

# Minimizing Makespan and Preemption Costs on a System of Uniform Machines\*

Hadas Shachnai †‡

Tami Tamir§

Gerhard J. Woeginger¶

## 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 a *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. Our scheme for a fixed number of machines has *linear* running time, and can be applied also for instances where jobs have release dates, and for instances with *arbitrary* preemption costs. For MPS, we derive matching lower and upper bounds on the number of preemptions required by any optimal schedule. Our results for MPS hold for any instance in which a job,  $J_j$ , can be processed simultaneously by  $\rho_j$  machines, for some  $\rho_j \geq 1$ .

---

\*A preliminary version of this paper appeared in the proceedings of the *10th European Symposium on Algorithms (ESA)*, September 2002.

†Computer Science Department, The Technion, Haifa 32000, Israel. E-mail: [hadass@cs.technion.ac.il](mailto:hadass@cs.technion.ac.il).

‡Currently on a leave in Bell Laboratories, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974.

§Department of Computer Science & Engineering, Univ. of Washington, Box 352350, Seattle, WA 98195.  
E-mail: [tami@cs.washington.edu](mailto:tami@cs.washington.edu).

¶Faculty of Mathematical Sciences, Univ. of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands.  
E-mail: [woeginge@math.utwente.nl](mailto:woeginge@math.utwente.nl).

# 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 [11]. 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 problem. Formally, suppose we are given a set of jobs,  $J_1, \dots, J_n$ , with *processing requirements*  $t_1, \dots, t_n$ ; that is,  $J_j$  requires  $t_j$  *processing units*. We need to schedule the jobs on  $m$  uniform machines,  $M_1, \dots, M_m$ ; the machine  $M_i$  has the rate  $u_i \geq 1$ . The *processing time* of  $J_j$  on  $M_i$  is  $t_j/u_i$ .

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. In another variant, we are given a bound  $Tot$  on the total number of preemptions.

In the *minimum preemptions scheduling (MPS)* problem, the only feasible schedules are preemptive schedules with the 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 (see e.g. [10, 7, 8, 9, 4]). It is well known that for preemptive scheduling there exist polynomial time exact algorithms (e.g. [11, 10, 7]), whereas for non-preemptive scheduling there are no such algorithms, unless  $P=NP$  [5]. 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 an 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 [7], and the non-preemptive scheduling problem is strongly NP-hard in both [5]. 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 [11] to obtain in this case the minimum makespan, on uniform machines the problem is strongly NP-hard (see in Section 2.2).

## 1.1 Our Results

The following are our main results for the GMS and the MPS problems.

- We give (in Section 2) proofs of hardness for GMS in the following 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  $1 \leq k < 2(m-1)$ . (As shown in [7], for  $k \geq 2(m-1)$  the problem is polynomially solvable.)
- We develop (in Section 3.1) a PTAS for any instance of GMS with job-wise bounded preemptions in which the number of machines is *fixed*. Our scheme, whose running time is *linear* in the input size, can be applied also for instances where jobs have release dates, and for instances with *arbitrary* preemption costs (See in Section 3.1.1).<sup>1</sup>
- We give (in Section 3.2) PTASs for instances of GMS with arbitrary number of machines and a bound on the total number of preemptions.
- We derive matching lower and upper bounds for the minimum preemptions scheduling problem (MPS). Our results hold for any instance in which a job  $J_j$ ,  $1 \leq j \leq n$ , can be processed simultaneously by  $\rho_j$  machines, for some  $1 \leq \rho_j \leq m$ . In particular, we show (in Section 4) 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, for all  $j$ ,  $\rho_j = 1$ , 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, to obtain approximation schemes for generalized multiprocessor scheduling. We show that by appropriately selecting the parameter values, the technique can be applied to most general instances of GMS (i.e., with arbitrary release times and arbitrary preemption costs), when either the number of machines or the total number of preemptions is fixed.

## 1.2 Related Work

The GMS problem generalizes the two classical problems of scheduling on uniform machines to minimize the makespan. When for all  $j$ ,  $a_j$  is unbounded, we get the preemptive scheduling problem, denoted in standard scheduling notation  $Q|pmtn|C_{\max}$ . 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,  $Q||C_{\max}$ , which is strongly NP-hard [5]. 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  $Q||C_{\max}$ . However, these schemes cannot be adapted

---

<sup>1</sup>In GMS we assume *unit* cost per preemption.

for the GMS problem, since they rely on the fact that all jobs are *independent*. 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.

Recently, Shchepin and Vakhania studied in [15] the problem of multiprocessor scheduling with bounded number of preemptions. They showed that GMS with the bound of  $(m - 2)$  on the total number of preemptions is NP-hard, already on identical machines.

The MPS problem was studied in [7], in the case where each job can be processed at any time by *at most one* 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).

## 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  $\rho_j$  machines; the processing of  $J_j$  can be shared arbitrarily among the machines. Our first result shows that any optimal algorithm for preemptive scheduling with no parallelism can be used to find the minimum makespan of  $I$ . Note that at this point we do not attempt to minimize the number of preemptions. For an instance  $I$ , let  $w_{OPT}(I)$  denote the minimum makespan for  $I$  when preemptions are allowed.

**Theorem 2.1** *Let  $\mathcal{A}_{opt}$  be an algorithm that solves the minimum makespan problem for non-parallelizable jobs, in  $f(n)$  steps. Then  $\mathcal{A}_{opt}$  can be adapted to yield the optimal makespan for any instance of MPS, in  $f(\sum_{j=1}^n \min(\rho_j, m))$  steps.*

**Proof:** Given an input  $I$  for MPS, replace each job  $J_j$ , of processing requirement  $t_j$  and parallelism parameter  $\rho_j$ , by  $\rho_j$  jobs  $t_j$  and parallelism parameter  $\rho_j$ , by  $\rho_j$  jobs  $J_{j_1}, \dots, J_{j_{\rho_j}}$ , each of processing requirement  $t_j/\rho_j$  and parallelism parameter 1. The resulting instance  $I'$  is an input for  $\mathcal{A}_{opt}$ . Given the optimal schedule of  $\mathcal{A}_{opt}$  for  $I'$ , we allocate to any job  $J_j$  in  $I$  the processing units which were allocated to the  $\rho_j$  jobs  $J_{j_1}, \dots, J_{j_{\rho_j}}$ . We first show that the output for  $I'$  is a valid solution for  $I$ .

**Claim 2.2** *Any schedule of  $I'$  yields a feasible schedule of  $I$  of the same length.*

**Proof:** Let  $\mathcal{A}_1$  be the optimal algorithm for  $Q|pmtn|C_{\max}$  used to schedule  $I'$ . For any  $1 \leq j \leq n$ , each of the jobs  $J_{j_1}, \dots, J_{j_{\rho_j}}$  is scheduled by  $\mathcal{A}_1$  on at most one machine, at any time. Thus,  $J_j$  is processed in parallel by at most  $\rho_j$  machines. In addition, since all the jobs in  $I'$  are completed, the total number of processing units allocated to  $J_j$  is  $\rho_j \cdot \frac{t_j}{\rho_j} = t_j$ . ■

In order to show that the makespan for  $I$  is minimal, we show that the minimal makespan for  $I'$  is equal to the minimal makespan for  $I$ . This follows from the next claim:

**Claim 2.3** Any feasible schedule of  $I$  induces a schedule of  $I'$  of the same length.

**Proof:** Given a schedule of  $I$ , we show that for any  $1 \leq j \leq n$ ,  $J_j$  can be partitioned into  $\rho_j$  sub-jobs of processing requirement  $\frac{t_j}{\rho_j}$  each, such that each of these  $\rho_j$  sub-jobs is processed by at most one machine, at any time. In order to construct the sub-jobs of  $J_j$ , we scan the time interval  $[0, \ell]$  ( $\ell$  is the length of the schedule), and ‘collect’ processing units for the  $\rho_j$  sub-jobs, using the following rule: each interval of length  $y$ , in which  $J_j$  is processed by  $h \leq \rho_j$  machines, is partitioned into  $\rho_j$  equal sub-intervals, each of length  $y/\rho_j$ . We schedule each of the  $\rho_j$  sub-jobs to run on each of the  $h$  machines, in  $h$  different sub-intervals (see Figure 1). Each machine  $M_i$  provides  $\frac{u_i y}{\rho_j}$  processing units to each sub-job. Consequently, each of the sub-jobs collects the same number of processing units. When we sum over all the time intervals in which  $J_j$  is processed, each of the sub-jobs  $J_{j_1}, \dots, J_{j_{\rho_j}}$  collects exactly  $t_j/\rho_j$  processing units.

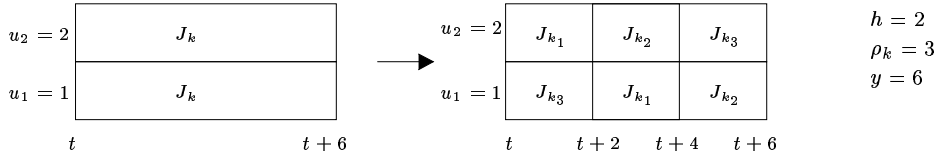


Figure 1: Partition into sub-jobs

Finally, assume that the jobs are sorted such that  $\frac{t_1}{\rho_1} \geq \frac{t_2}{\rho_2} \geq \dots \geq \frac{t_n}{\rho_n}$ , and the machines are sorted such that  $u_1 \geq u_2 \geq \dots \geq u_m$ . Let

$$w = \max\left\{\frac{T_n}{U_m}, \max_{1 \leq j \leq m} \frac{T'_j}{U_j}\right\}, \quad (1)$$

where  $U_j = \sum_{i=1}^j u_i$ ,  $T_n$  is the total processing requirement of the jobs in  $I$ , and  $T'_j$  is the total processing requirement of the first  $j$  jobs in  $I'$ . It is known ([10, 7]) that  $w_{OPT}(I') = w$ . By Claims 2.2 and 2.3,  $w_{OPT}(I') = w_{OPT}(I)$ . Thus, we have

$$w_{OPT}(I) = w. \quad (2)$$

## 2.2 Hardness Results

We show below the hardness of the GMS problem in the following special cases: (i) each of the jobs can be preempted at most *once* throughout the schedule. We call this problem *single preemption (SP)* scheduling; (ii) the overall number of preemptions is bounded by  $k$ , for some  $1 \leq k < 2(m-1)$ . Note that an algorithm that achieves the optimal makespan and uses at most  $2(m-1)$  preemptions is presented in [7].

**Theorem 2.4** The single preemption scheduling problem is strongly NP-hard.

**Proof:** We use in the proof a reduction from the *pair-partition* problem, defined as follows.

**Input:** A set  $\mathcal{A}$  of  $2k$  integers  $a_1, a_2, \dots, a_{2k}$ , and a set  $\mathcal{B}$  of  $k$  integers  $b_1, b_2, \dots, b_k$ , such that  $\sum_{j=1}^{2k} a_j = \sum_{i=1}^k b_i$ .

**Output:** A partition of  $\mathcal{A}$  into pairs such that  $a_{i_1} + a_{i_2} = b_i$ , for all  $1 \leq i \leq k$ .

**Lemma 2.5** *The pair-partition problem is strongly NP-hard.*

**Proof:** The proof is by reduction from *numerical 3-dimensional matching* (N3DM), which is strongly NP-hard ([5]): Given  $3k$  numbers  $x_1, x_2, \dots, x_k, y_1, y_2, \dots, y_k$ , and  $z_1, z_2, \dots, z_k$  such that  $\sum_i (x_i + y_i + z_i) = kC$ , is there a partition of the numbers into triples such that each triple sums up to  $C$  and contains exactly one element from each of the sets  $X, Y$ , and  $Z$ ?

Given an instance for N3DM, we construct the following instance for pair-partition:  $\mathcal{A} = \{x_1 + C, x_2 + C, \dots, x_k + C, y_1, y_2, \dots, y_k\}$ ,  $\mathcal{B} = \{2C - z_1, 2C - z_2, \dots, 2C - z_k\}$ . Clearly, a matching induces a pair-partition: any triplet satisfying  $x_i + y_j + z_\ell = C$  implies that  $\mathcal{A}, \mathcal{B}$  contain numbers  $a_{\ell_1} = x_i + C, a_{\ell_2} = y_j$  and  $b_\ell = 2C - z_\ell$  such that  $a_{\ell_1} + a_{\ell_2} = x_i + C + y_j = 2C - z_\ell = b_\ell$ . That is, any triple induces a pair and the whole matching induces a pair-partition.

To see that any pair-partition induces a matching, note that any two numbers in  $X$  contribute to  $\mathcal{A}$  numbers whose total value is larger than  $2C$ , thus, each of the  $k$  numbers that originated from  $X$  must belong to a different pair in the partition. Hence, in any pair of numbers in  $\mathcal{A}$  whose sum equals  $2C - z_\ell$ , exactly one number is originated from  $X$  and one number is originated from  $Y$ . That is, if  $a_{i_1} + a_{i_2} = b_i$  then for some  $i, j, \ell$ ,  $x_i + y_j + z_\ell = C$ , and a pair-partition induces a matching. ■

We call below the special case of GMS in which  $a_j \leq 1$ , for all  $J_j$ , the *single preemption (SP)* scheduling problem. Given an instance  $I = (\mathcal{A}, \mathcal{B})$  for pair-partition, we construct the following instance  $I'$  for the SP problem. There are  $2k$  jobs, the  $j$ th job has processing requirement  $p_j = W a_j$ , where  $W$  is a large constant (e.g., take  $W = \sum_{j=1}^{2k} a_j$ ), and  $2k$  machines:  $k$  fast machines,  $M_1, \dots, M_k$ , the  $i$ th fast machine has the rate  $u_i = W b_i - 1$ , and  $k$  slow machines,  $M_{k+1}, \dots, M_{2k}$ , each having the rate  $u_i = 1$ .

**Lemma 2.6** *If  $I$  has a pair-partition, then there is an SP schedule of  $I'$  whose makespan is equal to 1.*

**Proof:** For a pair  $a_{i_1}, a_{i_2}$  such that  $a_{i_1} + a_{i_2} = b_i$ , we schedule the jobs  $J_{i_1}$  and  $J_{i_2}$  on  $M_i$  and on one slow machine. Let  $t$  be the solution for the equation  $W a_{i_1} = (W b_i - 1)t + (1 - t)$ . Clearly, this equation has a solution  $t \in [0, 1]$ . Then we schedule  $J_{i_1}$  on  $M_i$  in the interval  $[0, t]$  and on the machine  $M_{k+i}$  in the interval  $[t, 1]$ . The job  $J_{i_2}$  is scheduled on  $M_i$  in the interval  $[t, 1]$  and on  $M_{k+i}$  in the interval  $[0, t]$ . Similarly, any pair such that  $a_{i_1} + a_{i_2} = b_i$  determines the schedule of the jobs  $J_{i_1}$  and  $J_{i_2}$  on  $M_i$ , and a pair-partition induces an SP schedule of length 1. ■

**Lemma 2.7** *If there is an SP schedule of  $I'$  whose makespan equals 1, then  $I$  has a pair-partition.*

**Proof:** We show that any SP schedule of  $I'$  whose makespan equals 1 satisfies a set of properties, as given in the next claims.

**Claim 2.8** *No machine is idle during the schedule, and each job is scheduled during the whole interval  $[0, 1]$ .*

**Proof:** Note that  $\sum_{i=1}^{2k} u_i = \sum_{i=1}^k (Wb_i - 1) + \sum_{i=k+1}^{2k} 1 = \sum_{i=1}^k Wb_i - k + k = \sum_{i=1}^k Wb_i = \sum_{j=1}^{2k} Wa_j = \sum_j p_j$ . Thus, in any schedule whose makespan is 1, no machine is idle. Also, since the number of machines is equal to the number of jobs ( $= 2k$ ), and no parallelism is allowed, each of the jobs is processed by some machine at any time along the schedule. ■

**Claim 2.9** *Each job is preempted exactly once.*

**Proof:** Since we deal with an SP schedule, each job is preempted at most once. In addition, each job is preempted at least once since for all  $i, j$ , the rate  $u_i$  is not a multiple of  $W$  and the processing requirement  $p_j$  is. Thus, in order to be allocated exactly  $p_j$  processing units during the interval  $[0, 1]$ , the processing of  $J_j$  must be shared by more than one machine. ■

**Claim 2.10** *Each machine processes exactly two jobs.*

**Proof:** Combining Claims 2.8 and 2.9, we conclude that each machine processes more than one job. Assume that some machine,  $M_i$ , processes three or more jobs. Consider the second job,  $J_{i_2}$ , processed by  $M_i$ . The processing of  $J_{i_2}$  on  $M_i$  starts after time 0 and completes before time 1. On the other hand, by Claim 2.8  $J_{i_2}$  is processed during the whole interval  $[0, 1]$ . Thus,  $J_{i_2}$  is preempted at least twice. A contradiction to Claim 2.9. ■

**Claim 2.11** *Each job is scheduled on exactly one fast machine and one slow machine.*

**Proof:** By Claim 2.9, the execution of each job consists of two segments. The slow machines are too slow to complete a job in one time unit, so each job must have at least one segment on a fast machine. On the other hand, since each machine processes exactly two jobs (by Claim 2.10), there are  $2k$  segments on the slow machines. We conclude that each of the  $2k$  jobs must have at least one segment on a slow machine. ■

Consider a fast machine  $M_i$ . By Claim 2.10,  $M_i$  processes segments of two jobs,  $J_{i_1}, J_{i_2}$ . Let  $t \in [0, 1]$  be such that  $M_i$  processes  $J_{i_1}$  in the interval  $[0, t]$  and  $J_{i_2}$  in the interval  $[t, 1]$ . By Claim 2.11, each of these two jobs has also one ‘slow segment’ in the complimentary intervals<sup>2</sup>. Then  $p_{i_1} = tu_i + (1-t)$  and  $p_{i_2} = (1-t)u_i + t$ . Thus,  $p_{i_1} + p_{i_2} = u_i + 1$ , i.e.,  $Wa_{i_1} + Wa_{i_2} = Wb_i - 1 + 1 = Wb_i$ . It follows that  $a_{i_1}$  and  $a_{i_2}$  form a pair whose sum is  $b_i$ .

Similarly, any other fast machine induces a pair and the complete schedule on the fast machines induces a pair-partition. ■

Combining Lemma 2.5 with Lemmas 2.6 and 2.7 we get the statement of Theorem 2.4. ■

**Theorem 2.12** *For any  $k \leq 2(m - 3)$ , the problem of finding a minimum makespan schedule with at most  $k$  preemptions is NP-hard.*

**Proof:** We show a reduction from non-preemptive scheduling. Given an instance,  $I$ , for non-preemptive scheduling, with jobs of processing requirements  $\{p_1, \dots, p_n\}$ , and machines with rates  $\{u_1, \dots, u_m\}$ , we construct the following instance,  $I'$ , for preemptive scheduling: Let  $k > 1$  be an even number, and let  $C > \max\{u_1, \dots, u_m\}$ . There are  $n' = n + \frac{k}{2} + 1$  jobs: the  $j$ th job  $1 \leq j \leq n$

<sup>2</sup>Not necessarily on the same slow machine.

has processing requirement  $p_j$  (as in  $I$ ), each of the other  $\frac{k}{2} + 1$  *additional jobs* has processing requirement  $\frac{kC^2+2C}{k+2}$ . There are  $m' = m + \frac{k}{2} + 1$  machines: the  $i$ th machine  $1 \leq i \leq m$  has rate  $u_i$  (as in  $I$ ), the machine  $M_{m+1}$  has the rate  $u_{m+1} = C$ , and each of the other  $\frac{k}{2}$  machines,  $M_{m+2}, \dots, M_{m+\frac{k}{2}+1}$ , has the rate  $C^2$ , these machines are denoted *fast*. Note that, for any  $m \geq 2$ , we have  $k \leq 2(m' - 3)$ . The proof follows from the next two claims in which we show that  $OPT(I) = 1$  (with no preemptions) iff  $OPT(I') = 1$  (with at most  $k$  preemptions).<sup>3</sup>

**Claim 2.13** *If there exists a non-preemptive schedule for  $I$  whose makespan is equal to 1, then there exists a preemptive schedule for  $I'$  whose makespan is equal to 1, and the total number of preemptions in the schedule is at most  $k$ .*

**Proof:** We first schedule non-preemptively the first  $n$  jobs on the first  $m$  machines. The additional jobs are scheduled on the additional machines in the following way: each job  $J_{n+1}, \dots, J_{n+\frac{k}{2}+1}$  is scheduled for  $\frac{2}{k+2}$  time units on  $M_{m+1}$ , and for  $\frac{k}{k+2}$  time units on fast machines. Thus, the total number of processing units allocated to each of the additional jobs is  $\frac{2}{k+2}C + \frac{k}{k+2}C^2 = \frac{kC^2+2C}{k+2}$ . The schedule can be done using  $k$  preemptions, as illustrated in Figure 2 (where  $J'_j$  denotes  $J_{n+j}$ ). Note that  $J_{n+1}$  and  $J_{\frac{k}{2}+1}$  are preempted once, while each of the  $\frac{k}{2} - 1$  jobs  $J_{n+2}, \dots, J_{n+\frac{k}{2}}$  is preempted twice. Thus, the total number of preemptions is  $2 + 2(\frac{k}{2} - 1) = k$ . ■

$M_{m'}$	$J'_{\frac{k}{2}+1}$		$J'_{\frac{k}{2}}$		
	⋮				
$M_{m+4}$	$J'_4$	$J'_3$			
$M_{m+3}$	$J'_3$	$J'_2$			
$M_{m+2}$	$J'_2$	$J'_1$			
$M_{m+1}$	$J'_1$	$J'_2$	$J'_3$	$J'_4$	⋯⋯⋯ $J'_{\frac{k}{2}+1}$
$M_m$	Non-preemptive schedule of $I$				
$M_1$					

Figure 2: An optimal schedule of  $I'$

**Claim 2.14** *If there exists a preemptive schedule of  $I'$  that uses at most  $k$  preemptions, and whose makespan is equal to 1, then there exists a non-preemptive schedule of  $I$  whose makespan is equal to 1.*

**Proof:** Consider any additional job,  $J_j$ , (with  $p_j = \frac{kC^2+2C}{k+2}$ ). We first show that  $J_j$  is scheduled at least  $\frac{k}{k+2}$  time units on fast machines (with rate  $C^2$ ). Let  $t \in [0, 1]$  be the total time that  $J_j$  is processed by fast machines. Note that any non-fast machine has rate  $u_i \leq C$ , thus, we must have  $p_j \leq tC^2 + (1-t)C$ . That is,  $kC^2 + 2C \leq (k+2)(tC^2 + C - tC)$ . We get that  $\frac{k(c-1)}{k(c-1)+2(c-1)} \leq t$ , or  $t \geq \frac{k}{k+2}$ . Summing up over all the additional jobs, we get that the total time allocated by

<sup>3</sup>The construction of  $I'$  can be scaled such that the value 1 can be replaced by any other value.

the fast machines to these jobs is at least  $\frac{k}{k+2}(\frac{k}{2} + 1) = \frac{k}{2}$ . On the other hand, in any schedule with makespan = 1, the total processing time of the fast machines is  $\frac{k}{2}$ . We conclude that the fast machines are dedicated to the additional jobs only, and that each additional job is executed on fast machines for *exactly*  $\frac{k}{k+2}$  time units. Moreover, in order to be completed,  $J_j$  must be allocated  $p_j - tC^2 = (1 - t)C$  processing units in the remaining  $\frac{2}{k+2}$  time units, thus, it must be processed on the (single) machine whose rate is  $C$ . We conclude that the additional jobs are scheduled on the machines  $M_{m+1}, \dots, M_{m+\frac{k}{2}+1}$ , and that each such job is processed during the entire interval  $[0, 1]$ , and must spend some time on  $M_{m+1}$ . Clearly, the first and last job on  $M_{m+1}$  are preempted at least once. Each of the other jobs on  $M_{m+1}$  is preempted at least twice. Thus, the total number of preemptions on  $M_{m+1}, \dots, M_{m+\frac{k}{2}+1}$  is at least  $2 + 2(\frac{k}{2} - 1) = k$ . Since  $k$  is the total number of preemptions in the schedule, the schedule on  $M_1, \dots, M_m$  is non-preemptive, and induces a non-preemptive schedule of  $I$ . ■ ■

### 2.3 LPT and the Power of Unlimited Preemptions

Consider the LPT algorithm, which assigns jobs to machines in order of non-increasing processing times. Each job is assigned to the machine in which its completion time will be the earliest. Clearly, the resulting schedule is non-preemptive and the parallelism constraints are preserved. In [6], it was shown that LPT yields a ratio of  $2m/(m+1)$  to the optimal *non-preemptive* makespan. A closer analysis of LPT implies the following. Let  $w$  denote the optimal preemptive makespan (as defined in Equation (1)).

**Theorem 2.15** *For a system of  $m$  uniform machines, an LPT schedule yields a  $2 - 1/m$ -approximation for  $w$ .*

**Proof:** For the case where  $n > m$ , i.e., the number of jobs is larger than the number of machines, the proof of Theorem 2.1 in [6] applies also for the preemptive case. The proof uses the fact that the makespan of any optimal non-preemptive schedule is at least  $T_n/U_m$ . By (2), this bound holds also for preemptive schedules.

We give below a proof for the case where  $n \leq m$ . We use in our proof the next claims.

**Claim 2.16** *When  $n \leq m$ , there exists an optimal schedule that uses only the  $n$  fastest machines.*

**Proof:** This clearly holds when  $n = m$ . For  $n < m$ , let  $F$  denote the set of  $n$  fastest machines. Assume that at time  $t$  some slow machine processes the job  $J_j$ . Since there is no parallelism,  $J_j$  is not processed at time  $t$  on any fast machine. Thus, some fast machine is idle at time  $t$  ( $n < m$  and without  $J_j$  only  $n - 1$  jobs are available for the fast machines). This implies that we can move  $J_j$  to that idle segment without increasing the makespan. ■

**Claim 2.17** *When  $n \leq m$ , LPT uses only the  $n$  fastest machines.*

**Proof:** This clearly holds when  $n = m$ . For  $n < m$ , let  $J_j$  be the next job to be scheduled. Some fast machine is idle at that time, since at most  $n - 1$  jobs were scheduled before  $J_j$ . Thus, LPT must select some idle fast machine for this job. ■

By Claim 2.16,  $w = T_n/U_n$ . Let  $M_k \in F$  be the machine that determines the makespan of LPT. Let  $t_n$  be the processing requirement of the last job on  $M_k$ . W.l.o.g we assume that the shortest job ( $J_n$ ) has the latest completion time; otherwise, we can remove from the instance all the jobs that completed before  $J_n$ , with no effect on the makespan of LPT.

Let  $A_i$  be the total processing time of the jobs scheduled on the machine  $M_i$  by LPT for  $i \neq k$ , and let  $A_k$  be the total processing time of jobs scheduled on  $M_k$  before  $J_n$ . We have

1.  $w_{LPT} = (A_k + t_n)/u_k$ , since  $M_k$  determines the makespan of LPT.
2.  $w_{LPT} \leq (A_i + t_n)/u_i$ , for all  $i \neq k$ , since LPT selected  $M_k$  for  $J_n$ .

We sum over the  $n$  fastest machines. By Claim 2.17,  $A_i = 0$  for any slow machines, therefore  $\sum_{i \in F} A_i = T_n - t_n$ . That is,  $w_{LPT} \cdot U_n \leq T_n + (n - 1)t_n$ . Also, since  $J_n$  is the shortest job,  $t_n \leq T_n/n$ . We get that

$$w_{LPT} \leq \frac{T_n}{U_n} + \frac{(n-1)T_n}{nU_n} = w \cdot \frac{2n-1}{n} \leq w \cdot \frac{2m-1}{m}.$$

■

We now show that the above bound is tight. Consider an instance with  $n$  jobs whose processing requirements are  $t_j = 2 - 1/m$ , and  $m$  machines, where  $u_1 = m$  and  $u_i = 1$  for  $i = 2, \dots, m$ ; then  $w_{LPT} = 2 - 1/m$ , whereas  $w = 1$ .

### 3 Approximation Schemes for GMS

#### 3.1 Fixed Number of Machines

In this section, we describe a polynomial time approximation scheme for GMS, where the number of machines,  $m$ , 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  $1 \leq j \leq n$ .

Our scheme builds on the technique introduced in [16], for open shop scheduling. Initially, we partition the jobs into subsets, by their processing times. We distinguish between ‘big’, ‘small’ and ‘tiny’ jobs. The *big* jobs, for which the number of scheduling possibilities is polynomial, are handled first. The scheme uses dynamic programming to find a feasible *preemptive* schedule of these jobs, which is within factor  $(1 + \varepsilon)$  from the optimal. Each of the big jobs,  $J_j$ , may be partitioned to at most  $(a_j + 1)$  *segments*. (The segment sizes are measured in processing units, and therefore can take values in  $(0, \max_j t_j]$ .) In scheduling the big jobs, we impose some restrictions on the starting times and sizes of the big job segments: these restrictions are needed to make the dynamic programming work.

The heart of the scheme is in the way we define ‘small’ and ‘tiny’ jobs. The *small* jobs have non-negligible processing times, yet, the number of possible schedules for these jobs may be exponentially large. We show (in Lemma 3.1) that we can select this set such that the *total*

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 in any optimal schedule, some or all of these jobs must be preempted), with only small increase in the overall completion time.

The *tiny* jobs are defined such that their processing times are much smaller than those of the big jobs. We use a simple algorithm for adding to the schedule these jobs, which are processed *non-preemptively*. Since the processing times of these jobs are very small, we show that they can fit well into the ‘holes’ generated in the schedule by the big and the small jobs. The increase in the overall completion time will result from holes that remained in the schedule after the tiny jobs were scheduled. We show that their total size is at most  $\varepsilon$  times the optimal completion time.

We need for the scheme the next lemma.

**Lemma 3.1** *There exists  $\alpha \in [(\varepsilon^3/(m^2 \cdot (a_{max} + 1)^2))^{m/\varepsilon - 1}, 1]$ , such that the set  $S$  of small jobs, selected with  $\alpha$ , satisfies*

$$\sum_{J_j \in S} t_j \leq \varepsilon P. \quad (3)$$

**Proof:** Let  $\alpha_k = (\varepsilon^3/(m^2 \cdot (a_{max} + 1)^2))^{k-1}$ , for  $1 \leq k \leq m/\varepsilon$ . Then, for each value of  $k$  we get the corresponding set of jobs  $S_k$ . Note that the sets  $S_1, \dots, S_{m/\varepsilon}$  are disjoint, and since the overall processing load in the system is at most  $mP$ , there exists a set  $S_k$ ,  $1 \leq k \leq m/\varepsilon$ , satisfying (3). We choose  $\alpha = \alpha_k$ . ■

Our scheme proceeds in the following steps. For a given  $\varepsilon > 0$ , let  $\delta = \alpha\varepsilon^3/(m(a_{max} + 1)^2)$ , where  $\alpha$  is as defined in Lemma 3.1, and  $a_{max} = \max_{1 \leq j \leq n} a_j$ .

1. Guess the minimum makespan,  $T_{\mathcal{O}}$ , within factor  $1 + \varepsilon$ .
2. Given  $T_{\mathcal{O}}$ , guess  $P$ , the maximal load (in processing units) on any machine.
3. Partition the jobs in the instance by their processing times to *big*, *small* and *tiny*, given by the sets  $B$ ,  $S$  and  $T$ ; that is, for the value of  $\alpha \in (0, 1]$  found in Lemma 3.1, we define

$$B = \{J_j \mid t_j > \alpha\varepsilon P\}, \quad S = \left\{ J_j \mid \frac{\alpha\varepsilon^4 P}{m^2(a_{max} + 1)^2} < t_j \leq \alpha\varepsilon P \right\}, \quad T = \left\{ J_j \mid t_j \leq \frac{\alpha\varepsilon^4 P}{m^2(a_{max} + 1)^2} \right\}.$$

4. Find an optimal schedule of all the jobs in  $B$ , such that
  - (P1) each job  $J_j$  is partitioned into at most  $a_j + 1$  segments; any segment is of size at least  $\alpha\varepsilon^2 P/(a_{max} + 1)^2$ , rounded up to the closest integral multiple of  $\alpha\varepsilon^4 P/((a_{max} + 1)^2 \cdot m^2)$ .
  - (P2) the starting time of each segment is an integral multiple of  $\delta T_{\mathcal{O}}$ .
5. Schedule the jobs in  $S$  non-preemptively on the machines, by assigning greedily the jobs (in arbitrary order) to *holes* on the machines. (The time gap between the end of the schedule on any machine and  $T_{\mathcal{O}}$  is also considered a ‘hole’.) This is done as follows. We say that a hole on the machine  $M_i$  is *suitable* for  $J_j$  if  $J_j$  can be scheduled non-preemptively in the hole and processed to completion. For any job  $J_j \in S$ , we look for the minimal suitable

hole, on any machine. The jobs that are scheduled in each hole are processed with no-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.

6. Schedule on the machines the jobs in  $T$ . Starting with the first hole on  $M_1$ , we fill each of the holes by scheduling sequentially (in arbitrary order) 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  $M_{i+1}$ .

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

**Lemma 3.2** *Setting the starting times of the segments of the jobs in  $B$  to be integral multiples of  $\delta T_{\mathcal{O}}$ , can increase the overall length of the schedule at most by factor  $1 + \varepsilon$ .*

**Proof:** Recall that the size of each segment of a big job  $J_j$  is at least  $\alpha\varepsilon^2 P / (a_{max} + 1)^2$ . Therefore, the number of segments of big jobs on each machine is at most  $(a_{max} + 1)^2 / (\alpha\varepsilon^2)$ . Suppose we are given a schedule of the segments with arbitrary start times; then, starting from  $i = 1$ , we shift sequentially each job segment on the machine  $M_i$  to the next possible start time, i.e. an integral multiple of  $\delta T_{\mathcal{O}}$ , such that no overlap occurs with segments of the same job processed by  $M_1, \dots, M_{i-1}$ . Note that by this we add on  $M_i$  at most  $(i - 1) \cdot (a_{max} + 1)^2 / (\alpha\varepsilon^2)$  idle time intervals, each of length at most  $\delta T_{\mathcal{O}} = \alpha\varepsilon^3 T_{\mathcal{O}} / (m(a_{max} + 1)^2)$ . Hence, the total increase in the length of the schedule on each machine is bounded by

$$\frac{\alpha\varepsilon^3 T_{\mathcal{O}}}{m(a_{max} + 1)^2} \cdot \frac{m(a_{max} + 1)^2}{\alpha\varepsilon^2} = \varepsilon T_{\mathcal{O}}.$$

■

In the discussion below, we consider only schedules in which the starting times of the segments in  $B$  are integral multiples of  $\delta T_{\mathcal{O}}$ .

**Lemma 3.3** *Any schedule of length  $T_{\mathcal{O}}$  of the segments of jobs in  $B$  can be transformed into one of length at most  $(1 + 4\varepsilon)T_{\mathcal{O}}$  in which (i) the minimal size of any segment of  $J_j$  is at least  $\alpha\varepsilon^2 P / (a_{max} + 1)^2$ , and (ii) the size of each segment is an integral multiple of  $\alpha\varepsilon^4 P / ((a_{max} + 1)^2 \cdot m^2)$ .*

**Proof:** To obtain a schedule that satisfies (i) and (ii), we proceed as follows. For any  $1 \leq j \leq n$ , if  $J_j$  has segments whose sizes are smaller than  $\alpha\varepsilon^2 P / (a_{max} + 1)^2$  then we group these segments into a single segment. Note that by that we have not increased the number of preemptions of  $J_j$ . If the total size of this segment is still small, we round it up to  $\alpha\varepsilon^2 P / (a_{max} + 1)^2$ . Clearly, there is a machine  $M_i$  which processes a segment of  $J_j$  of size  $t_j / (a_j + 1)$ . We schedule the new segment on  $M_i$  in the first possible time, that is, if there is a hole before  $T_{\mathcal{O}}$  in which we can fit the segment, we assign the segment to this hole. (Note that a segment of  $J_j$  may not fit in a hole since another segment of  $J_j$  is processed at the same time on another machine.) If the new

segment does not fit in any hole on  $M_i$ , we add the segment at the end of the schedule on this machine. Indeed, the size of the new segment of  $J_j$  is at most

$$\frac{a_j \alpha \varepsilon^2 P}{(a_{max} + 1)^2} \leq \frac{a_j \varepsilon t_j}{(a_{max} + 1)^2} \leq \frac{\varepsilon t_j}{a_j + 1}.$$

Thus, we have increased the processing requirement of  $J_j$  on  $M_i$  at most by factor  $(1 + \varepsilon)$ , with no effect on the other machines.

Now, we round up the size of each large segment of  $J_j$  to the next integral multiple of  $\alpha \varepsilon^4 P / ((a_{max} + 1)^2 \cdot m^2)$ . To bound the overall increase in the completion time of each machine, we first lower bound  $L$ , the minimum processing load on each machine. We may assume that  $L \geq \varepsilon P / m$ ; otherwise, we can move the corresponding job segments to the maximally loaded machine, and increase its overall processing load at most by  $\varepsilon P$ . Thus, while rounding up the size of each segment, we add to it at most  $(\alpha \varepsilon^3 L) / (m(a_{max} + 1)^2)$  processing units, which require on any machine at most  $\alpha \varepsilon^3 T_{\mathcal{O}} / (m(a_{max} + 1)^2)$  time units. Rounding up the segment sizes on some machine  $M_i$  we may cause overlaps in the schedule on this machine. Thus, we shift the overlapping segments on each machine to the next scheduling point. This may result in overlaps among segments of the same job running on different machines. We fix the schedule by shifting the job segments, as in the proof of Lemma 3.2, starting from the first job segment on  $M_1$ , so that each segment starts in the next possible scheduling point. We get that the total increase in the completion time of any machine is at most  $2\varepsilon T_{\mathcal{O}}$ . Overall, in the above procedure, we have increased the completion time of the jobs at most by  $4\varepsilon T_{\mathcal{O}}$ . ■

We summarize our analysis in the next result.

**Theorem 3.4** *The above scheme yields a  $(1 + \varepsilon)$ -approximation to the minimum makespan in  $O(n)$  steps.*

**Proof:** By Lemmas 3.2 and 3.3, the schedule of segments of the big jobs increases the length of any optimal schedule at most by  $5\varepsilon T_{\mathcal{O}}$ . In Steps 5. and 6. we schedule non-preemptively the jobs in  $S$  and  $T$ . Note that, by Lemma 3.1, the overall increase in the processing time of any machine due to jobs in  $S$  is at most  $\varepsilon T_{\mathcal{O}}$ . Now, consider the jobs in  $T$ . Assume as before that the minimal processing load on any machine is  $L = \varepsilon P / m$ . Thus, the processing load generated by any tiny job is smaller than  $\delta L$ , and its running time on any machine is at most  $\delta T_{\mathcal{O}}$ . If a job that is last in some hole on  $M_i$  moves to the end of the schedule then, clearly, the length of the remaining hole is smaller than  $\delta T_{\mathcal{O}}$  time units. Since after scheduling the small jobs we have at most  $(a_{max} + 1)^2 / (\alpha \varepsilon^2)$  holes on each machine, in scheduling the tiny jobs, we totally increase the length of the schedule at most by  $\varepsilon T_{\mathcal{O}}$ . The desired ratio of  $(1 + \varepsilon)$  is obtained by taking  $\varepsilon' = \varepsilon / 7$ .

For the complexity of the scheme, we first note that  $T_{\mathcal{O}}$  can be guessed in  $O(\log(1/\varepsilon))$  steps, since we have a 2-approximation from using LPT. Also,  $P$  can be guessed in  $O(\log(m/\varepsilon))$  steps, i.e., in constant time. This follows from the fact that  $T_{\mathcal{O}}$  yields a bound,  $\hat{P} = \min(\sum_{j=1}^n t_j, \max_{1 \leq i \leq m} u_i T_{\mathcal{O}})$ , on the maximum processing load of any machine. Since  $P \geq (\sum_{j=1}^n t_j) / m$ , we have an  $m$ -approximation for  $P$ .

Now, we turn to analyze Step 4. In this step, we find an optimal schedule for the big jobs. Since  $|B| \leq m / (\alpha \varepsilon)$ , we can enumerate efficiently all the possible schedules of the big jobs in the

$1/\delta$  possible starting points on all the machines. More specifically, we use for each machine,  $M_i$ , a *configuration vector*,  $\mathbf{c}_i$ , of length  $2/\delta$ . Let  $s_\ell = (\ell - 1)\delta T_{\mathcal{O}}$  denote the  $\ell$ -th possible start-time in the schedule,  $1 \leq \ell \leq 1/\delta$ . We say that  $s_\ell$  is the beginning of a hole, if a segment that occupied the machine in  $s_{\ell-1}$  completes in  $t \in (s_{\ell-1}, s_\ell]$ . Each of the first  $1/\delta$  entries in  $\mathbf{c}_i$  indicates, for any  $1 \leq \ell \leq 1/\delta$ , whether  $s_\ell$  is a start-time for some segment ('1'), a start of a hole ('2'), or none of the two ('0').

Each of the last  $1/\delta$  entries in  $\mathbf{c}_i$  gives the index of the job to which the segment that runs in  $s_\ell$  belongs. If  $s_\ell$  is part of a hole, then we put  $n + 1$  ( $J_{n+1}$  is a dummy job of processing requirement 0). Given a configuration vector for  $M_i$ , we 'cut' from the corresponding jobs the largest segments that fit into the schedule, such that each segment is of size at least  $\alpha\varepsilon^2 P / (a_{max} + 1)^2$ , given as an integral multiple of  $\alpha\varepsilon^4 P / ((a_{max} + 1)^2 \cdot m^2)$ . The number of possible configurations for  $M_i$  is  $(3|B|)^{1/\delta}$ , and the overall number of configurations is  $(3|B|)^{m/\delta} = O((\frac{1}{\alpha\varepsilon})^{1/(\alpha\varepsilon^3)})$ , which is a constant.

The running time of Steps 5. and 6. is  $O(n)$ . Summarizing, we get that the complexity of the scheme is

$$O(\log(1/\varepsilon) \cdot \log(m/\varepsilon) \cdot ((\frac{1}{\alpha\varepsilon})^{1/(\alpha\varepsilon^3)} + n)) = O(n).$$

■

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 (P1) (Step 4.) we require that the total number of preemptions in the schedule is at most  $Tot$ . In generating the configuration vectors for the machines, we discard sets of vectors in which the overall number of preemptions is larger than  $Tot$ . Hence, we get

**Theorem 3.5** *The above scheme yields a  $(1 + \varepsilon)$ -approximation to the minimum makespan, for GMS with a bound on the total number of preemptions, in  $O(n)$  steps.*

### 3.1.1 Extensions

**Jobs with release times:** Suppose that each job  $J_j$  has a (known) release time,  $r_j$ . We modify our scheme as follows.

1. Guess the minimum makespan,  $T_{\mathcal{O}}$ , within factor  $1 + \varepsilon$ . This is done by obtaining initially a  $2m$ -approximation for  $T_{\mathcal{O}}$ . Consider the Greedy algorithm that schedules each job,  $J_j$ , on the machine  $M_{i'}$  on which its processing time is minimized, i.e.  $t_{j i'} = \min_i t_{j i}$ . Clearly, Greedy is an  $m$ -approximation algorithm for  $T_{\mathcal{O}}$  when all the jobs are ready at time 0. When the jobs have release times, we apply an algorithm of Shmoys et al. [17], in which Greedy is used as a procedure to obtain a  $2m$ -approximation for  $T_{\mathcal{O}}$ .
2. Guess  $P$ , the maximal load (in processing units) on any machine. As before, this can be done using the fact that  $(\sum_j t_j)/m \leq P \leq \sum_j t_j$ .
3. Partition the jobs as before to *big*, *small* and *tiny*.

4. Find an optimal schedule of all the jobs in  $B$ , satisfying **(P1)** and **(P2)**, such that the start-time of any segment of  $J_j$  is at least  $r_j$ .
5. Schedule the jobs in  $S$  non-preemptively on the machines. In assigning 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, without increasing the number of holes in the schedule; if such a hole does not exist we add  $J_j$  at the end of the schedule, on the machine that will complete its execution first.
6. Schedule the jobs in  $T$ . We keep a list of the holes on the machines sorted in non-decreasing order by their start-times. We schedule the next job,  $J_j \in T$ , in the hole with the minimal possible start-time  $\tau \geq r_j$ . If  $J_j$  can complete, we modify the start-time of the remaining hole, and update the sorted list accordingly; if the hole is too short for  $J_j$ , we move  $J_j$  to the end of the schedule on the same machine, and mark that hole as full.

The analysis is similar to the analysis of the original scheme, except that in Step 1. we guess  $T_{\mathcal{O}}$  in  $O(\log(m/\varepsilon))$  steps (which is a constant). Thus, the complexity of the modified scheme remains  $O(n)$ .

**Arbitrary preemption costs** As shown in Theorem 3.5, our scheme can be applied also in the case where we have a bound,  $Tot$ , on the total number of preemptions. Suppose that the cost of preemption for  $J_j$  on the machine  $M_i$  is  $c_{ij}$ , for  $1 \leq i \leq m$ ,  $1 \leq j \leq n$ . Given the value  $C > 1$ , we need to find a preemptive schedule in which the makespan is minimized, and the overall preemption cost is at most  $C$ . Indeed, in Theorem 3.5 we consider a special case of this problem, where  $c_{ij} = 1$  for all  $i, j$ . Our scheme can be easily extended to apply to arbitrary preemption costs. Recall that, by [7], in any instance, we can schedule the jobs to obtain minimum makespan by using at most  $2(m-1)$  preemptions. The scheme proceeds as the above scheme, only that in Step 4. we allow  $2(m-1)$  preemptions for each job; we select a schedule whose total cost is at most  $C$ .

We summarize our discussion in the next result.

**Theorem 3.6** *For any fixed  $m > 1$ , there is a linear time approximation scheme for GMS with release dates and arbitrary preemption costs on uniform machines.*

### 3.2 Arbitrary Number of Machines

In this section, we describe approximation schemes for GMS where the number of machines is given as part of the input. We refer here to the variant of GMS in which we are given a bound,  $Tot$ , on the overall number of preemptions throughout the schedule. We first consider the special case where the machines are *identical*, and later discuss *uniform* machines.

### 3.2.1 Identical Machines

In any preemptive schedule, there might be *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 3.1** *A preemptive schedule for a GMS instance is called primitive, if it satisfies the following two conditions.*

- *Every preempted job is preempted exactly once. These two segments are processed on two different machines.*
- *Every machine processes at most two segments of preempted jobs. These segments are either processed first or last on the machine.*

**Lemma 3.7** *For every instance of GMS with an arbitrary number of identical machines and a bound on the total number of preemptions, there exists an optimal schedule that is primitive.*

**Proof:** Consider an arbitrary optimal schedule, and define an undirected graph whose vertices are the machines. There is an edge between the machines  $M_i$  and  $M_k$  if and only if they process a common (preempted) job. A connected component in this graph on  $k$  vertices corresponds to  $k$  machines and jobs with at least  $k - 1$  preemptions. The schedule for such a connected component can be replaced by the corresponding McNaughton schedule without increasing the makespan and without using more than  $k - 1$  preemptions. By repeating this procedure for every connected component, we end up with an optimal schedule that is primitive. ■

We now describe a PTAS for approximating the optimal primitive schedule. Let  $T$  denote the makespan of the LPT-schedule. By Theorem 2.15,  $T$  is a 2-approximation to the optimal makespan. Let  $\varepsilon > 0$  be the desired precision of approximation, and let  $\delta = \varepsilon/10$ . A job is *big*, if  $t_j \geq \delta T$ ; 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  $t_j$  rounded up to the next integer multiple of  $\delta^2 T$ . Rounded big jobs can only be preempted once to form two segments, whose sizes are integer multiples of  $\delta^2 T$  and at least  $\delta T$ . Let  $P_s$  denote the total processing time of all small jobs; then the rounded instance contains a number of small jobs of length  $\delta T$ , such that their total size equals  $P_s$  rounded down to the next integer multiple of  $\delta^2 T$ . 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. We skip the details of these standard methods.

**Claim 3.8** *An optimal primitive schedule for the rounded instance can be translated into a primitive schedule for the original instance that is a  $(1 + \varepsilon)$ -approximation of the optimal primitive schedule for the original instance.*

**Proof:** Since the rounded jobs are larger than the original jobs, we can process each job in the time interval in which its rounded job is scheduled. On each machine, there are at most  $1/\delta$  big jobs and at most two (the first and last) additional segments of big jobs. For each job or segment,

the rounding causes an extension of at most  $T\delta^2$ . Thus, the total extension in the makespan due to rounding of big jobs is at most  $\delta T + 2\delta^2 T$ . In addition, we round (to  $\delta T$ ) at most one group of small jobs, and we may need to compensate for rounding down the total size of the small jobs ( $P_s$ ) to the nearest multiple of  $\delta^2 T$ . Thus, the extension due to rounding of small jobs is at most  $\delta T + \delta^2 T$ . Overall, we get that the makespan may be extended by  $2\delta T + 3\delta^2 T \leq 5\delta T$ . Finally, recall that  $T \leq 2T_{opt}$ , therefore we get that the extension due to rounding is at most  $10\delta T_{opt} = \varepsilon T_{opt}$ . ■

Hence, the above scheme is a PTAS for GMS on identical machines.

**Theorem 3.9** *For any  $m > 1$ ,  $Tot > 1$ , there is a polynomial time approximation scheme for GMS on identical machines, with at most  $Tot$  preemptions.*

### 3.2.2 Uniform Machines

Assume now that the machines are uniform, and  $Tot \geq 1$  is some constant. If  $Tot \geq 2(m - 1)$  then, by [7], the problem can be solved in polynomial time. For the case where  $Tot \leq 2m - 3$ , we show that the scheme given in Section 3.1 can be extended to yield  $(1 + \varepsilon)$ -approximation for this problem. As before, we initially guess the optimal completion time,  $T_{\mathcal{O}}$ . Then, we guess the set of machines,  $\mathcal{M}_p$ , on which we may use preemptions throughout the schedule. Let  $m' = |\mathcal{M}_p|$  denote the size of this set of machines; then,  $1 \leq m' \leq Tot + 1$ . The jobs that are scheduled on the remaining set of machines,  $\mathcal{M}_{np}$ , will run non-preemptively.

We first find the set of jobs that will be processed on  $\mathcal{M}_{np}$ . We renumber the machines in  $\mathcal{M}_{np}$  by  $1, \dots, (m - m')$ . We define an instance of the *multiple knapsack* problem, with  $N = m - m'$  bins; bin  $i$  represents the machine  $M_i \in \mathcal{M}_{np}$ . Given a good guess of  $T_{\mathcal{O}}$ , the capacity of bin  $i$  is the maximum number of processing units allocated to  $M_i$  in an optimal schedule, given by  $u_i T_{\mathcal{O}}$ .

We have a set of  $n$  items; item  $j$ ,  $1 \leq j \leq n$  has the size  $t_j$ , and the same value. We need to pack a subset of the items of maximal value in the  $(m - m')$  bins. Using scaling and rounding of the item sizes and values (see, e.g., in [2]), we can guess in polynomial time the subset of items that is packed in the bins in an optimal solution. Then, we can use, e.g., the scheme of Epstein and Sgall [4] to pack these items with overflow of  $(1 + \varepsilon)$ . Thus, for a ‘good’ guess of  $T_{\mathcal{O}}$  and of the subset of jobs to be scheduled on the machines in  $\mathcal{M}_{np}$ , we find a schedule whose makespan is at most  $T_{\mathcal{O}}(1 + \varepsilon)$ .

Now, we have a smaller instance,  $I'$ , of the jobs that need to be scheduled on the machines in  $\mathcal{M}_p$ , using at most  $Tot$  preemptions. Since  $m'$  is a constant, we can use the scheme in Section 3.1 to obtain a feasible schedule of  $I'$ , whose makespan is at most  $T_{\mathcal{O}}(1 + \varepsilon)$ .

We now analyze the complexity of the scheme. The optimal makespan,  $T_{\mathcal{O}}$ , can be guessed, as before, in  $O(\log(1/\varepsilon))$  steps. We then guess in  $O(m^{(Tot+1)})$  steps the set  $\mathcal{M}_p$ . Using the technique of [2] (see also in [13]), the set of jobs (=items) that will be scheduled (=packed) on the machines in  $\mathcal{M}_{np}$  can be guessed in  $O(n^{O(1/\varepsilon \ln(1/\varepsilon))})$  steps. A non-preemptive schedule of the jobs with overall completion time at most  $(1 + \varepsilon)T_{\mathcal{O}}$  can be found in  $O(n^{1/\varepsilon^2})$  steps [4]. Finally, we apply the scheme in Section 3.1 to the instance  $I'$ . Hence, we get

**Theorem 3.10** *For any  $m > 1$ , there is a polynomial time approximation scheme for GMS on uniform machines with a fixed bound,  $Tot$ , on the number of preemptions. The running time of the scheme is  $O(m^{Tot+1} \cdot \log(\frac{1}{\varepsilon}) \cdot n^{O(1/\varepsilon \ln(1/\varepsilon))} \cdot (n^{1/\varepsilon^2} + n)) = O(n^{1/\varepsilon^2 + 1/\varepsilon \ln(1/\varepsilon)})$ .*

## 4 Minimizing the Number of Preemptions

In this section we consider the MPS problem. For convenience, we refer below to the number of *segments*,  $N_s(I)$ , generated for the instance  $I$ . (The number of segments of  $J_j$  is the number of preemptions incurred plus one.) Recall that we now allow each job  $J_j$  to be processed in parallel by  $\rho_j$  machines, for some  $1 \leq \rho_j \leq m$ . Theorem 2.1 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  $I$  (since the sub-jobs composing  $J_j$  are not necessarily scheduled adjacent to each other).

We consider the number of job-segments generated by algorithms that achieve the minimal makespan. We first derive a lower bound for  $N_s(I)$ ; then, we present an algorithm,  $\mathcal{A}_p$ , that achieves this bound.

**Theorem 4.1** *For any  $m, b$ ,  $1 \leq b \leq m$ , there exists an instance,  $I$ , in which  $\rho_j \geq b$ , for all  $j$ , and in any optimal schedule of  $I$ ,  $N_s(I) \geq m + n + \lfloor \frac{m}{b} \rfloor - 2$ .*

**Proof:** Let  $m = bn + y$ . Consider an instance with  $n = \lfloor m/b \rfloor$  jobs. For any integer  $k > 1$ , an instance that achieves the lower bound consists of one job,  $J_1$ , with  $t_1 = (b + y) + (k - 1)/n$  and  $\rho_1 = b + y$ , and  $n - 1$  identical jobs,  $J_2, \dots, J_n$ , with  $t_j = b + (k - 1)/n$  and  $\rho_j = b$ ,  $\forall 2 \leq j \leq n$ . There is one fast machines,  $M_1$ , with the rate  $u_1 = k$ , and  $m - 1$  identical slow machines,  $M_2, \dots, M_m$  with the rate  $u_i = 1$ ,  $\forall 2 \leq i \leq m$ . By equation (1), the minimal makespan for this system is  $(\sum_j t_j / \sum_i u_i) = 1$ . Since  $\sum_j t_j = \sum_i u_i$ , none of the machines is idle in the interval  $[0, 1]$ . In addition,  $m = bn + y = \sum_j \rho_j$ , thus, in order to have a schedule with no idle times, each job  $j$  must be processed by  $\rho_j$  machines at any time during the schedule. Also, since for all the jobs  $t_j = \rho_j + (k - 1)/n$ , each job must spend  $\frac{1}{n}$  of the time on the fast machine  $M_1$  (see Figure 3 for  $n = 5, m = 11$ ). It follows that the first and last job on  $M_1$  are partitioned into at least  $\rho_j + 1$  segments, and each of the other  $n - 2$  jobs is partitioned into at least  $\rho_j + 2$  segments. Thus,  $N_s(I) \geq \sum_{j=1}^n (\rho_j + 2) - 2 = nb + y + 2n - 2 = m + n + \lfloor \frac{m}{b} \rfloor - 2$ .  $\blacksquare$

### 4.1 An Upper Bound on $N_s(I)$

Let  $w$  be the length of an optimal schedule for  $I$ . Recall that, by (2),  $w$  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  $\frac{t_1}{\rho_1} \geq \dots \geq \frac{t_n}{\rho_n}$ , and the machines are sorted such that  $u_1 \leq \dots \leq u_m$ .

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  $M^1, \dots, M^r$  be a set of  $r$  machines with non-decreasing rates. Then, these  $r$

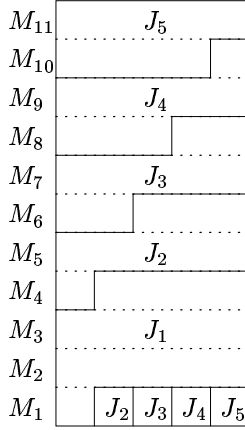


Figure 3: An example of the lower bound ( $m = 11$ ,  $n = 5$ ,  $b = 2$ )

machines form a DPS if  $M^h$  is idle exactly from  $\tau^{h-1}$  to  $\tau^h$ , where  $\tau^0 = 0$ ,  $\tau^r = w$  and  $\tau^{h-1} < \tau^h$ , for all  $1 \leq h \leq r$  (see Figure 4). Note that a DPS is defined relative to a given (partial) schedule.

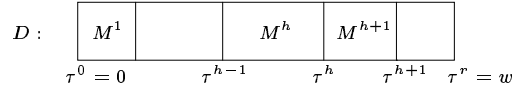


Figure 4: A DPS  $D_k$

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,  $Q_{L[1]} \leq Q_{L[2]} \leq \dots$ , 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:** The greedy-schedule 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:** The moving-window procedure schedules a job,  $J_j$ , on  $\rho_j$  DPSs whose total potential is *exactly*  $t_j$ . This set of  $\rho_j$  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  $L[1], \dots, L[\rho_j + 1]$ ). In each iteration we replace the weakest DPS in the window by the next DPS in  $L$ , until the total potential

of the  $\rho_j$  strongest DPSs in the window is large enough to complete  $J_j$ . Let  $D_{k_1}, \dots, D_{k_{\rho_j+1}}$  be the set of  $\rho_j + 1$  DPSs in the window. We first allocate to  $J_j$  all the potential of the DPSs  $D_{k_2}, \dots, D_{k_{\rho_j}}$  (if  $\rho_j = 1$  this is an empty set), and complete  $J_j$  by allocating to it also some of the potential of  $D_{k_1}, D_{k_{\rho_j+1}}$ ; this allocation is done such that the length of the schedule of  $J_j$  on these two DPSs is exactly  $w$ , and the non-used intervals of  $D_{k_1}, D_{k_{\rho_j+1}}$  form a new DPS.

Figure 5 demonstrates 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  $D_b$ .

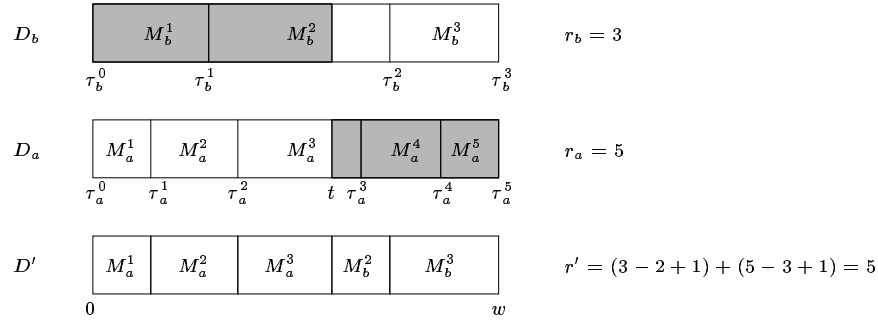


Figure 5: The schedule of  $J_j$  on consecutive DPSs

We now give a formal description of the algorithm.

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

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

We first show that  $\mathcal{A}_p$  generates a feasible 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.

Before we give the formal proof, we list some properties of  $L$ , the list of DPSs. These properties will be used to show that the parallelism constraint of each job is preserved, and that no machine processes more than one job any time.

- Initially, the list consists of  $m$  machines.
- Each time the moving window procedure is used, two DPSs are merged into a single one, and therefore the list  $L$  becomes shorter by one DPS.

- DPSs do not split. Therefore, at any moment, each machine is contained in at most one DPS.
- At any moment, the DPSs cover exactly all the idle times of all the machine.

For simplicity, we analyze  $\mathcal{A}_p$  assuming that for all  $j$ ,  $\rho_j = 1$ . In the sequel, we explain how the analysis can be applied for arbitrary  $\rho_j$ '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 move to the greedy-schedule stage, we can complete the execution of all the remaining jobs. Finally, we show that the total number of segments is at most  $n + 2(m - 1)$  (which matches the lower bound of Theorem 4.1). 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 4.2** *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  $Q_{L[i+1]} \geq t_j$ .*

**Proof:** From Equation (1), for any  $j$ ,  $w \geq \sum_{k=1}^j t_k / \sum_{k=1}^j u_k$ . In particular, the longest  $j - 1$  jobs scheduled so far, were allocated at most the processing potential of the strongest  $j - 1$  DPSs, and  $J_j$  can be processed on the strongest available DPS. This is true since the value of  $w$  considers (see Equation 1) the total processing requirement of the longest  $j$  jobs for any  $j$ . In other words, we have that  $Q_{L[|len|]} \geq t_j$ , where  $len$  is the current number of DPSs in  $L$ . Since the DPSs are sorted in non-decreasing order by their potential and  $Q_{L[1]} < t_j$ , there must be an index  $i$  for which  $Q_{L[i]} \leq t_j$  and  $Q_{L[i+1]} \geq t_j$ . ■

**Claim 4.3** *Given two DPSs  $D_a, D_b$  from  $L$ , such that  $Q_{D_a} \leq t_j$  and  $Q_{D_b} \geq t_j$ , it is always possible to schedule  $J_j$  on  $D_a, D_b$  such that the length of the schedule of  $J_j$  is exactly  $w$ .*

**Proof:** We show that for some  $t \in [0, w]$ , we can schedule  $J_j$  to run on the stronger DPS ( $D_b$ ) at the time interval  $[0, t]$ , and on the weaker DPS ( $D_a$ ) at the time interval  $[t, w]$  (see Figure 5, the shaded intervals are allocated to  $J_j$ ). The value of  $t$  can be found as follows.

1. Initially, we allocate to  $J_j$  the  $r_a$  idle-segments composing  $D_a$ .
2. As long as  $J_j$  is allocated less than  $t_j$  processing units, schedule  $J_j$  on the first idle-segment of  $D_b$  in which it is not scheduled; remove  $J_j$  for the same time interval from  $D_a$ .
3. Since  $Q_{D_a} \leq Q_{D_b}$  and  $Q_{D_b} \geq t_j$ , at some point,  $J_j$  is allocated at least  $t_j$  processing units.
4. Suppose that  $J_j$  is allocated at least  $t_j$  processing units after it is scheduled on the idle-segment  $M_b^h$  on  $D_b$ , for some  $1 \leq h \leq r_b$ . That is,  $J_j$  is now scheduled to run on the machines  $M_b^0, \dots, M_b^h$  and in the complimentary interval,  $[\tau_b^h, w]$ , on  $D_a$ .
5. Clearly,  $t \in (\tau_b^{h-1}, \tau_b^h]$ . Note that the rate of  $D_b$  in this interval is uniform (equals  $u_b^h$ ). The rate of  $D_a$  in this interval may not be uniform, since  $D_a$  may include in this interval idle-segments of more than one machine. To find the value of  $t$ , we proceed as follows. We find an interval  $I \subseteq (\tau_b^{h-1}, \tau_b^h]$  such that  $t \in I$  and the rate of  $D_a$  in  $I$  is uniform (that is,

the interval  $I$  is contained in an idle-segment of a *single* machine in  $D_a$ ). Once we have such an interval, we can find  $t$  by solving a simple equation that considers the rates of the idle segments of  $D_a, D_b$  in this interval.

To find the interval  $I$ , we initially set  $I = (\tau_b^{h-1}, \tau_b^h]$ . As long as  $I$  ‘covers’ in  $D_a$  more than one machine, let  $M_a^\ell$  be the rightmost machine in  $D_a$  included (possibly partly) in the interval. We allocate to  $J_j$  the idle-segment of  $M_a^\ell$ , and remove  $J_j$  from the complimentary interval in  $D_b$ . If, as a result of that allocation,  $J_j$  is allocated less than  $t_j$  processing units, then  $t \geq \tau_a^{\ell-1}$  and a required interval, in which the rate of  $D_a$  is uniform, is detected  $((\tau_a^{\ell-1}, \min(\tau_a^\ell, \tau_b^h)])$ . Otherwise,  $I = (\tau_b^{h-1}, \tau_a^{\ell-1})$ , and we continue to allocate to  $J_j$  the next rightmost machine in  $D_a$ . We iterate until we find a required interval (this will surely happen when  $I$  includes a single machine from  $D_a$ ).

In the simplest case, when  $r_a = r_b = 1$ ,  $t$  is the solution of the equation  $(w - t)u_a + tu_b = t_j$ . Since  $wu_a = Q_{D_a} \leq t_j$  and  $wu_b = Q_{D_b} \geq t_j$ , this equation always has a solution  $t \in [0, w]$ . When  $r_b > 1$ , we iterate as described in Step 2. above, to find an interval that contains  $t$ , and in which the rate of  $D_b$  is uniform. Then, we iterate as described in Step 5. above, to find an interval that contains  $t$ , and in which the rate of  $D_a$  is also uniform. Now we can find  $t$  by solving an equation which considers the rates of  $D_a$  and  $D_b$  in this interval.

Note that the total number of iterations in Step 2. is bounded by the number of machines from  $D_b$  that are allocated to  $J_j$ . Also, the total number of iterations in Step 5. is bounded by the number of machines from  $D_a$  that are allocated to  $J_j$ . Thus, the total number of iteration when scheduling  $J_j$  is bounded by the number of segments in its schedule. As we show below, the total number of segments in the whole schedule, which bounds the total number of iterations in Steps 2. and 5. required by all the jobs, is  $O(m + n)$ . ■

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

**Claim 4.4** *If  $Q_{L[1]} \geq t_j$ , then we can complete the schedule of the remaining jobs greedily.*

**Proof:** The length,  $w$ , of the schedule was selected such that  $w \geq \sum_{j=1}^n t_j / \sum_{i=1}^m u_i$ ; thus, the total available processing potential is at least the total processing request. It means that we only need to show that any  $J_k, j \leq k \leq n$  can be completed on a single DPS, and thus, the parallelism constraint is kept. Since the jobs are sorted in non-increasing order by their processing requirements, and since we schedule greedily on the slowest available DPSs, it is clear that for each  $J_k$ , for all  $j < k \leq n$ , when  $\mathcal{A}_p$  needs to schedule  $J_k$ , the processing potential of the weakest DPS in the next  $w$  time units is at least  $t_k$ . ■

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

**Lemma 4.5** *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 one machine is processing one job without preemptions. Throughout the algorithm, for each machine,  $M_i$ , the  $w$ -interval of  $M_i$  may consist of busy-segments, that were already allocated to some jobs, and of one *idle-segment*, which belongs to some DPS. Recall that that  $M_i$  belongs to at most one DPS. Each idle-segment may be partitioned into many busy-segments by the end of the schedule.

Denote by  $X_0$  the initial number of idle or busy segments. Clearly, we start with 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 at most by two, and in any greedy-schedule, the total number of segments may increase at most by 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. We use the following Claims.

**Claim 4.6** *If  $J_j$  is scheduled in the first phase then  $X_j \leq X_{j-1} + 2$ .*

**Proof:** Assume that  $J_j$  is scheduled on the two DPSs  $D_a, D_b$ , consisting of  $r_1, r_2$  idle-segments respectively. Let  $c_1, c_2$  denote the number of idle-segments in  $D_a, D_b$ , allocated to  $J_j$  (For example, in Figure 5,  $c_1 = 3, c_2 = 2$ ). As a result of this schedule, a new DPS,  $D'$ , with at most  $(r_1 - c_1 + 1 + r_2 - c_2 + 1)$  idle-segments is generated, and the DPSs  $D_a, D_b$  are removed from  $L$ . The number of idle-segments is therefore decreased by  $c_1 + c_2 - 2$ , while the number of busy-segments is increased by  $c_1 + c_2$ , which are allocated to  $J_j$ . Overall, we get that  $X_j = X_{j-1} + (c_1 + c_2) - (c_1 + c_2 - 2) = X_{j-1} + 2$ . ■

**Claim 4.7** *If  $J_j, j < n$ , is scheduled in the second phase then  $X_j \leq X_{j-1} + 1$ .*

**Proof:** When  $J_j$  is scheduled greedily, we assign to  $J_j$  a consecutive set of idle-segments, until it is allocated  $t_j$  processing units. The last segment in which  $J_j$  is scheduled may split: to one busy-segment — allocated to  $J_j$  and one idle-segment — on which the schedule of  $J_{j+1}$  will start. Thus, the total number of segments is increased by one. ■

**Claim 4.8**  $X_n^B \leq X_{n-1}$ .

**Proof:** When the last job,  $J_n$ , is scheduled, there are two possible scenarios:

1.  $J_n$  fits exactly into the remaining idle-segments. In this case, these idle-segments become busy-segments, and  $X_n^B = X_n = X_{n-1}$ .
2. The total size of the available idle-segments is larger than  $t_n$ . In this case,  $J_n$  will only partially use the last idle-segment, in which it is scheduled; thus,  $X_n = X_{n-1} + 1$ . However, since we are left with at least one idle-segment,  $X_n^B < X_n$ , which means that  $X_n^B \leq X_{n-1}$ .

In both cases,  $X_n^B \leq X_{n-1}$ . ■

Denote by  $n_w, n_g$  the number of jobs scheduled during the first and the second phase respectively. Clearly,  $n_g = n - n_w$ . Combining Claims 4.6-4.8, we get:

$$X_n^B \leq X_{n-1} \leq X_0 + 2n_w + n_g - 1 = m + 2n_w + n_g - 1 = m + n + n_w - 1. \quad (4)$$

To bound  $X_n^B$ , we bound  $n_w$ . Note that each job scheduled during the first phase reduces by one the size of the list of DPSs. Thus, after we schedule at most  $m - 1$  jobs, we schedule greedily

the remaining jobs. Thus,  $n_w \leq m - 1$ . From Equation (4),  $X_n^B \leq m + n + n_w - 1 \leq n + 2(m - 1)$ . This completes the proof of the lemma.  $\blacksquare$

The analysis of  $\mathcal{A}_p$  for instances with arbitrary  $\rho_j$ '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  $D_{k_1}, \dots, D_{k_{\rho_j+1}}$ . All the potential of the DPSs  $D_{k_2}, \dots, D_{k_{\rho_j}}$  is allocated to  $J_j$ , and Claim 4.3 holds for  $D_{k_1}$  and  $D_{k_{\rho_j+1}}$ . The extension of Claims 4.2 and 4.4 is straightforward: we consider the set of  $\rho_j$  weakest DPSs instead of the weakest one. Finally, Claims 4.6 - 4.8 are valid also for the general case: in any moving-window schedule, independent of  $\rho_j$ , 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 for all  $j, \rho_j = 1$ , the resulting number of segments satisfies  $X_n^B \leq m + n + n_w - 1$ . However,  $n_w$  can be bounded by the minimal  $k$  for which  $\sum_{j=1}^k \rho_j \geq m - \min_j \rho_j$ . Note that now, in any moving-window schedule, we remove  $\rho_j$  DPSs from the DPS list, and in the worst case we move to the greedy phase when we are left with  $\min_j \rho_j$  DPSs.

Indeed, sorting the jobs by the ratios  $t_j/\rho_j$  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 there exists  $b$  such that for all  $j, \rho_j \geq b$ , we get that  $n_w \leq \lfloor \frac{m-b}{b} \rfloor$ , thus,  $N_s(I) = X_n^B \leq m + n + \lfloor \frac{m-b}{b} \rfloor - 1 = m + n + \lfloor \frac{m}{b} \rfloor - 2$ . This matches the lower bound, as given in Theorem 4.1.

We now turn to compute the running time of  $\mathcal{A}_p$ . The algorithm can be implemented in  $O(\max(m \log m, n \log n))$  steps:  $O(m \log m) + O(n \log n)$  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. Recall that the jobs are sorted by their processing ratios, and each job is scheduled on exactly  $\rho_j + 1$  DPSs from  $L$ . The idea is to start scanning the list for each job,  $J_j$ , from a fixed point, which depends on  $J_j$ . Recall that in this phase, each job  $J_j, j > 1$ , is scheduled on exactly  $\rho_j + 1$  consecutive DPSs. This set of DPSs must contain the strongest DPS among those whose potential is at most  $\frac{t_j}{\rho_j}$ . Since  $L$  is sorted, finding this machine can be done (e.g., using skip-lists [12]) in  $O(\log m)$  steps. We can now find in  $O(\rho_j)$  steps the set of  $\rho_j + 1$  DPSs that will process  $J_j$ .

To find the exact schedule of  $J_j$  on the weakest and strongest DPSs contained in the window (that is, to find the time  $t \in [0, w]$  as described in Claim 4.3), we may perform some iterations until we detect an interval  $I$ , in which  $t$  can be found by solving an equation. As detailed in the proof of Claim 4.3, the total number of these iterations is at most  $O(n + m)$ . Finally, after we schedule  $J_j$ , we need to reposition in  $L$  the new DPS,  $D'$ , composed from the remainders of the first and last DPSs in the window. Since  $L$  is sorted, this can be done in  $O(\log m)$  steps.

Let  $P_1$  denote the set of jobs scheduled in the first phase. We now bound  $\sum_{J_j \in P_1} \rho_j$ . Each of these jobs,  $J_j \in P_1$ , uses up the potential of a set of  $\rho_j + 1$  DPSs, which are then omitted from

$L$ , and one DPS is added to  $L$ . This implies that  $\sum_{J_j \in P_1} \rho_j \leq m$ , and therefore, the total time required for positioning the window and scheduling the jobs in  $P_1$  is  $O(n \log m) + O(m)$ .

During the second phase of the algorithm we schedule the jobs greedily. Hence, this phase requires  $O(m + n)$  steps.

We summarize our discussion of the algorithm  $\mathcal{A}_p$  in the next theorem.

**Theorem 4.9** *For any  $m, b$ , and any instance  $I$  in which  $\rho_j \geq b$  holds for all  $j$ , algorithm  $\mathcal{A}_p$  finds in  $O(\max(m \log m, n \log n))$  steps an optimal schedule of  $I$  with  $N_s(I) \leq m + n + \lfloor \frac{m}{b} \rfloor - 2$ .*

## 5 Open Problems

Our study of GMS leaves several questions open for future work. In particular, for instances with *job-wise* bounds on the number of preemptions, is there a value  $2 < d < 2m - 1$  such that GMS is solvable in polynomial time, if  $a_j \leq d \forall j$ ?

Concerning approximation results, we have resolved the approximability of GMS instances in which the number of machines is fixed. In the case where the number of machines can be part of the input, we considered GMS instances in which we have a bound on the *total* number of preemptions. While our scheme for identical machines (in Section 3.2.1) can be modified to handle also instances with (arbitrary) job-wise bounds on the number of preemptions, the existence of a PTAS for uniform machines (and job-wise bounds) is still open, even if for all  $j$ ,  $a_j$  is a fixed constant.

Finally, our scheme (as given in Section 3.1) has *linear* running time; yet, the additive constants make it impractical. We expect that with some optimization steps (which were not considered in this paper), these constants can be reduced.

## Acknowledgment

We thank Chandra Chekuri for helpful discussions on this paper. We also thank two anonymous referees for insightful comments and suggestions.

## 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 the 11th ACM-SIAM Symposium on Discrete Algorithms*, pp. 213–222, 2000.
- [3] X. Deng, N. Gu, T. Brecht and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *Proc. of the Seventh ACM-SIAM Symposium on Discrete Algorithms*, pp. 159–167, 1996.

- [4] L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. In *Proc. of the 7th European Symposium on Algorithms*, LNCS 1643, pp. 151–162, Springer-Verlag, 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. *Journal 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. *SIAM J. of Computing*, 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] R. McNaughton. Scheduling with deadlines and loss functions. *Manage. Sci.*, 6:1–12, 1959.
- [12] J.I. Munro, T. Papadakis and R. Sedgewick. Deterministic skip lists. In *Proc. of the Third ACM-SIAM Symposium on Discrete Algorithms*, pp. 367–375, 1992.
- [13] H. Shachnai and T. Tamir, Polynomial Time Approximation Schemes for Class-Constrained Packing Problems. *J. of Scheduling*, vol. 4(6), pp. 313–338, 2001.
- [14] H. Shachnai and T. Tamir. Multiprocessor scheduling with machine allotment and parallelism constraints. *Algorithmica*, Vol. 32:4, 651–678, 2002.
- [15] E. Shchepin and N. Vakhania, Preemptive, non-preemptive and  $\pi$ -preemptive multiprocessor and open-shop scheduling. Submitted, 2003.
- [16] S.V. Sevastianov and G.J. Woeginger. Makespan minimization in open shops: A polynomial time approximation scheme. *Mathematical Programming*, Vol. 82, 191–198, 1998.
- [17] D. Shmoys, J. Wein, and D. Williamson, Scheduling parallel machines on-line. *SIAM J. Comput.* Vol. 24(6), pp. 1313–1331, 1995.