (GMS)*, we have *job-wise* or *total* bound on the number of preemptions throughout a feasible schedule. We need to find a schedule that satisfies the preemption constraints, such that the maximum job completion time is minimized. In *minimum preemptions scheduling (MPS)*, the only feasible schedules are preemptive schedules with smallest possible makespan. The goal is to find a feasible schedule that minimizes the overall number of preemptions. Both problems are NP-hard, even for two machines and zero preemptions.

For GMS, we develop *polynomial time approximation schemes*, distinguishing between the cases where the number of machines is *fixed*, or given as part of the input. For MPS, we derive matching lower and upper bounds on the number of preemptions required by *any* optimal schedule.

1 Introduction

The problem of preemptive scheduling on uniform machines so as to minimize the overall completion time (or *makespan*), is well-known to be solvable in polynomial time. However, for some instances, any optimal schedule requires $\Omega(m)$ preemptions, where m is the number of machines [7]. While in traditional multiprocessor scheduling the cost of preemptions is relatively small, in the modern distributed computing environment, preemptions typically involve communication, and sometimes require job migration over a network. This can significantly increase the cost of the schedule. Therefore it is natural to seek schedules that minimize the overall completion time, while incurring a small number of preemptions. We consider the resulting variants of the classic preemptive scheduling

⋆ On leave from the Department of Computer Science, Technion, Haifa 32000, Israel.

problem. Formally, suppose we are given a set of jobs, $J_1, \ldots, J_n$, with processing times $t_1, \ldots, t_n$, and m uniform machines, $M_1, \ldots, M_m$; the machine M_i has the rate $u_i \geq 1$.

> In the *generalized multiprocessor scheduling (GMS)* problem, each job J_j has an associated parameter a_j, which bounds the number of times J_j can be preempted throughout a feasible schedule. We need to find a schedule that satisfies the preemption constraints, such that the maximum job completion time is minimized.[4]
>
> In the *minimum preemptions scheduling (MPS)* problem, the only feasible schedules are preemptive schedule with smallest possible makespan. The goal is to find a feasible schedule that minimizes the overall number of preemptions.

A straightforward reduction from the *Partition* problem [5] shows that all these problems are NP-hard, even for two machines and zero preemptions. The two classical problems of preemptive and non-preemptive scheduling on uniform machines, which are special cases of GMS, were extensively studied. It is well known that for preemptive scheduling there exist polynomial time exact algorithms, whereas for non-preemptive scheduling there are probably no such algorithms. However, it is not clear how many preemptions (in total, or per job) suffice in order to guarantee an optimal polynomial time algorithm. In this paper we investigate exactly this hardness gap. We give proofs of hardness for some special cases of GMS, and we develop *polynomial time approximation schemes (PTAS)* for this problem.

Our results for generalized multiprocessor scheduling yield important distinction between the identical and uniform machine environments. For the two fundamental scheduling problems that we generalize here, similar solvability/approximability results were obtained in these two environments. In particular, the preemptive scheduling problem is optimally solvable on both identical and uniform machines, and the non-preemptive scheduling problem is strongly NP-hard in both. Yet, the two environments differ already when we allow each of the jobs to be preempted *at most once*. While on identical machines we can use McNaughton's rule [12] to obtain the minimum makespan, on uniform machines the problem is strongly NP-hard (see in Section 2.2).

1.1 Our Results

We now describe our main results for GMS and MPS. We give (in Section 2) proofs of hardness for GMS in the following special cases: (i) each of the jobs can be preempted at most *once* throughout the schedule, and (ii) the overall number of preemptions is bounded by k, for some $1 \leq k < 2(m-1)$.[5] We then develop (in Section 3.1) a PTAS for any instance of GMS in which the number of machines is *fixed*. Our scheme has *linear* running time, and can be applied also for instances where jobs have release dates, for GMS on *unrelated* machines, and

[4] Alternatively, we are given a bound *Tot* on the total number of preemptions.

[5] As shown in [7], for $k \geq 2(m-1)$ the problem is polynomially solvable.

for instances with *arbitrary* preemption costs. In Section 3.2, we give PTASs for instances of GMS with arbitrary number of machines and a bound on the total number of preemptions.

For MPS, we derive (in Section 4) matching lower and upper bounds on the number of preemptions required by *any* optimal schedule. Our results hold for any instance in which a job J_j, $1 \leq j \leq n$, can be processed simultaneously by ρ_j machines, for some $1 \leq \rho_j \leq m$. In particular, we show that a lower bound for the overall number of preemptions, in a schedule that yields the minimum makespan, is $m + \lfloor m/b \rfloor - 2$, where $b = \min_{1 \leq j \leq n} \rho_j$. We give a polynomial time algorithm that achieves this bound. For the special case where $\rho_j = 1\ \forall j$, our algorithm uses $2(m-1)$ preemptions, as the algorithm presented in [7]; however, our algorithm and its analysis are simpler.

Our main technical contribution is the extension of the approximation technique introduced in [16] for open shop scheduling (and later extended in [11] to other *shop* scheduling problems), to obtain approximation schemes for generalized multiprocessor scheduling, where the number of machine is fixed. We exemplify this in our scheme for GMS on uniform machines. However, the technique can be applied for other variants of the problem, as discussed in Section 3.

1.2 Related Work

The GMS problem generalizes the two classical problems of scheduling on uniform machines to minimize the makespan. When $\forall j, a_j$ is unbounded, we get the preemptive scheduling problem, denoted in standard scheduling notation $Q \mid pmtn \mid C_{max}$. Horvath et al. [10] gave the first optimal algorithm for this problem. When $a_j = 0\ \forall j$, we get the non-preemptive scheduling problem, $Q \mid\mid C_{max}$, which is strongly NP-hard. For this problem, it was shown in [6] that algorithm *longest processing time (LPT)* yields a ratio of 2 to the optimal makespan. In Section 2.3 we extend this result for the GMS and MPS problems. Hochbaum and Shmoys [8, 9], and later Epstein and Sgall [4], developed PTASs for $Q \mid\mid C_{max}$. However, these schemes cannot be adapted for the GMS problem, since they assume no dependencies among the scheduled jobs. When we allow preemptions in the schedule, each job becomes a set of segments, only one of which can be processed at any point of time.

The MPS problem was studied in [7], in the case where each job can be processed at any time by a *single* machine. The paper shows that there are instances for which any optimal algorithm uses at least $2(m-1)$ preemptions. The paper presents an algorithm that achieves this bound. Other previous works on *parallel* jobs (see, e.g., [1, 3]) assume an *even* partition of the processing of a job J_j among machines that run J_j in parallel, while we do not use this assumption (see in Section 2.1).

Due to space constraints we state some of the results without proofs.[6]

[6] The detailed proofs are given in [14].

2 Preliminaries

2.1 Minimum Makespan on Uniform Machines

Let I be an instance of MPS, where job J_j has processing time t_j for $1 \leq j \leq n$. J_j can be processed simultaneously on ρ_j machines; the processing of J_j can be shared arbitrarily among the machines. We partition J_j to ρ_j jobs, $J_{j,1}, \ldots, J_{j,\rho_j}$; the processing time of $J_{j,l}$ is t_j/ρ_j. The resulting instance is I'. In the following, we use the instance I' to define the makespan of any optimal schedule of I.

Assume that the jobs in I' are sorted such that $\frac{t_1}{\rho_1} \geq \frac{t_2}{\rho_2} \geq \ldots \geq \frac{t_n}{\rho_n}$, and the machines are sorted such that $u_1 \geq u_2 \geq \ldots \geq u_m$. Let

$$w = \max\{\frac{T_n}{U_m}, \max_{1 \leq j \leq m} \frac{T'_j}{U_j}\}, \tag{1}$$

where $U_j = \sum_{i=1}^{j} u_i$, T_n is the total length of the jobs in I, and T'_j is the total length of the first j jobs in I'. We omit the proofs of the next results.

Lemma 1. *The makespan of any optimal schedule of I is $w_{OPT}(I) = w$.*

Lemma 2. *Let $\mathcal{A}_{opt}$ be an algorithm that solves the preemptive scheduling problem for I', in $f(n)$ steps. Then $\mathcal{A}_{opt}$ can be adapted to yield the optimal makespan for I, in $f(\sum_{j=1}^{n} \min(\rho_j, m))$ steps.*

2.2 Hardness Results

Consider the *single preemption* scheduling problem, where each of the jobs can be preempted at most *once* throughout the schedule.

Theorem 1. *The single preemption scheduling problem is strongly NP-hard.*

When the overall number of preemptions is bounded by k, for some $1 \leq k \leq 2(m-1)$, we draw the line between hard and easy instances. Specifically, an algorithm that achieves the optimal makespan and uses at most $2(m-1)$ preemptions was given in [7].

Theorem 2. *For any $k < 2(m-1)$, the problem of finding a minimum makespan schedule with at most k preemptions is strongly NP-hard.*

The proofs of Theorems 1 and 2 are given in [14].

2.3 LPT and the Power of Unlimited Preemptions

Consider the LPT algorithm, that assigns jobs to machines in order of non-increasing processing times. Each job is assigned to the machine that will complete its execution first. When $\rho_j > 1$ for some $1 \leq jleqn$, we apply this rule to the instance I' (introduced in Section 2.1). Clearly, the resulting schedule is non-preemptive and the parallelism constraints are preserved. In [6], it was shown that LPT yields a ratio of 2 to the optimal *non-preemptive* makespan. A closer analysis of LPT implies the following. Let w denote the optimal preemptive schedule (as defined in Equation (1)).

Theorem 3. *Any LPT schedule yields a 2-approximation to w.*

Thus, LPT is a non-preemptive schedule that achieves 2-approximation for MPS and GMS. It can be shown that this bound is tight (see in [14]).

3 Approximation Schemes for GMS

3.1 Fixed Number of Machines

In this section we describe a polynomial time approximation scheme for GMS, where m is a fixed constant. Note that in this case we may assume that a_j, the maximal number of preemptions of J_j, is fixed and bounded by $2(m-1)$, for all $1 \leq j \leq n$.

We combine in our scheme the technique of [16] for open shop scheduling, with the *configuration graph* approach used in [9, 4], for scheduling on uniform machines. As in [16], we partition the jobs to subsets, by their processing times. The *big* jobs are handled first. The scheme finds a feasible *preemptive* schedule of these jobs, which is within factor $(1+\varepsilon)$ from the optimal. The *small* jobs have non-negligible processing times, yet, the number of possible schedules for these jobs may be exponentially large. Thus, we select this set so that the *overall* processing time of the jobs is small relative to an optimal solution for the problem. This allows us to schedule these jobs *non-preemptively* on the machines (although some or all of these jobs must be preempted, in any optimal solution). Finally, we use a simple algorithm for adding to the schedule the *tiny* jobs, which are processed *non-preemptively.* Since their running times are negligible, so is their contribution to the overall completion time of the schedule. Our scheme proceeds in the following steps. For a given $\varepsilon > 0$, let $\delta = \alpha\varepsilon^3/(m(a_{max}+1)^2)$.

1. Guess the minimum makespan, $T_{\mathcal{O}}$, to within factor $1+\varepsilon$.
2. Given $T_{\mathcal{O}}$, guess P, the maximal load (in processing units) on any machine.
3. Partition the jobs by their processing times to *big*, *small* and *tiny*, given by the sets B, S and T, respectively. Let $\alpha \in (0,1]$ be a constant (to be determined), and $a_{max} = \max_{1 \leq j \leq n} a_j$. Then,
$$B = \{J_j \mid t_j > \alpha\varepsilon P\}\ ,\ S = \left\{J_j \mid \frac{\delta\varepsilon}{m}P < t_j \leq \alpha\varepsilon P\right\}\ ,\ T = \left\{J_j \mid t_j \leq \frac{\delta\varepsilon}{m}P\right\}$$
4. For any $J_j \in B$, guess a partition of J_j to at most $a_j + 1$ segments, such that each segment is of length at least $\alpha\varepsilon^2 P/(a_{max}+1)^2$, rounded up to the closest integral multiple of $\delta\varepsilon P/m$.
5. Find an optimal schedule of all the segments of the jobs in B, such that the starting time of each segment is an integral multiple of $\delta T_{\mathcal{O}}$.
6. Schedule the jobs in $S \cup T$ non-preemptively on the machines. First, assign greedily all the jobs in S to *holes* on the machines: for any job $J_j \in S$, we look for the minimal hole in which J_j can be scheduled non-preemptively. The jobs that are scheduled in each hole are processed with non-idle times. If no hole fits for scheduling J_j, then we add J_j at the end of the schedule, on the machine that will complete its execution first. We then proceed to schedule the jobs in T. Starting with the first hole on M_1 (the time interval between the end of the schedule on any machine and $T_{\mathcal{O}}$ is also considered as a 'hole'), we fill each of the holes, by scheduling sequentially jobs in T. If the hole is too short for the next job, we move this job to the end of the schedule on the same machine. We mark that hole as 'full' and move to the

next hole on the same machine. Once all the holes on M_i are marked as 'full', we move to the first hole on M_{i+1}.

In Step 5. we construct a *configuration graph* to find a partial schedule of the segments of big jobs on the first i machines, $1 \leq i \leq m$.[7]

Analysis We use the next lemmas to show that the schedule output by the scheme is of length at most $T_{\mathcal{O}}(1+\varepsilon)$.

Lemma 3. *Setting the starting times of the segments of the jobs in B to be integral multiples of $\delta \varepsilon P/m$. can increase the overall length of the schedule at most by factor $1+\varepsilon$.*

Lemma 4. *Any schedule of length $T_{\mathcal{O}}$ of the segments of jobs in B, can be transformed into one of length at most $(1+\varepsilon)T_{\mathcal{O}}$, in which (i) the minimal length of any segment of J_j is at least $\alpha\varepsilon^2 P/(a_{max}+1)^2$, and (ii) the length of each segment is an integral multiple of $\alpha\varepsilon^4 P/((a_{max}+1)^2 \cdot m^2)$.*

Lemma 5. *There exists $\alpha \in [(\varepsilon^3/(m^2 \cdot (a_{max}+1)^2))^{m-1}, 1]$, such that the set S of small jobs, selected with α, satisfies $\sum_{J_j \in S} t_j \leq \varepsilon P$.*

Theorem 4. *The above scheme yields a $(1+\varepsilon)$-approximation to the minimum makespan, in $O(n + (\frac{1}{\alpha\varepsilon^2})^{1/\alpha\varepsilon^3})$ steps.*

Extensions Our scheme can be applied without change (except for the initial guess for $T_{\mathcal{O}}$) to instances where jobs have release dates, and to GMS on unrelated machines.

When we are given a bound, *Tot*, on the total number of preemptions in the schedule, we can apply the above scheme, except that in Step 4. we guess a partition of the big jobs to at most $|B|+Tot$ segments, such that each segment is of length at least $\alpha\varepsilon^2 P/(a_{max}+1)^2$, rounded up to the closest integral multiple of $\delta\varepsilon P/m$.

Arbitrary preemption costs Consider now the case where the cost of preemption for J_j on the machine M_i is $c_{ij} \geq 1$, for $1 \leq i \leq m$, $1 \leq j \leq n$. Given $C > 1$, we need to find a preemptive schedule in which the makespan is minimized, and the overall preemption cost is at most C. Our scheme can be applied to this case as follows. to this case. For any $1 \leq j \leq n$, let $a_j = 2(m-1)$. (i) Guess $T_{\mathcal{O}}$ and P. (ii) Partition the jobs (as before) to the sets B, S and T. (iii) Guess the maximal total cost of preemptions on any machine, $\hat{C}$. (iv) Partition the big jobs to segments, as before; round the value c_{ij} up to the next integral multiple of $\varepsilon\hat{C}/(m \cdot (a_{max}+1)^2/(\alpha\varepsilon^2))$; partition the jobs to classes, such that all jobs in a class have the same segment configuration, and the same cost of preemption on each machine. (v) Find a schedule that minimizes preemption cost for the segments of all jobs in B, such that the overall completion time is at most $(1+\varepsilon)T_{\mathcal{O}}$. (vi) The jobs in $S \cup T$ are scheduled as before.

In Step (v) we find the *minimal cost* of any final assignment of the job segments to the machines. Note that the last segment of any job does not incur

[7] We give the details in [14].

preemption cost, thus, given a schedule of the segments, we need to find an assignment of the last segments of the jobs, that minimizes the total preemption cost. This can be done in polynomial time, since the number of machines is fixed. We summarize in the next result.

Theorem 5. *For any fixed $m > 1$, there is a polynomial time approximation scheme for GMS, on unrelated machines, with release dates and arbitrary preemption costs.*

3.2 Arbitrary Number of Machines

Suppose that the number of machines is given as part of the input, and we are given a bound, Tot, on the overall number of preemptions.

Identical Machines In any preemptive schedule, there are *preempted* and *non-preempted* jobs. The processing of a preempted job is split into two or more segments. Note that for identical machines McNaughton schedule incurs at most $m-1$ preemptions (of the last job on each of the first $m-1$ machines).

Definition 1. *A preemptive schedule for a GMS instance is called primitive, if it satisfies the following two conditions. (i) Every preempted job is preempted exactly once. These two segments are processed on two different machines, and (ii) Every machine processes at most two segments of preempted jobs. These segments are either processed first or last on the machine.*

Lemma 6. *For every instance of GMS with an arbitrary number of identical machines and a value of $Tot \geq 1$, there exists an optimal schedule that is primitive.*

Next, we sketch a PTAS for approximating the optimal primitive schedule. Let T denote the makespan of the LPT-schedule; hence, T is a 2-approximation of the optimal makespan. Let $\varepsilon > 0$ be the desired precision of approximation. A job is *big*, if $t_j \geq \varepsilon T$; otherwise, the job is *small*. We introduce a *rounded instance* that corresponds to instance I. For every big job J_j, we define a corresponding rounded job whose processing time equals to t_j rounded down to the next integer multiple of $\varepsilon^2 T$. Rounded big jobs can only be preempted once to form two segments, whose lengths are integer multiples of $\varepsilon^2 T$ and $\geq \varepsilon T$. Let P_s denote the total processing time of all small jobs; then the rounded instance contains a number of small jobs of length εT, such that their total size equals P_s rounded down to the next integer multiple of $\varepsilon^2 T$. These small jobs must not be preempted at all.

An optimal primitive schedule for the rounded instance can be computed in polynomial time by dynamic programming or via integer programming in fixed dimension. Moreover, an optimal primitive schedule for the rounded instance can be translated into a primitive schedule for the original instance that is a $(1+10\varepsilon)$-approximation of the optimal primitive schedule for the original instance.

Uniform Machines Assume now that the machines are uniform, and $1 \leq Tot < 2(m-1)$ is some constant; then, initially, we guess the optimal completion time, $T_{\mathcal{O}}$. Next, we guess the set of machines, $\mathcal{M}_p$, on which some jobs will be preempted. Let $m' = |\mathcal{M}_p|$ denote the size of this set; then, $1 \leq m' \leq Tot$.

The jobs scheduled on the remaining machines, the set $\mathcal{M}_{np}$, will run non-preemptively. These jobs can be identified efficiently and scheduled to yield a makespen of at most $T_{\mathcal{O}}(1+\varepsilon)$, by using a transformation to the *multiple knapsack* problem (see in [14]). The makespan of the resulting schedule is bounded by $T_{\mathcal{O}}(1+\varepsilon)$.

Finally, we use the scheme in Section 3.1 to obtain a feasible schedule of the jobs on the machines in $\mathcal{M}_p$, whose overall completion time is at most $T_{\mathcal{O}}(1+\varepsilon)$.

4 Minimizing the Number of Preemptions

In this section we consider the MPS problem. For convenience, we refer below to the number of *segments*, $N(I)$, generated for the instance I.[8] Lemma 2 implies that we can find efficiently a schedule that minimizes the makespan for a given instance of MPS. However, the schedules generated by adopting an optimal algorithm for $Q\,|\,pmtn\,|\,C_{max}$, may have an excessive number of job segments. In fact, we may get $\sum_j (\rho_j - 1)$ extra job-segments in the schedule of I.

We derive below a lower bound for $N_s(I)$, for any algorithm that achieves the minimal makespan; then, we present an algorithm, $\mathcal{A}_p$, that achieves this bound.

Theorem 6. *For any m, b, $1 \leq b \leq m$, there exists an instance, I, in which $\rho_j \geq b, \forall j$, and in any optimal schedule of I, $N_s(I) \geq m + n + \lfloor \frac{m}{b} \rfloor - 2$.*

4.1 An Upper Bound on $N_s(I)$

Let w be the length of an optimal schedule for I (By Lemma 1, w can be calculated efficiently using equation (1)). We now describe the algorithm $\mathcal{A}_p$, which achieves the makespan w and also minimizes $N_s(I)$. Assume that the jobs are sorted such that $\frac{t_1}{\rho_1} \geq \ldots \geq \frac{t_n}{\rho_n}$, and the machines are sorted such that $u_1 \leq \ldots \leq u_m$.

As in [7], we define a *disjoint processor system* (DPS) to be a union of disjoint *idle-segments* of r machines with non-decreasing rates, such that the union of the idle-segments is the interval $[0, w]$. Formally, let $M_k^1, \ldots, M_k^r$ be a set of r machines with non-decreasing rates. Then, these r machines form a DPS if M_k^h is idle exactly from τ_k^{h-1} to τ_k^h, where $\tau_k^0 = 0, \tau_k^r = w$ and $\tau_k^{h-1} < \tau_k^h$, for all $1 \leq h \leq r$ (see Figure 1). Note that a DPS is defined relative to a given (partial) schedule.

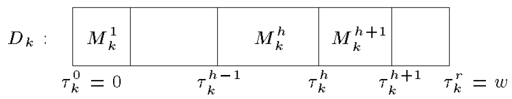

Fig. 1. A DPS D_k

[8] The number of segments of J_j is the number of preemptions incurred plus one.

For each DPS, D_k, let Q_{D_k} denote the *potential* of D_k, that is, the number of processing units that D_k can allocate. Initially, each of the m machines forms a DPS with $r = 1$ and $Q_{D_k} = wu_k$. In general, Q_{D_k} is the weighted-average of the processing potential of the machines that form D_k. Formally, let r_k denote the number of machines that compose D_k, and let u_k^h denote the rate of the machine M_k^h, then $Q_{D_k} = \sum_{h=1}^{r_k} u_k^h(\tau_k^h - \tau_k^{h-1})$. The algorithm $\mathcal{A}_p$ maintains a list, L, of the DPSs, sorted by their potential in non-decreasing order. That is, $Q_{L[1]} \leq Q_{L[2]} \leq \ldots$, where $L[i]$ is the DPS at position i in L. The list L is updated during the execution of the algorithm, according to the current available DPSs. Given a pair of DPSs D_a, D_b, we say that D_a is weaker (stronger) than D_b, if $Q_{D_a} \leq Q_{D_b}$ ($Q_{D_a} \geq Q_{D_b}$).

The jobs, sorted by their processing ratios, are scheduled one after the other. $\mathcal{A}_p$ uses two scheduling procedures:

Greedy-schedule: This procedure schedules a job, J_j, on the machines that form the DPSs, one after the other, starting from the first machine in the weakest DPS, until J_j is allocated exactly t_j processing units.

Moving-window: This procedure schedules a job, J_j, on ρ_j DPSs whose total potential is *exactly* t_j. This set of ρ_j DPSs consists of $\rho_j - 1$ DPSs from L and one DPS that is formed from two DPSs in L. To find this set, we scan the list using a window of size $\rho_j + 1$. Initially, the window covers the set of the weakest $\rho_j + 1$ DPSs (given by $L[1], ..., L[\rho_j + 1]$). In each iteration we replace the weakest DPS in the window by the next DPS in L, until the total potential of the ρ_j strongest DPSs in the window is large enough to complete J_j. Let $D_{k_1}, \ldots, D_{k_{\rho_j+1}}$ be the set of $\rho_j + 1$ DPSs in the window. We first allocate to J_j all the potential of the DPSs $D_{k_2}, \ldots, D_{k_{\rho_j}}$ (if $\rho_j = 1$ this is an empty set), and complete J_j by allocating to it also some of the potential of $D_{k_1}, D_{k_{\rho_j+1}}$. The allocation is done such that the length of the schedule of J_j on these two DPSs is exactly w, and the unused intervals of $D_{k_1}, D_{k_{\rho_j+1}}$ form a new DPS.

Figure 2 gives a schedule on two consecutive DPSs: the shaded intervals are allocated to J_j. The non-used intervals of D_a, D_b form a new DPS, D', with potential Q' which is the weighted-average of the non-used intervals. The DPSs D_a, D_b are removed from L and D' is added to L. Since $Q_{D_b} \leq Q' \leq Q_{D_a}$, D' is positioned in L in place of D_a and D_b. We now give a formal description of the algorithm.

Algorithm $\mathcal{A}_p$: The jobs, sorted by the ratios t_j/ρ_j, are scheduled sequentially. The following rules are used when $\mathcal{A}_p$ schedules a job J_j.

- If there are at most ρ_j DPSs in L, or if the total potential of the ρ_j weakest DPSs in L is at least t_j, schedule sequentially J_j and all the remaining jobs, using the greedy-schedule procedure.
- Otherwise, schedule J_j using the moving-window procedure.

We first show that $\mathcal{A}_p$ generates a legal schedule of length w for I; then, we compute the resulting number of job-segments. We distinguish between two phases of $\mathcal{A}_p$: (i) Jobs are scheduled using the moving-window procedure. (ii) Jobs are scheduled using the greedy-schedule procedure.

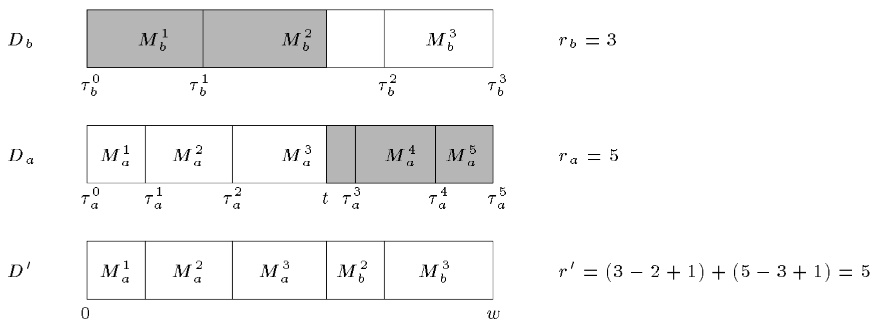

Fig. 2. The schedule of J_j on consecutive DPSs

For simplicity, we analyze $\mathcal{A}_p$ assuming that $\forall j, \rho_j = 1$. In the sequel, we explain how the analysis can be applied for arbitrary ρ_j's. We first show that during the first phase we can always schedule J_j on two consecutive DPSs, such that the idle-segments of these DPSs that remained unused, form a DPS. Next, we show that when we proceed to the greedy-schedule stage, we can schedule all the remaining jobs. Finally, we show that the total number of segments is at most $n+2(m-1)$ (which is optimal by Theorem 6). Recall that the list L of the DPSs is sorted such that $L[1]$ is the weakest DPS. Note that L becomes shorter as the number of available DPSs decreases.

Claim 1. If $Q_{L[1]} < t_j$ then there exist two consecutive DPSs $D_{L[i]}, D_{L[i+1]}$ such that $Q_{L[i]} \leq t_j$ and $Q_{L[i+1]} \geq t_j$.

Claim 2. Given two DPSs, D_a, D_b, from L, such that $Q_{D_a} \leq t_j$ and $Q_{D_b} \geq t_j$, we can always schedule J_j on D_a, D_b such that the length of the schedule of J_j is exactly w.

The next Claim implies that greedy is suitable for the second phase.

Claim 3. If $Q_{L[1]} \geq t_j$, then we can complete the schedule of the remaining jobs greedily.

We now consider the number of job segments generated by $\mathcal{A}_p$.

Lemma 7. *The total number of segments is at most $n+2(m-1)$.*

Proof: For a given schedule, define a *busy-segment* as a maximal time interval in which a machine is processing some job without preemptions. Throughout the algorithm, for each machine, M_i, the w-interval of M_i may consist of busy-segments, and one *idle-segment*, which belongs to some DPS (Note that M_i belongs to at most one DPS). Each idle-segment may be partitioned to many busy-segments by the end of the schedule.

Denote by X_0 the initial number of idle or busy segments. Clearly, initially we have one idle-segment of length w on each machine, and no busy-segments. Thus, $X_0 = m$. Denote by X_j the total number of segments (of both types) after the job J_j was scheduled, and by X_n^B the total number of busy-segments after the last job, J_n, was scheduled. That is, $X_n^B = N_s(I)$.

We show that in any schedule that uses the moving-window procedure, the total number of segments may increase by at most two, and in any greedy-schedule, the total number of segments may increase by at most one. Then, we bound the number of moving-window schedules, in order to bound the maximal possible number of segments by the end of the schedule.

Claim 4. If J_j is scheduled in the first phase then $X_j \le X_{j-1} + 2$.

Claim 5. If $J_j, j < n$, is scheduled in the second phase then $X_j \le X_{j-1} + 1$.

Claim 6. $X_n^B \le X_{n-1}$.

Denote by n_w, n_g the number of jobs scheduled in the first and the second phase respectively. Clearly, $n_g = n - n_w$. Combining Claims 4-6, we get:

$$X_n^B \le X_{n-1} \le X_0 + 2n_w + n_g - 1 = m + 2n_w + n_g - 1 = m + n + n_w - 1. \quad (2)$$

To bound X_n^B, we bound n_w. Note that each job scheduled during the first phase, unites two DPSs. Thus, after we schedule at most $m-1$ jobs, we have a single DPS on which we can schedule greedily the remaining jobs. (In some cases, the greedy phase starts when some job J_j with $j < m$ is scheduled.) Thus, $n_w \le m - 1$. From Equation (2), $X_n^B \le m + n + n_w - 1 \le n + 2(m-1)$. This completes the proof of the lemma. □

The analysis of $\mathcal{A}_p$ for instances with arbitrary ρ_j's is similar. Note that when we schedule a job J_j with $\rho_j > 1$ using the moving-window procedure, J_j runs on $\rho_j + 1$ DPSs $D_{k_1}, \ldots, D_{k_{\rho_j+1}}$. All the potential of the DPSs $D_{k_2}, \ldots, D_{k_{\rho_j}}$ is allocated to J_j, and Claim 2 holds for D_{k_1} and $D_{k_{\rho_j+1}}$. The extension of Claims 1 and 3 is straightforward: we consider the set of ρ_j weakest DPSs instead of the weakest one. Finally, Claims 4 - 6 are valid also for the general case: in any moving-window schedule, independent of ρ_j, only the first and last DPSs in the window are united, and in any greedy schedule only the last busy-segment may split (thus adding a single idle-segment). Hence, as in the case where $\forall j, \rho_j = 1$, the resulting number of segments satisfies $X_n^B \le m + n + n_w - 1$. However, n_w can be bounded by the minimal k for which $\sum_{j=1}^{k} \rho_j \ge m - \min_j \rho_j$ (Note that now, in any moving-window schedule we remove ρ_j DPSs from the DPSs list, and in the worst case, the greedy phase starts with $\min_j \rho_j$ DPSs).

Indeed, sorting the jobs by the ratios t_j/ρ_j, may not provide the minimal k; however, it guarantees that once $\mathcal{A}_p$ reaches the greedy phase, all the remaining jobs can be completed. Thus, for some instances, a different order of the jobs may result with fewer preemptions. Given that $\exists\ b\ \forall\ j,\ \rho_j \ge b$, we get that $n_w \le \lfloor \frac{m-b}{b} \rfloor$, thus, $N_s(I) = X_n^B \le m + n + \lfloor \frac{m-b}{b} \rfloor - 1 = m + n + \lfloor \frac{m}{b} \rfloor - 2$. This fits the lower bound given in Theorem 6.

Algorithm $\mathcal{A}_p$ can be implemented in $O(\max(m \log m,\ n \log n))$ steps. Specifically, $O(m \log m) + O(n \log n)$ steps are required for sorting the lists and calculating w. Given that the lists are sorted, the total time for scheduling the jobs is

$O(m+n)+O(n \log m)$. The first phase of the algorithm, in which we schedule the jobs by scanning the list L with the moving-window, can be implemented in $O(m+n)+O(n \log m)$ steps (using skip-lists). The second (greedy) phase requires $O(m+n)$ steps. We summarize our discussion of algorithm $\mathcal{A}_p$ in the next theorem.

Theorem 7. *For any m, b, and any instance I in which $\rho_j \geq b$ for all j, $\mathcal{A}_p$ finds in $O(\max(m \log m,\ n \log n))$ steps an optimal schedule of I with $N_s(I) \leq m+n+\lfloor \frac{m}{b} \rfloor - 2$.*

References

1. J. Blazewick, M. Drabowski, and J. Weglarz. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Trans. Comput*, 35(C):389–393, 1986.
2. C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *Proc. of SODA*, 213–222, 2000.
3. X. Deng, N. Gu, T. Brecht and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. In SODA'96, pp. 159–167.
4. L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. In *Proc. of ESA* LNCS 1643, 151–162, 1999.
5. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.
6. T. Gonzalez, O.H. Ibarra and S. Sahni. Bounds for LPT schedules on uniform processors. *SIAM J. on Comp.*, 6, 155–166, 1977.
7. T. Gonzalez and S. Sahni. Preemptive scheduling of uniform processor systems. *Journal of the ACM*, 25:92–101, 1978.
8. D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: Practical and theoretical results. *J. of the ACM*, 34(1):144–162, 1987.
9. D.S. Hochbaum and D.B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SICOMP*, 17(3):539–551, 1988.
10. E.G. Horvath, S. Lam, and R.Sethi. A level algorithm for preemptive scheduling. *Journal of the ACM*, 24:32–43, 1977.
11. K. Jansen, M. Sviridenko and R. Solis-Oba. Makespan minimization in job shops: a polynomial time approximation scheme. In *Proc.of STOC*, 394-399. 1999
12. R. McNaughton. Scheduling with deadlines and loss functions. *Manage. Sci.*, 6:1–12, 1959.
13. H. Shachnai and T. Tamir. Multiprocessor scheduling with machine allotment and parallelism constraints. 2001. *Algorithmica*, Vol. 32:4, 651–678, 2002.
14. H. Shachnai, T. Tamir and G.J. Woeginger. Minimizing Makespan and Preemption Costs on a System of Uniform Machines, `http://www.cs.technion.ac.il/~hadas/PUB/ucosts.ps`.
15. J.I. Munro, T. Papadakis and R. Sedgewick. Deterministic skip lists. SODA'92.
16. S.V. Sevastianov and G.J. Woeginger. Makespan minimization in open shops: A polynomial time approximation scheme. *Math. Programming* 82, 191–198, 1998.
17. D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra and D.B. Shmoys. Short shop schedules. Unpublished manuscript, 1994.