Scheduling with Processing Set Restrictions: A Survey

Joseph Y.-T. Leung
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ 07102, USA
Email: *leung@cis.njit.edu*
Phone: +1-973-596-3387
Fax: +1-973-596-5777

Chung-Lun Li*
Department of Logistics and Maritime Studies
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
Email: *lgtclli@polyu.edu.hk*
Phone: +852-2766-7410
Fax: +852-2330-2704

August 2008
Revised September 2008

*Corresponding author

Abstract

Scheduling problems with processing set restrictions have been studied extensively by computer scientists and operations researchers under different names. These include "scheduling typed task systems," "multi-purpose machine scheduling," "scheduling with eligibility constraints," "scheduling with processing set restrictions," and "semi-matchings for bipartite graphs." In this paper we survey the state of the art of these problems. Our survey covers offline and online problems, as well as nonpreemptive and preemptive scheduling environments. Our emphasis is on polynomial-time algorithms, complexity issues, and approximation schemes. While our main focus is on the makespan objective, other performance criteria are also discussed.

Keywords: Scheduling; parallel machines; processing set restrictions; computational complexity; worst-case analysis

1 Introduction

Scheduling problems with processing set restrictions have been studied extensively by computer scientists and operations researchers under different names. These include "scheduling typed task systems" (Jaffe 1980, Jansen 1994), "multi-purpose machine scheduling" (Brucker 2004, Brucker *et al.* 1997, Vairaktarakis and Cai 2003), "scheduling with eligibility constraints" (Hwang *et al.* 2004a,b, Lee *et al.* 2008, Li 2006, Lin and Li 2004, Park *et al.* 2006), "scheduling with processing set restrictions" (Glass and Kellerer 2007, Glass and Mills 2006, Huo *et al.* 2008, Li and Wang 2008, Ou *et al.* 2008), and "semi-matchings for bipartite graphs" (Harvey *et al.* 2006). The problem can be stated as follows: We are given a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of m parallel machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. Each job J_j has a processing time p_j and a set of machines $\mathcal{M}_j \subseteq \mathcal{M}$ to which it can be assigned. Our goal is to find a schedule such that each job J_j is assigned to one of the machines in $\mathcal{M}_j$ and such that the makespan $C_{\max}$ is minimized. Using the 3-field notation of Graham *et al.* (1979), we denote this problem $P|\mathcal{M}_j|C_{\max}$ if the machines are identical, $Q|\mathcal{M}_j|C_{\max}$ if the machines are uniform, and $R|\mathcal{M}_j|C_{\max}$ if the machines are unrelated. We denote these three problems as $Pm|\mathcal{M}_j|C_{\max}$, $Qm|\mathcal{M}_j|C_{\max}$, and $Rm|\mathcal{M}_j|C_{\max}$ if the number of machines, m, is fixed.

The classical problems $P||C_{\max}$, $Q||C_{\max}$, and $R||C_{\max}$ are clearly special cases of the problems $P|\mathcal{M}_j|C_{\max}$, $Q|\mathcal{M}_j|C_{\max}$, and $R|\mathcal{M}_j|C_{\max}$, respectively. In the classical problems, $\mathcal{M}_j = \mathcal{M}$ for $j = 1, 2, \ldots, n$. On the other hand, $R|\mathcal{M}_j|C_{\max}$ is a special case of the classical problem $R||C_{\max}$, since we can set the processing time of job J_j on machine M_i to $+\infty$ if machine M_i is not in $\mathcal{M}_j$. Thus, $R|\mathcal{M}_j|C_{\max}$ is equivalent to $R||C_{\max}$. Hence, we have the following hierarchy: $P|\mathcal{M}_j|C_{\max}$ is a special case of $Q|\mathcal{M}_j|C_{\max}$, which in turn is a special case of $R||C_{\max}$. An algorithm that solves $R||C_{\max}$ also solves $P|\mathcal{M}_j|C_{\max}$ and $Q|\mathcal{M}_j|C_{\max}$.

In the most general case of processing set restrictions, each $\mathcal{M}_j$ can be an arbitrary subset of $\mathcal{M}$. This corresponds to the situation where jobs have special characteristics and can only be assigned to certain machines. Vairaktarakis and Cai (2003) study such a scheduling problem, which is motivated by the throughput management of hospital operating rooms. A typical operating room

is usually equipped with high tech equipment which costs millions of dollars. Depending on the equipment available, each room may be available for use by a subset of patients. Minimizing the makespan is a good proxy for the room utilization objective. Azar *et al.* (1995) study the online version of such a scheduling problem, which has applications in wireless communication networks, where each arriving customer must be assigned a base station, from those within range, to service it. Yang (2000) also studies a class of scheduling problems encountered in steel production which has such job processing restrictions.

There are two important special cases that have received considerable attention in the literature: The *nested* processing set and the *inclusive* processing set restrictions. The nested processing set restrictions have the property that for each pair $\mathcal{M}_j$ and $\mathcal{M}_k$, either $\mathcal{M}_j$ and $\mathcal{M}_k$ are disjoint, $\mathcal{M}_j \subseteq \mathcal{M}_k$, or $\mathcal{M}_k \subseteq \mathcal{M}_j$. Inclusive processing set is a special case of nested processing set in that for every pair $\mathcal{M}_j$ and $\mathcal{M}_k$, either $\mathcal{M}_j \subseteq \mathcal{M}_k$ or $\mathcal{M}_k \subseteq \mathcal{M}_j$. In both cases, we can view the machines to be linearly ordered as $M_1, M_2, \ldots, M_m$. In the case of nested processing set restrictions, associated with each $\mathcal{M}_j$ are two machine indices a_j and b_j such that $\mathcal{M}_j = \{M_{a_j}, M_{a_j+1}, \ldots, M_{b_j}\}$. Furthermore, the intervals $[a_1, b_1]$, $[a_2, b_2]$, ..., $[a_n, b_n]$ are nested. In the case of inclusive processing set restrictions, each $\mathcal{M}_j$ is associated with a single machine index a_j such that $\mathcal{M}_j = \{M_{a_j}, M_{a_j+1}, \ldots, M_m\}$. We use $P|\mathcal{M}_j(nested)|C_{\max}$ and $Q|\mathcal{M}_j(nested)|C_{\max}$ to denote the parallel and uniform machine minimum makespan problems with nested processing set restrictions, and $P|\mathcal{M}_j(inclusive)|C_{\max}$ and $Q|\mathcal{M}_j(inclusive)|C_{\max}$ to denote the corresponding problems with inclusive processing set restrictions.

An application of scheduling problems with nested processing set restrictions in a food processing plant is provided by Glass and Mills (2006). Inclusive processing set restrictions also occur quite often in practice. Hwang *et al.* (2006) discuss applications in the service industry in which a service provider has customers categorized as platinum, gold, silver, and regular members, where those "special members" are entitled to premium services. In those applications, servers (i.e., machines) and customers (i.e., jobs) are labeled with grade of service (GoS) levels, and a customer is allowed to be served by a server only when the GoS level of the customer is no less than the GoS level of the server. Glass and Kellerer (2007) describe the following scenario where inclusive

processing set restrictions occur: Consider the assignment of computer programs to one of several computer processors with identical speed but different memory capacities. Each job has a memory requirement and can only be assigned to the processor with as much memory capacity as the job's memory requirement (see also Kafura and Shen 1977). Ou *et al.* (2008) describe the following application in loading and unloading cargoes of a vessel: There are multiple loading/unloading cranes with different weight capacity limits working in parallel. Each piece of cargo can be handled by any crane with a weight capacity limit no less than the weight of the cargo. Hence, if we view each piece of cargo as a job and each crane as a machine, then the problem is to schedule the jobs on the parallel machines, and each job has its inclusive processing set of machines.

Classical multi-machine scheduling has been studied extensively. Several surveys have been written about this subject; see, for example, Chen *et al.* (1998). By contrast, there is no survey of scheduling problems with processing set restrictions. In this survey paper we concentrate on the processing set restriction aspect. While our main focus is on scheduling problems with the makespan objective, other performance criteria will also be discussed.

The rest of this paper is organized as follows: In the next section we survey nonpreemptive scheduling algorithms, and in Section 3 we survey preemptive scheduling algorithms. In Section 4 we discuss other performance criteria, and in Section 5 we survey other scheduling models. Finally, we draw some concluding remarks in Section 6 and suggest future directions of research.

2 Nonpreemptive Scheduling Discipline

In this section we discuss algorithms for nonpreemptive scheduling models with processing set restrictions. In Section 2.1 we survey offline algorithms, and in Section 2.2 we survey online algorithms.

2.1 Offline Algorithms

Since $P||C_{\max}$ is NP-hard in the strong sense (Garey and Johnson 1979), problems $P|\mathcal{M}_j(inclusive)|C_{\max}$, $P|\mathcal{M}_j(nested)|C_{\max}$, and $P|\mathcal{M}_j|C_{\max}$ are strongly NP-hard as well. By the

theory of strong NP-hardness (Garey and Johnson 1979), one can rule out the possibility of a fully polynomial time approximation scheme (FPTAS) for $P|\mathcal{M}_j(inclusive)|C_{\max}$, $P|\mathcal{M}_j(nested)|C_{\max}$, and $P|\mathcal{M}_j|C_{\max}$ unless $P = NP$. To develop polynomial-time effective algorithms for these problems, the best one can hope for is a polynomial time approximation scheme (PTAS). At the present time, there is no known PTAS for $P|\mathcal{M}_j|C_{\max}$ and $P|\mathcal{M}_j(nested)|C_{\max}$. However, as we shall see later, there are PTASs for $P|\mathcal{M}_j(inclusive)|C_{\max}$ (Ou *et al.* 2008, Li and Wang 2008).

Because of the strong NP-hardness of the problem, researchers have designed approximation algorithms for $P|\mathcal{M}_j|C_{\max}$, $P|\mathcal{M}_j(nested)|C_{\max}$, and $P|\mathcal{M}_j(inclusive)|C_{\max}$, with increasingly better worst-case ratios. Lenstra *et al.* (1990) have developed a polynomial-time algorithm for $R|\,|C_{\max}$ with a worst-case performance ratio of 2 (i.e., solution value is guaranteed to be no more than twice the optimum). Later, Shchepin and Vakhania (2005) developed a polynomial-time algorithm for $R|\,|C_{\max}$ with an improved worst-case bound of $2-1/m$. Since $P|\mathcal{M}_j|C_{\max}$ is a special case of $R|\,|C_{\max}$, the algorithm of Shchepin and Vakhania also serves as a $(2-1/m)$-approximation algorithm for $P|\mathcal{M}_j|C_{\max}$. We note that the algorithm of Shchepin and Vakhania makes use of a linear programming method whose computational complexity has not yet been proven to be strongly polynomial.

Theorem 1 *(Shchepin and Vakhania 2005) There exists a polynomial-time algorithm for $R|\mathcal{M}_j|C_{\max}$ and $P|\mathcal{M}_j|C_{\max}$ with a worst-case ratio of $2-1/m$.*

Vairaktarakis and Cai (2003) develop a branch-and-bound algorithm to solve $P|\mathcal{M}_j|C_{\max}$ optimally. They report that the algorithm can solve problem instances for up to 50 jobs. They also develop several heuristic algorithms and compared their performances empirically. Finally, they study the amount of flexibility of the machines on the performance when compared with a set of identical machines. They show that very small amounts of flexibility appropriately distributed across the machines provide almost the same performance as a system of identical machines.

If the processing set restriction is more structured, such as nested and inclusive processing set restrictions, then there are algorithms with strongly polynomial running time and worst-case bounds of $2-1/m$ or better. Consider the case of nested processing sets. Let us label the machine

intervals $\mathcal{M}_j$ in increasing order of their levels. That is, machine intervals at the innermost level (i.e., machine intervals that do not include any other machine intervals) are labeled first, followed by machine intervals at the next level, etc., and finally followed by machine intervals at the outermost level. Jobs at the same level can be ordered in an arbitrary manner. Suppose jobs are labeled in this manner and then scheduled in increasing order of their labels. The next job, say J_j, is always scheduled on the machine $M_i \in \mathcal{M}_j$ with the least load. Glass and Kellerer (2007) show that this simple algorithm has a worst-case ratio of $2 - 1/m$.

Theorem 2 *(Glass and Kellerer 2007) There exists a strongly polynomial-time algorithm for* $P|\mathcal{M}_j(nested)|C_{\max}$ *with a worst-case ratio of* $2 - 1/m$.

For $P|\mathcal{M}_j(inclusive)|C_{\max}$, there are several efficient algorithms with worst-case bounds better than $2 - 1/m$. The earliest such algorithms were introduced by Kafura and Shen (1977). They present a "largest-memory-time-first" (LMTF) algorithm which can be described as follows: Recall that for $P|\mathcal{M}_j(inclusive)|C_{\max}$, each job J_j is associated with a machine index a_j and $\mathcal{M}_j = \{M_{a_j}, M_{a_j+1}, \ldots, M_m\}$. The LMTF algorithm initially sorts the jobs in the order $J_1, J_2, \ldots, J_n$, such that either (i) $a_j > a_{j+1}$ or (ii) $a_j = a_{j+1}$ and $p_j \geq p_{j+1}$. With this ordering of jobs, the algorithm assigns the next job J_j to a machine $M_i \in \mathcal{M}_j$ with the least load. They show that the worst-case ratio is $5/4$ for $m = 2$ and $2 - 1/(m-1)$ for $m \geq 3$. They also present an "incremental" (INC) method, which has a performance ratio of $2 - 2/(m+1)$.

Spyropoulos and Evans (1985a) develop data-dependent worst-case bounds on some of Kafura and Shen's algorithms. Hwang *et al.* (2004b) study a "lowest grade-longest processing time" (LG-LPT) algorithm, which is the same as the LMTF algorithm. They obtain the same worst-case result as Kafura and Shen, namely a worst-case ratio of $5/4$ for $m = 2$ and $2 - 1/(m-1)$ for $m \geq 3$.

Glass and Kellerer (2007) present a polynomial-time algorithm with an improved worst-case ratio. Their algorithm works as follows: Assume that the jobs have been sorted in nonincreasing order of processing times. Furthermore, assume that there are at least $m+1$ jobs; otherwise, we can eliminate some machines to obtain another instance with more jobs than machines. The algorithm calls a subroutine $\mathcal{B}(\ell)$ for $\ell = 0, 1, \ldots, m$ (i.e., it calls the subroutine a total of $m + 1$ times). For

each $\ell = 0, 1, \ldots, m$, subroutine $\mathcal{B}(\ell)$ attempts to assign the largest ℓ jobs (i.e., $J_1, J_2, \ldots, J_\ell$) to ℓ different machines, where it always assigns the next job to the eligible machine with the smallest index. If it fails to assign the largest ℓ jobs to ℓ different machines, then the ℓ^{th} iteration will not be considered further. Otherwise, $\mathcal{B}(\ell)$ calculates a bound $C(\ell)$, and the remaining jobs (i.e., jobs $J_{\ell+1}, J_{\ell+2}, \ldots, J_n$) are scheduled in an arbitrary order. Each of these jobs is assigned to the eligible machine with the lowest index such that the makespan of the resulting schedule will not exceed $3C(\ell)/2$. If it is not possible to obtain such a schedule, then subroutine $\mathcal{B}(\ell)$ will not be considered any further. Otherwise, $\mathcal{B}(\ell)$ generates a feasible solution. The algorithm outputs the best of all the feasible solutions (i.e., the one with the smallest makespan) generated by the $m + 1$ iterations. Glass and Kellerer (2007) show that this algorithm has a worst-case ratio of 3/2.

Ou *et al.* (2008) develop a polynomial-time algorithm that has a worst-case ratio of 4/3, as well as a strongly polynomial-time algorithm with a worst-case ratio of $4/3 + \epsilon$, where ϵ is a positive constant which can be set arbitrarily close to zero. Their algorithm first determines an upper bound on the optimal solution value by the LG-LPT algorithm of Hwang *et al.* (2004b). Let the makespan of this schedule be C_U, and let $C_L = C_U/2$. Then, we know that the optimal makespan C^* lies within the interval $[C_L, C_U]$. Their algorithm applies binary search to search for a desirable value $C^{(u)} \in [C_L, C_U]$ (the strongly polynomial-time algorithm uses a slightly different binary search). In each iteration of the binary search, the algorithm attempts to assign all n jobs to the m machines via a Modified First Fit Decreasing procedure with a makespan no greater than $4C^{(u)}/3$. The Modified First Fit Decreasing procedure schedules jobs on the machines in the order of $M_1, M_2, \ldots, M_m$. Let $S_1 = \{J_j \mid a_j = 1\}$. The jobs in S_1 are scheduled on M_1 in nonincreasing order of processing times. Let S_1' be the set of jobs in S_1 that cannot be scheduled on M_1, and let $S_2 = S_1' \cup \{J_j \mid a_j = 2\}$. The jobs in S_2 are scheduled on M_2 in nonincreasing order of processing times. If some jobs in S_2 cannot be scheduled on M_2, they will be considered for M_3, and so on. This process is repeated until M_m has been considered. Ou *et al.* (2008) show that the binary search algorithm will return a feasible schedule with a makespan of at most $\frac{4}{3}C^*$ (or $(\frac{4}{3} + \epsilon)C^*$ if the strongly polynomial-time binary search method is applied).

We note that Zhou *et al.* (2007) have considered a special case of $P|\mathcal{M}_j(inclusive)|C_{\max}$, where

$\{a_1, a_2, \ldots, a_n\} = \{1, k\}$ for some $1 < k \leq m$. They present an algorithm with a worst-case ratio of $4/3 + 1/2^r$, where r is the number of iterations used in the algorithm.

As stated earlier, there is no known PTAS for $P|\mathcal{M}_j|C_{\max}$ and $P|\mathcal{M}_j(nested)|C_{\max}$. However, Ou *et al.* (2008) have developed a PTAS for $P|\mathcal{M}_j(inclusive)|C_{\max}$. Li and Wang (2008) study an extension of $P|\mathcal{M}_j(inclusive)|C_{\max}$ with job release times (denoted as $P|\mathcal{M}_j(inclusive), r_j|C_{\max}$), and they present a PTAS for it. Clearly, their PTAS can be applied to $P|\mathcal{M}_j(inclusive)|C_{\max}$ as well.

We now outline the PTAS developed by Li and Wang (2008). First, we obtain a lower bound LB on the optimal solution value C^* such that $r_{\max}+1 \leq \text{LB}$ and $p_{\max} \leq \text{LB}$, where $r_{\max}$ is the largest release time and $p_{\max}$ is the largest processing time. Next, we divide each release time and processing time by LB, so that we have $C^* \geq 1$. We consider an arbitrary small positive value ϵ. Select $\bar{\epsilon} > 0$ such that $(1 + 2\bar{\epsilon})^2(1 + \bar{\epsilon}) \leq 1 + \epsilon$ and that $1/\bar{\epsilon}$ is integral. Those jobs with processing time less than $\bar{\epsilon}^2$ are called "small jobs," and the other jobs are called "big jobs." Then, we perform the following steps:

1. Round every release time down to the nearest multiple of $\bar{\epsilon}$. Let $r^{(1)}, r^{(2)}, \ldots, r^{(h)}$ be the rounded release times. Partition the job set into subsets $S_1^{(1)}, S_1^{(2)}, \ldots, S_1^{(h)}, S_2^{(1)}, S_2^{(2)}, \ldots, S_2^{(h)}, \ldots, S_m^{(1)}, S_m^{(2)}, \ldots, S_m^{(h)}$ such that $S_i^{(k)}$ contains jobs with machine index i and release time $r^{(k)}$.

2. For each subset $S_i^{(k)}$, repeatedly select any two small jobs in $S_i^{(k)}$ and merge them to form a composed job. At the end of the merging process, the processing time of every composed job is less than $2\bar{\epsilon}^2$, and there is at most one small job in $S_i^{(k)}$.

3. For each big job J_j, the processing time of J_j is in $[\bar{\epsilon}^2(1+\bar{\epsilon})^y, \bar{\epsilon}^2(1+\bar{\epsilon})^{y+1})$ for some nonnegative integer y. Round the processing time of J_j down to $\bar{\epsilon}^2(1 + \bar{\epsilon})^y$.

4. Ignore the small jobs and obtain an optimal solution to the problem, which can be done via dynamic programming. This gives us an assignment of big jobs to machines.

5. Assign each small job J_j to machine M_{a_j}.

6. Restore the big jobs' original processing times, and replace the composed jobs by their original small jobs.

7. On each machine, sequence the jobs in nondecreasing order of release times, and schedule each job to start as soon as the job has been released and the machine has completed the previous job.

Li and Wang (2008) have shown that the solution generated by this procedure has a makespan no greater than $(1+\epsilon)C^*$. After the rounding of the release times and processing times in step 3, we categorize the big jobs in such a way that two jobs belong to the same category if they have the same release time and the same processing time. Note that the number of job categories is bounded by a function of $\bar{\epsilon}$ (and the function is independent of m and n). Using this property, a dynamic program can be developed for step 4 with a polynomial running time when $\bar{\epsilon}$ is fixed. Thus, the entire solution procedure has a polynomial running time when ϵ is fixed. The following theorem summarizes the result.

Theorem 3 *(Ou et al. 2008, Li and Wang 2008) There exist PTASs for* $P|\mathcal{M}_j(inclusive)|C_{\max}$ *and* $P|\mathcal{M}_j(inclusive), r_j|C_{\max}$.

When m is fixed, $Pm|\,|C_{\max}$ is NP-hard in the ordinary sense, which implies that $Pm|\mathcal{M}_j|C_{\max}$ and $Rm|\mathcal{M}_j|C_{\max}$ are also NP-hard. Even though no FPTAS exists for $R|\mathcal{M}_j|C_{\max}$ (unless $P = NP$), there is an FPTAS for $Rm|\mathcal{M}_j|C_{\max}$ for each fixed m. Horowitz and Sahni (1976), Jansen and Porkolab (2001), and Efraimidis and Spirakis (2006) present different FPTASs for $Rm|\,|C_{\max}$ (which is equivalent to $Rm|\mathcal{M}_j|C_{\max}$). Ji and Cheng (2008) also present an FPTAS for the special case $Pm|\mathcal{M}_j(inclusive)|C_{\max}$.

2.1.1 Equal Processing Times

In this subsection we consider the scheduling of equal-processing-time jobs. Lin and Li (2004) and Harvey *et al.* (2006) independently develop polynomial-time algorithms for the problem $P|\mathcal{M}_j, p_j = 1|C_{\max}$ (i.e., $P|\mathcal{M}_j|C_{\max}$ with unit-length jobs). Their algorithms have running time

of $O(n^3 \log n)$ and $O(n^2 m)$, respectively. Lin and Li's algorithm can be generalized to solve $Q|\mathcal{M}_j, p_j = 1|C_{\max}$. Low (2002) also develops an algorithm for problem $P|\mathcal{M}_j, p_j = 1|C_{\max}$ and shows that his algorithm has a running time of $O(n^2 + mn)$ when the number of eligible machines for each job is bounded by a constant.

Li (2006) makes the following generalization of Lin and Li's results: Each job J_j has a penalty function f_j which is a nondecreasing function of the completion time of J_j. The objective is to minimize $\max_{j=1,\ldots,n}\{f_j\}$ or to minimize $\sum_{j=1}^{n} f_j$; that is, minimize the maximum penalty or the sum of the penalties. We note that the cost functions $\max_{j=1,\ldots,n}\{f_j\}$ and $\sum_{j=1}^{n} f_j$ encompass many of the standard objective functions in scheduling theory such as maximum lateness, weighted number of tardy jobs, total weighted tardiness, etc. Thus, we have the following theorem:

Theorem 4 *(Li 2006) Problems* $P|\mathcal{M}_j, p_j = 1|\max\{f_j\}$, $P|\mathcal{M}_j, p_j = 1|\sum f_j$, $Q|\mathcal{M}_j, p_j = 1|\max\{f_j\}$, *and* $Q|\mathcal{M}_j, p_j = 1|\sum f_j$ *can be solved in polynomial time.*

If the processing set restrictions are more structured, then there are even more efficient algorithms to solve the problem. Pinedo (2002, p. 103) shows that the Least Flexible Job (LFJ) algorithm solves the problem $P|\mathcal{M}_j(nested), p_j = 1|C_{\max}$ optimally. The LFJ algorithm initially sorts the jobs in nondecreasing order of the cardinality of the processing sets of the jobs. The jobs are then scheduled in the order of the list, and the next job is always assigned to a machine in the processing set with the least load. Pinedo shows the following:

Theorem 5 *(Pinedo 2002) The LFJ algorithm solves* $P|\mathcal{M}_j(nested), p_j = 1|C_{\max}$.

Lee *et al.* (2008) study the problem of scheduling equal-processing-time jobs; that is, $p_j = p$ for all j. They show that even the problem $P|\mathcal{M}_j, p_j = p, r_j|C_{\max}$ (i.e., problem $P|\mathcal{M}_j|C_{\max}$ with job release times and equal processing times) can be solved in polynomial time. The key idea of their algorithm is the fact that there is an optimal schedule with makespan equal to $r_j + kp$ for some $j, k \in \{1, 2, \ldots, n\}$. Using this fact, the problem is then transformed into a matching problem. Thus, we have the following theorem.

Theorem 6 *(Lee et al. 2008)* $P|\mathcal{M}_j, p_j = p, r_j|C_{\max}$ *can be solved in polynomial time.*

Table 1 summarizes the major worst-case analysis results for nonpreemptive offline models.

2.2 Online Algorithms

Online algorithms are much more difficult to design, since we have no knowledge of the future arrival of jobs. The performance of an online algorithm is usually compared with an optimal offline algorithm. An online algorithm A is said to have a competitive ratio c if $C_A \leq cC^*$ for all instances, where C_A is the makespan of the schedule generated by A, and C^* is the makespan of the solution of an optimal offline algorithm that knows the job arrivals in advance. An online algorithm is said to be optimal if it has a competitive ratio equal to a lower bound of the competitive ratio of the related problem. Thus, if an online algorithm has a competitive ratio of 1, or equivalently, it is 1-competitive, then it is optimal trivially. In general, online algorithms have higher competitive ratios than offline algorithms, since online algorithms have to operate without knowledge of the future arrival of jobs. In this section we survey known results of online algorithms.

There are two models of online scheduling problems studied in the literature. The first model assumes that all jobs arrive at time 0, but that the jobs are given to the scheduler one at a time. The scheduler has to make scheduling decisions before the next job is given to him, and the scheduling decisions may not be revoked. The second model assumes that jobs arrive over time, and the scheduler has to make scheduling decisions when new jobs arrive. However, the scheduler may revoke its earlier scheduling decision if the job has not been processed yet. In the case of preemptive scheduling discipline, the scheduler can preempt a job which has been assigned earlier and continue the processing at a later time. Both models have been studied in the literature. To differentiate between the two models, we will attach the symbol "r_j" to the 3-field notation for the second model, while there is no such symbol in the first model.

Azar *et al.* (1995) consider an online algorithm which assigns a job, when it arrives, to the machine with the least load among all the machines in its eligible set. They show that this algorithm achieves a competitive ratio of $\lceil \log_2 m \rceil + 1$. Furthermore, they prove that any online algorithm has a competitive ratio of at least $\lceil \log_2(m+1) \rceil$. Thus, we have the following theorem:

Theorem 7 *(Azar et al. 1995) There is an online algorithm for $P|\mathcal{M}_j|C_{\max}$ with a competitive ratio of $\lceil \log_2 m \rceil + 1$. Furthermore, any online algorithm must have a competitive ratio of at least $\lceil \log_2(m+1) \rceil$.*

We note that the above theorem is also applicable to the case of unit-length jobs; that is, the problem $P|\mathcal{M}_j, p_j = 1|C_{\max}$. Hwang *et al.* (2004a) extend Azar *et al.*'s analysis and obtain a posterior (i.e., data-dependent) competitive ratio of $\log_2(4m/\lambda) - 1/\lambda$ for $P|\mathcal{M}_j|C_{\max}$, where λ is the number of machines eligible for processing the job with the latest completion time.

Note that the online algorithm of Azar *et al.* does not yield a constant competitive ratio. Unfortunately, a data-dependent competitive ratio is the best one can hope for. The situation improves greatly when we consider inclusive processing sets. Bar-Noy *et al.* (2001) give an online algorithm with a constant competitive ratio, as stated in the next theorem:

Theorem 8 *(Bar-Noy et al. 2001) There is an online algorithm for $P|\mathcal{M}_j(inclusive)|C_{\max}$ with a competitive ratio of $e + 1 \approx 3.718$. The same algorithm gives a competitive ratio of $e \approx 2.718$ if all jobs have unit lengths.*

Jiang (2008) considers a special case of $P|\mathcal{M}_j(inclusive)|C_{\max}$ in which $\{a_1, a_2, \ldots, a_n\} = \{1, k\}$ for some $1 < k < m$, and he shows that any online algorithm must have a competitive ratio of at least 2. Furthermore, he shows that the algorithm of Azar *et al.* (1995), when applied to this special case, has a competitive ratio of $4 - 1/m$, and this bound is tight. Finally, he presents an improved algorithm with a competitive ratio of $(12 + 4\sqrt{2})/7$.

Jiang *et al.* (2006) and Park *et al.* (2006) independently present online algorithms for $P2|\mathcal{M}_j(inclusive)|C_{\max}$ with competitive ratio 5/3. They also show that any online algorithm must have a competitive ratio of at least 5/3.

Park *et al.* (2006) consider a "semi-online" variant of $P2|\mathcal{M}_j(inclusive)|C_{\max}$ in which the total processing time of all the jobs is known before any job is scheduled. They develop a semi-online algorithm for this problem with a competitive ratio of 3/2. They also show that any semi-online algorithm must have a competitive ratio of at least 3/2.

Centeno and Armacost (2004) propose several heuristic algorithms for the online version of $P|\mathcal{M}_j, r_j|C_{\max}$. They show, empirically, that scheduling jobs by the Largest Processing Time (LPT) rule outperforms scheduling jobs by the Least Flexible Job (LFJ) rule.

For nested processing set restrictions, the LFJ algorithm of Pinedo (2002) is optimal for unit-length jobs (i.e., problem $P|\mathcal{M}_j(nested), p_j = 1, r_j|C_{\max}$). If $p_j = p$ for all j, then we can use the Delayed LFJ algorithm which delays the scheduling of a job until the next integer multiple of p. Lee *et al.* (2008) show that the Delayed LFJ algorithm has an absolute error of p. Thus, we have the following theorem:

Theorem 9 *(Lee et al. 2008) The LFJ algorithm is 1-competitive for $P|\mathcal{M}_j(nested), p_j = 1, r_j|C_{\max}$. The Delayed LFJ algorithm has an absolute error of p for $P|\mathcal{M}_j(nested), p_j = p, r_j|C_{\max}$.*

Lee *et al.* (2008) also consider online scheduling of two machines with equal-processing-time jobs. They show that any online algorithm for $P2|\mathcal{M}_j, p_j = p, r_j|C_{\max}$ must have a competitive ratio of at least $(1 + \sqrt{5})/2$, and any online algorithm for $P2|\mathcal{M}_j(inclusive), p_j = p, r_j|C_{\max}$ must have a competitive ratio of at least $\sqrt{2}$. Moreover, they provide two online algorithms with competitive ratios equal to the lower bounds. Table 2 summarizes the major results for nonpreemptive online models.

3 Preemptive Scheduling Discipline

In this section we consider preemptive schedules. We first consider the offline scheduling environment and then the online scheduling environment.

3.1 Offline Algorithms

As stated in Section 1, $R|\mathcal{M}_j|C_{\max}$ is equivalent to $R||C_{\max}$. When job preemption is permitted, $R|\mathcal{M}_j, pmtn|C_{\max}$ is equivalent to $R|pmtn|C_{\max}$. Thus, an algorithm for $R|pmtn|C_{\max}$ will also solve $R|\mathcal{M}_j, pmtn|C_{\max}$. Lawler and Labetoulle (1978) have developed an algorithm to solve $R|pmtn|C_{\max}$. The idea of their algorithm is as follows: Let p_{ij} denote the processing time of job J_j

on machine M_i; that is, p_{ij} is the time required by job J_j if it is worked on exclusively by machine M_i. For a given schedule, let t_{ij} denote the time that machine M_i works on job J_j. Then, it is necessary that $\sum_{i=1}^{m} t_{ij}/p_{ij} = 1$ in order for job J_j to be completed. It is clear that the optimal makespan C^* and the values of t_{ij} constitute a feasible solution to the following linear program:

$$
\begin{aligned}
\text{minimize} \quad & C^* \\
\text{subject to} \quad & \sum_{i=1}^{m} \frac{t_{ij}}{p_{ij}} = 1 \quad (j = 1, 2, \ldots, n) \\
& \sum_{i=1}^{m} t_{ij} \leq C^* \quad (j = 1, 2, \ldots, n) \\
& \sum_{j=1}^{n} t_{ij} \leq C^* \quad (i = 1, 2, \ldots, m) \\
& t_{ij} \geq 0 \quad (i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n).
\end{aligned}
$$

Lawler and Labetoulle have shown that one can construct a schedule from a solution of this linear program. Hence, we have the following theorem:

Theorem 10 *(Lawler and Labetoulle 1978)* $R|\mathcal{M}_j, pmtn|C_{\max}$ *can be solved in polynomial time.*

Lawler and Labetoulle's algorithm can be extended to solve $R|pmtn, r_j|C_{\max}$; see Section 4 for a discussion. Their algorithm, of course, can be used to solve $R|\mathcal{M}_j, pmtn, r_j|C_{\max}$ and $P|\mathcal{M}_j, pmtn, r_j|C_{\max}$.

Huo *et al.* (2008) develop another algorithm for $P|\mathcal{M}_j, pmtn, r_j|C_{\max}$, which can be described as follows: Let $r_{[0]} < r_{[1]} < \cdots < r_{[k-1]}$ be the k distinct release times, where $r_{[0]} = 0$. The optimal makespan C^* lies in the interval $[L, U]$, where $L = r_{[k-1]}$ and $U = L + \sum_{j=1}^{n} p_j$. Their algorithm conducts a binary search in the interval $[L, U]$ to search for C^*. For each value C obtained in the binary search, the algorithm determines if there is a schedule with makespan at most C. The testing is done by solving the following network flow problem.

We divide the time span into k segments: $TS_0 = [r_{[0]}, r_{[1]}]$, $TS_1 = [r_{[1]}, r_{[2]}]$, ..., $TS_{k-1} = [r_{[k-1]}, C]$. We use ℓ_q to denote the length of segment TS_q $(0 \leq q \leq k-1)$. A network is then constructed as follows:

1. For each machine M_i and each time segment TS_q, we create a node $M_{i,q}$, called a machine-segment node.

2. For each job J_j with release time $r_{[q]}$, we create nodes $J_{j,q}, J_{j,q+1}, \ldots, J_{j,k-1}$, called the job-segment nodes.

3. For each job J_j, we create a job node J_j. Finally, we add a source node s and a sink node t.

4. For each job node J_j, we add an arc from s to J_j with capacity p_j.

5. For each job node J_j and each $q = 0, 1, \ldots, k-1$, we add an arc from J_j to $J_{j,q}$ with capacity ℓ_q if $r_{[q]} \geq r_j$.

6. For each job-segment node $J_{j,q}$ and each $i = 1, 2, \ldots, m$, we add an arc from $J_{j,q}$ to $M_{i,q}$ with capacity ℓ_q if $M_i \in \mathcal{M}_j$.

7. For each machine-segment node $M_{i,q}$, we add an arc from $M_{i,q}$ to the sink t with capacity ℓ_q.

It is not difficult to check that there is a feasible schedule with makespan at most C if and only if the maximum flow is $\sum_{j=1}^{n} p_j$. Using the max-flow algorithm of Goldberg and Tarjan (1988), the running time of the algorithm is $O(mn^2k^2 \log n \log P)$ (assuming that $m \leq n$), where $P = \sum_{j=1}^{n} p_j$ and k is the number of distinct release times.

Theorem 11 *(Huo et al. 2008)* $P|\mathcal{M}_j, pmtn, r_j|C_{\max}$ *can be solved in* $O(mn^2k^2 \log n \log P)$ *time, where* $P = \sum_{j=1}^{n} p_j$ *and* k *is the number of distinct release times.*

Kafura and Shen (1977) present an algorithm to solve $P|\mathcal{M}_j(inclusive), pmtn|C_{\max}$ with an $O(n \log n)$ running time. Martel (1983) gives an $O(n \log m)$ implementation of Kafura and Shen's algorithm. Huo *et al.* (2008) give an algorithm for $P|\mathcal{M}_j(nested), pmtn|C_{\max}$ with an $O(n \log n)$ running time. Since the algorithm of Huo *et al.* also solves $P|\mathcal{M}_j(inclusive), pmtn|C_{\max}$, we give an outline of their algorithm. Let $MI = \{MI_1, MI_2, \ldots, MI_\lambda\}$ denote the set of all distinct machine intervals; that is, for each $j \in \{1, 2, \ldots, n\}$, there is an index $k \in \{1, 2, \ldots, \lambda\}$ such that $\mathcal{M}_j = MI_k$. The average load of MI_k, denoted by $\Delta(MI_k)$, is defined as

$$\Delta(MI_k) = \frac{\sum_{\mathcal{M}_j \subseteq MI_k} p_j}{|MI_k|},$$

where $|MI_k|$ is the number of machines in MI_k. We define $\rho(MI_k)$ as

$$\rho(MI_k) = \max\Big\{ \max_{MI_{k'} \subseteq MI_k} \big\{\Delta(MI_{k'})\big\}, \max_{\mathcal{M}_j \subseteq MI_k} \big\{p_j\big\}\Big\}.$$

Clearly, $\rho(MI_k)$ is a lower bound on the makespan of all the jobs in the machine interval MI_k. A lower bound on the optimal makespan C^* is given by

$$\rho = \max\big\{\rho(MI_1), \rho(MI_2), \ldots, \rho(MI_\lambda)\big\}.$$

Huo *et al.* provide an algorithm to find a schedule with makespan exactly ρ (see Huo *et al.* 2008 for details). Since ρ is a lower bound of the optimal makespan, the algorithm of Huo *et al.* is optimal. The running time of their algorithm is $O(n \log n)$.

Theorem 12 *(Huo et al. 2008)* $P|\mathcal{M}_j(nested), pmtn|C_{\max}$ *can be solved in* $O(n \log n)$ *time.*

Martel (1983, 1985) provides an algorithm to solve $Q|\mathcal{M}_j(inclusive), pmtn|C_{\max}$. His algorithm runs in $O(mn \log^2 m)$ time. Table 3 summarizes the major offline algorithms for preemptive models.

3.2 Online Algorithms

There are very few results for online preemptive scheduling with processing set restrictions. Huo *et al.* (2008) prove the following theorem. This is in sharp contrast to $P|pmtn, r_j|C_{\max}$, which has a 1-competitive algorithm; see Hong and Leung (1992).

Theorem 13 *(Huo et al. 2008)* $P|\mathcal{M}_j(inclusive), pmtn, r_j|C_{\max}$ *cannot be solved by a 1-competitive algorithm.*

Lai and Sahni (1983) consider "nearly online" algorithms for $Q|\mathcal{M}_j(inclusive), pmtn, r_j|C_{\max}$. An algorithm is said to be nearly online if it always knows the next release time. That is, when jobs are released at time r_j, the algorithm will know the next release time, r_{j+1}. Lai and Sahni (1983) study this problem in the context of assigning jobs with memory requirements to uniform machines with memory capacity. They show that there is no 1-competitive nearly online algorithm if there is at least one machine with memory size smaller than the memory size of at least two other

machines. They then give a 1-competitive nearly online algorithm for the case where the machines do not satisfy such condition.

Jiang *et al.* (2006) consider online scheduling on two identical machines where idle time between jobs is not allowed on either machine. They show that any online algorithm for $P2|\mathcal{M}_j(inclusive), pmtn, no\ idle\ time|C_{\max}$ must have a competitive ratio of at least 3/2. Moreover, they provide an algorithm with competitive ratio of exactly 3/2. Thus, their algorithm is optimal.

Dosa and Epstein (2008) develop online algorithms for $Q2|\mathcal{M}_j(inclusive), pmtn|C_{\max}$ and $P3|\mathcal{M}_j(inclusive), pmtn|C_{\max}$, when idle time is allowed on the machines. For $Q2|\mathcal{M}_j(inclusive), pmtn|C_{\max}$, let the faster machine have speed s and the slower machine have speed 1. They give an online algorithm when M_2 is the faster machine with a competitive ratio of $\frac{s(s+1)^2}{s^3+s^2+1}$. Moroever, they show that this is also a lower bound. When M_1 is the faster machine, they give an online algorithm with a competitive ratio of $\frac{(s+1)^2}{s^2+s+1}$, and they show that this is also a lower bound. For $P3|\mathcal{M}_j(inclusive), pmtn|C_{\max}$, they give an online algorithm with competitive ratio of 3/2, and they show that any online algorithm must have a competitive ratio of at least 3/2. Thus, their algorithms are optimal. In addition, they give optimal linear-time offline algorithms for these three cases.

4 Other Performance Criteria

In this section we discuss performance criteria other than $C_{\max}$. We first consider the $L_{\max}$ objective. Since $P|\mathcal{M}_j|C_{\max}$ is NP-hard in the strong sense, so is problem $P|\mathcal{M}_j, d_j|L_{\max}$. There is very little research being done on nonpreemptive scheduling problems with processing set restrictions and the $L_{\max}$ objective. As far as we know, the only paper that deals with this topic is the one by Centeno and Armacost (1997), who develop a heuristic algorithm for $P|\mathcal{M}_j, r_j, d_j|L_{\max}$ under the assumption that the due date of each job is equal to its release date plus a constant. They evaluate the heuristic algorithm using real data from an operational environment of a semiconductor manufacturing firm.

We now turn our attention to preemptive schedules. As it turns out, the problem becomes easier when preemptions are allowed. Lawler and Labetoulle (1978) give a linear programming method to solve $R|pmtn, d_j|L_{\max}$ (which is equivalent to $R|\mathcal{M}_j, pmtn, d_j|L_{\max}$). We describe their solution method below.

Let the jobs be sorted in ascending order of their due dates; that is, $d_1 \leq d_2 \leq \cdots \leq d_n$. Let p_{ij} denote the total processing time required to complete job J_j if job J_j is worked on exclusively by machine M_i. Let $t_{ij}^{(1)}$ denote the total amount of time that machine M_i works on job J_j in the time interval $[0,\ d_1{+}L_{\max}]$. For $k = 2, 3, \ldots, n$, let $t_{ij}^{(k)}$ denote the total amount of time that machine M_i works on job J_j in the time interval $[d_{k-1}{+}L_{\max},\ d_k{+}L_{\max}]$. Consider the following linear program:

$$
\begin{aligned}
\text{minimize} \quad & L_{\max} \\
\text{subject to} \quad & \sum_{i=1}^{m} \sum_{k=1}^{j} \frac{t_{ij}^{(k)}}{p_{ij}} = 1 \quad (j = 1, 2, \ldots, n) \\
& \sum_{i=1}^{m} t_{ij}^{(1)} \leq d_1 + L_{\max} \quad (j = 1, 2, \ldots, n) \\
& \sum_{i=1}^{m} t_{ij}^{(k)} \leq d_k - d_{k-1} \quad (j = k, k+1, \ldots, n;\ k = 2, 3, \ldots, n) \\
& \sum_{j=1}^{n} t_{ij}^{(1)} \leq d_1 + L_{\max} \quad (i = 1, 2, \ldots, m) \\
& \sum_{j=k}^{n} t_{ij}^{(k)} \leq d_k - d_{k-1} \quad (i = 1, 2, \ldots, m;\ k = 2, 3, \ldots, n) \\
& t_{ij}^{(k)} \geq 0 \quad (i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n;\ k = 1, 2, \ldots, n).
\end{aligned}
$$

Lawler and Labetoulle have shown that one can construct an optimal schedule from an optimal solution of this linear program. Thus, we have the following theorem:

Theorem 14 *(Lawler and Labetoulle 1978) $R|\mathcal{M}_j, pmtn, d_j|L_{\max}$ can be solved in polynomial time.*

Note that using the same approach, we can develop a polynomial-time linear programming method for problem $R|\mathcal{M}_j, pmtn, r_j|C_{\max}$. Lai and Sahni (1984) give a more efficient algorithm for $P|\mathcal{M}_j(inclusive), pmtn, d_j|L_{\max}$ which has a running time of $O(k^2 n + n \log n)$ (assuming that $m \leq n$), where k is the number of distinct due dates.

We next consider the total completion time objective. The problem $R|\mathcal{M}_j|\sum C_j$ (which is equivalent to the problem $R||\sum C_j$) can be formulated as a weighted matching problem in a bipartite graph; see Horn (1973). Unfortunately, its preemptive version (i.e., $R|\mathcal{M}_j, pmtn|\sum C_j$) has been shown to be NP-hard in the strong sense by Sitters (2001). This is one of the rare cases in scheduling theory where the preemptive version is harder to solve than the nonpreemptive version.

Theorem 15 *(Horn 1973, Sitters 2001)* $R|\mathcal{M}_j|\sum C_j$ *can be solved in polynomial time, whereas* $R|\mathcal{M}_j, pmtn|\sum C_j$ *is NP-hard in the strong sense.*

Spyropoulos and Evans (1985b) give a faster algorithm for $P|\mathcal{M}_j(inclusive)|\sum C_j$. Their algorithm does not involve solving a matching problem, and its running time is just $O(n^2)$.

Su (2008) considers the problem of minimizing the total completion time when each job has a deadline; that is, $P|\mathcal{M}_j, \bar{d}_j|\sum C_j$, where $\bar{d}_j$ is the deadline of J_j. He notes that the problem is NP-hard in the strong sense, since $P|\bar{d}_j|\sum C_j$ is. He gives a branch-and-bound algorithm and a heuristic algorithm. The heuristic algorithm is evaluated empirically.

Logendran and Subur (2004) consider an unrelated machine scheduling model with a total weighted tardiness objective, processing set restrictions, job release times, due dates, and an additional constraint (i.e., $R|\mathcal{M}_j, r_j, d_j|\sum w_jT_j$ with an additional constraint). In their model some jobs are "split jobs." Job splitting is not part of the scheduling decision (i.e., it is determined before scheduling). However, the additional constraint specifies a maximum permissible difference between the completion time of the split portions of every original job. They develop tabu-search-based heuristic algorithms and evaluate their performances empirically.

We now consider jobs with equal processing times. We will be concentrating on an environment with uniform machines. Clearly, the algorithms will also work on an environment with identical machines. Let the processing requirement of each job be p time units on machine M_i if the job is eligible on M_i; otherwise, the processing requirement is $+\infty$. Let s_i be the speed of M_i. The total weighted number of tardy job problem (i.e., $Q|\mathcal{M}_j, p_j = p, d_j|\sum w_jU_j$) can be solved as a minimum cost matching problem of the following bipartite graph $G = (V_1 \cup V_2, A)$ (see Brucker *et al.* 1997), where

1. $V_1 = \{J_1, J_2, \ldots, J_n\}$ is the set of n jobs.

2. V_2 is the set of all ordered pairs (i, t), where i represents machine M_i, and t represents the time period $\left[\frac{(t-1)p}{s_i}, \frac{tp}{s_i}\right]$, for $i = 1, 2, \ldots, m$ and $t = 1, 2, \ldots, n$.

3. There is an arc from J_j to (i, t) for $t = 1, 2, \ldots, n$ if $M_i \in \mathcal{M}_j$.

4. The cost associated with the arc from J_j to (i, t) is given by w_j if $\frac{tp}{s_i} > d_j$, and zero otherwise.

A minimum cost matching in this graph can be obtained in $O(mn^2(n + \log m))$ time.

The total weighted tardiness problem (i.e., $Q|\mathcal{M}_j, p_j = p, d_j| \sum w_j T_j$) can also be solved by this method. In the bipartite graph constructed above, we replace the cost associated with the arc from J_j to (i, t) by $w_j \cdot \max\{0, \frac{tp}{s_i} - d_j\}$. The total weighted completion time problem (i.e., $Q|\mathcal{M}_j, p_j = p \sum w_j C_j$) is a special case of the total weighted tardiness problem and hence it can also be solved by the above method.

Theorem 16 *(Brucker et al. 1997) Problems $Q|\mathcal{M}_j, p_j = p, d_j| \sum w_j U_j$, $Q|\mathcal{M}_j, p_j = p, d_j| \sum w_j T_j$, and $Q|\mathcal{M}_j, p_j = p| \sum w_j C_j$ can all be solved in polynomial time.*

The maximum lateness problem (i.e., $Q|\mathcal{M}_j, p_j = p, d_j|L_{\max}$) can be solved by the following method: Assume we know a lower bound LB and an upper bound UB of $L_{\max}$. We conduct a binary search in the interval [LB, UB] for the optimal value of $L_{\max}$. For each value z obtained in the binary search, we modify the due date of each job by setting $d'_j = d_j + z$, where d'_j is the modified due date. We then construct the bipartite graph as above, using the modified due date instead of the original due date. If the minimum cost matching obtained has a value zero, then the optimal value of $L_{\max}$ is less than or equal to z (and hence we can search the lower half of the range). Otherwise, the optimal value of $L_{\max}$ is larger than z (and hence we can search the upper half of the range).

Theorem 17 *$Q|\mathcal{M}_j, p_j = p, d_j|L_{\max}$ can be solved in polynomial time.*

For the problem $P|\mathcal{M}_j, pmtn| \sum C_j$, Brucker *et al.* (1997) have shown that any preemptive schedule can be transformed into a nonpreemptive schedule without increasing the cost function

(i.e., $\sum C_j$). Therefore, problem $P|\mathcal{M}_j, pmtn|\sum C_j$ is equivalent to problem $P|\mathcal{M}_j|\sum C_j$ and therefore can be solved in polynomial time (since $P|\mathcal{M}_j|\sum C_j$ is a special case of $R|\mathcal{M}_j|\sum C_j$).

Theorem 18 *(Brucker et al. 1997)* $P|\mathcal{M}_j, pmtn|\sum C_j$ *can be solved in polynomial time.*

5 Other Models

In this section we survey models that do not fit into the framework of the previous sections. Nonetheless, they are still very much related to job scheduling with processing set restrictions.

5.1 Typed Task Systems

One of the earliest studies of job scheduling with processing set restrictions is the scheduling of "typed task systems" (Liu and Liu 1978, Jaffe 1980, Jansen 1994). In this model the m machines are partitioned into r different types. Machines of the same type are functionally identical, although they may have different speeds. Machines of different types are functionally different. Let m_i denote the number of machines of type i ($1 \leq i \leq r$). Thus, $m = \sum_{i=1}^{r} m_i$. Each job can be processed by one and only one type of machines, say type i (we say that the job is a type i job). Furthermore, the jobs have precedence relations defined on them. The goal is to minimize the makespan. We denote this class of problems as $P|\mathcal{M}_j(typed), prec|C_{\max}$ for identical machines and $Q|\mathcal{M}_j(typed), prec|C_{\max}$ for uniform machines.

Liu and Liu (1978) study list scheduling of $P|\mathcal{M}_j(typed), prec|C_{\max}$. For an arbitrary list $\mathcal{L}$, let $C_{\max}(\mathcal{L})$ denote the makespan of the list schedule using list $\mathcal{L}$. Let $C^*_{\max}$ denote the optimal makespan. They prove that

$$\frac{C_{\max}(\mathcal{L})}{C^*_{\max}} \leq 1 + r - \min_j \left\{\frac{1}{m_j}\right\}.$$

Moreover, they show that this bound is the best possible. They also consider another model where machines of the same type are further classified according to their memory capacities. A type i job can only be assigned to a type i machine whose memory capacity is greater than or equal to the memory requirement of the job. We can denote this model as $P|\mathcal{M}_j(typed, inclusive), prec|C_{\max}$.

Similar bounds have been obtained for an arbitrary list schedule for this model. Further details can be found in Liu and Liu (1978).

Jaffe (1980) considers uniform machines, that is, the problem $Q|\mathcal{M}_j(typed), prec|C_{\max}$. Denote the speeds of the machines of type i as $s_{i1}, s_{i2}, \ldots, s_{im_i}$. Let $f_i = \max_{k=1,\ldots,m_i}\{s_{ik}\}$, $s_i = \min_{k=1,\ldots,m_i}\{s_{ik}\}$, and $\beta_i = \sum_{k=1}^{m_i} s_{ik}$. He shows that for any list $\mathcal{L}$,

$$\frac{C_{\max}(\mathcal{L})}{C^*_{\max}} \leq r + \max_{i=1,\ldots,r}\left\{\frac{f_i}{s_i}\right\}\left(1 - \min_{i=1,\ldots,r}\left\{\frac{s_i}{\beta_i}\right\}\right).$$

Moreover, he shows that there are examples achieving a performance ratio of $r + \max_{i=1,\ldots,r}\{f_i/s_i - f_i/\beta_i\}$.

Jansen (1994) analyzes a varieity of scheduling problems with typed task systems. He shows that $P2|\mathcal{M}_j(typed), chains, p_j = p|C_{\max}$ is NP-hard in the strong sense. He also studies the complexity of deciding whether there is a schedule with $C_{\max} \leq \omega$, where ω is a small constant. For example, if $\omega = 2$, then the feasibility of $P|\mathcal{M}_j(typed), prec, p_j = 1|C_{\max}$ can be determined in polynomial time. On the other hand, if $\omega = 4$, then determining the feasibility of $P|\mathcal{M}_j(typed), outtree, p_j = 1|C_{\max}$ and $P|\mathcal{M}_j(typed), intree, p_j = 1|C_{\max}$ is NP-complete. Finally, he shows that the scheduling problem can be solved in polynomial time if the precedence relation is an interval order. We refer the interested readers to Papadimitriou and Yannakakis (1979) for a discussion of interval orders.

5.2 Machines with Availability Constraints

Some of the scheduling problems discussed in previous sections have been studied in conjunction with machine availability constraints. That is, some of the machines may not be available in certain time intervals. We use "avail" in the middle field of the 3-field notation to indicate machine availability constraints.

Sheen and Liao (2007) develop a polynomial-time algorithm for $P|\mathcal{M}_j, avail, pmtn, d_j|L_{\max}$. Their algorithm involves a series of maximum flow problems and runs in $O((3n+2x)^3 \log(\text{UB}-\text{LB}))$ time, where UB and LB are upper and lower bounds of $L_{\max}$, respectively, and x is the total number of available periods of all the machines. Sheen *et al.* (2008) present a branch-and-bound algorithm for $P|\mathcal{M}_j, avail, d_j|L_{\max}$. Liao and Sheen (2008) present a polynomial-time algorithm

for $P|\mathcal{M}_j, avail, pmtn, r_j|C_{\max}$. Their algorithm also involves a series of maximum flow problems and runs in $O((n+m)^3(n+x)^3 \log H)$ time, where x is the total number of available periods of all the machines and H is the length of the planning horizon.

5.3 Miscellaneous

In this subsection we discuss models that do not fit into any one of the previous categories. The first model is exactly the model we described in Section 1, except that jobs now have precedence constraints. Kellerer and Woeginger (1992) show that $P2|\mathcal{M}_j, prec, p_j = p|C_{\max}$ is NP-hard. They also show that the problem is polynomially solvable for some special precedence constraints. For example, they show that $P2|\mathcal{M}_j, interval\ order, p_j = p|C_{\max}$ can be solved in $O(n^{2.5})$ time. Problem $P|\mathcal{M}_j(inclusive), interval\ order, p_j = p|C_{\max}$ can be solved in polynomial time. If $|\mathcal{M}_j| = 1$ for all $j = 1, 2, \ldots, n$, then $P|\mathcal{M}_j, interval\ order, p_j = p|C_{\max}$ can also be solved in polynomial time.

Like the first model, the second model also involves jobs with precedence constraints. However, in this model, when a job is assigned to a machine that was previously idle, there is a loading time incurred. All subsequent jobs that are assigned consecutively will not incur additional loading time. Thus, the loading time is incurred only once. The objective is to minimize the total loading time of all the machines. Bhatia *et al.* (2000) show that for any constant k, no polynomial time approximation algorithm can have a worst-case ratio of $\log^k n$ or better, unless $NP \subseteq DTIME(n^{O(\log\log n)})$, where $DTIME(n^{O(\log\log n)})$ is the class of decision problems that can be solved by a deterministic Turing machine which runs in $O(n^{O(\log\log n)})$ time. (Note that it is still an open question of whether $NP \subseteq DTIME(n^{O(\log\log n)})$, although it is unlikely to be true.) Furthermore, they show that a natural greedy algorithm must have a worst-case ratio of at least $\Omega(\sqrt{n})$. Finally, they present an approximation algorithm with a worst-case ratio of m.

The third model is when the machines form a hierarchical topology; that is, each machine can be represented by a node and the nodes are connected to form a rooted tree. When a job request is made to a certain machine, then the job can be processed by that machine and all its ancestors, but not the other machines. (Note that this is different from the nested processing set restrictions.)

Bar-Noy *et al.* (2001) consider online scheduling of jobs in these kinds of hierarchical servers. They give a 4-competitive online algorithm for equal processing time jobs and a 5-competitive online algorithm for unequal processing time jobs.

The fourth model is to minimize the total setup costs of machines. Aubry *et al.* (2008) consider the problem of scheduling a set of jobs with arbitrary processing set restrictions on uniform machines. A setup cost is incurred before a job can be processed. (Note that there is no setup time involved.) Job splitting is allowed; that is, a job can be processed simultaneously by several machines. Moreover, it is desired that all machines finish processing at the same time. Aubry *et al.* called this problem the "minimum-cost load-balanced configuration problem." They show that the problem is strongly NP-hard and that it can be solved by a mixed integer linear program. Finally, they show that if all setup costs are identical, then the problem can be stated as a transportation problem and solved in polynomial time.

6 Conclusions

In this paper we have surveyed parallel scheduling problems with processing set restrictions. We have concentrated on identical, uniform, and unrelated machine environments. The performance criteria we studied include makespan, maximum lateness, total (weighted) completion time, total (weighted) number of tardy jobs, as well as total (weighted) tardiness. On the other hand, we have not considered shop-type scheduling problems such as open shops, flow shops, and job shops. The interested reader is referred to Brucker and Schlie (1990), Jurisch (1995), Brucker *et al.* (1997), and Brucker (2004) for more information about the shop-type scheduling problems with processing set restrictions.

While much work has been done on this topic, many interesting problems remain open. Below is a list of open problems that we feel are extremely worthwhile to pursue in the future. We note that some of these problems are currently under intensive investigation:

1. The simple binary search algorithms of Ou *et al.* (2008) for $P|\mathcal{M}_j(inclusive)|C_{\max}$ have worst-case bounds of $4/3$ and $4/3+\epsilon$, yet there are examples achieving a ratio of $5/4$. What

is the exact worst-case ratio of their algorithm?

2. Are there any polynomial-time algorithms for $P|\mathcal{M}_j(nested)|C_{\max}$ and $P|\mathcal{M}_j|C_{\max}$ with worst-case ratios better than $2-1/m$? In particular, are there any PTASs for these problems?

3. Is there any polynomial-time algorithm for $Q|\mathcal{M}_j(inclusive)|C_{\max}$ with a worst-case ratio better than $2-1/m$? Are there any PTASs for $Q|\mathcal{M}_j(inclusive)|C_{\max}$, $Q|\mathcal{M}_j(nested)|C_{\max}$, and $Q|\mathcal{M}_j|C_{\max}$?

4. Are there any online algorithms with constant competitive ratios for the following problems: $Q|\mathcal{M}_j(inclusive), r_j|C_{\max}$, $Q|\mathcal{M}_j(nested), r_j|C_{\max}$, $P|\mathcal{M}_j(inclusive), r_j, pmtn|C_{\max}$, $P|\mathcal{M}_j(nested), r_j, pmtn|C_{\max}$, $Q|\mathcal{M}_j(inclusive), r_j, pmtn|C_{\max}$, $Q|\mathcal{M}_j(nested), r_j, pmtn|C_{\max}$?

5. Is there any polynomial-time approximation algorithm for $P|\mathcal{M}_j|L_{\max}$ with a constant worst-case ratio?

6. What are the computational complexities of $Q|\mathcal{M}_j, pmtn|\sum C_j$, $Q|\mathcal{M}_j(inclusive), pmtn|\sum C_j$, and $Q|\mathcal{M}_j(nested), pmtn|\sum C_j$? Are these problems NP-hard or solvable in polynomial time?

7. Is there any approximation algorithm for $P|\mathcal{M}_j|\sum w_jC_j$ with a constant worst-case bound? A possible candidate for an approximation algorithm works as follows: Whenever a machine becomes free for assignment, assign that eligible job with the smallest p_i/w_i among all eligible jobs. Ties are broken in an arbitrary manner. What is the worst-case ratio of this algorithm? Note that this is a natural generalization of the weighted shortest processing time first (WSPT) rule, where Kawaguchi and Kyan (1986) have shown that the worst-case ratio is $(1+\sqrt{2})/2 \approx 1.207$ when it is applied to the classical scheduling problem $P||\sum w_jC_j$.

Acknowledgments

This research was supported in part by The Hong Kong Polytechnic University under grant 1-BBZL. The work of the first author was supported in part by the National Science Foundation under grant DMI-0556010.

References

Aubry, A., Rossi, A., Espinouse, M.-L., Jacomino, M., 2008. Minimizing setup costs for parallel multi-purpose machines under load-balancing constraint. European Journal of Operational Research 187 (3), 1115–1125.

Azar, Y., Naor, J., Rom, R., 1995. The competitiveness of on-line assignments. Journal of Algorithms 18 (2), 221–237.

Bar-Noy, A., Freund, A., Naor, J., 2001. On-line load balancing in a hierarchical server topology. SIAM Journal on Computing 31 (2), 527–549.

Bhatia, R., Khuller, S., Naor, J., 2000. The loading time scheduling problem. Journal of Algorithms 36 (1), 1–33.

Brucker, P., 2004. Scheduling Algorithms, 4th Edition. Springer, Berlin.

Brucker, P., Jurisch, B., Kramer, A., 1997. Complexity of scheduling problems with multi-purpose machines. Annals of Operations Research 70, 57–73.

Brucker, P., Schlie, R., 1990. Job-shop scheduling with multi-purpose machines. Computing 45 (4), 369–375.

Centeno, G., Armacost, R.L., 1997. Parallel machine scheduling with release time and machine eligibility restrictions. Computers and Industrial Engineering 33 (1–2), 273–276.

Centeno, G., Armacost, R.L., 2004. Minimizing makespan on parallel machines with release time and machine eligibility restrictions. International Journal of Production Research 42 (6), 1243–1256.

Chen, B., Potts, C.N., Woeginger, G.J., 1998. A review of machine scheduling: Complexity, algorithms and approximability. In: Du, D.-Z., Pardalos, P.M. (Eds.), Handbook of Combinatorial Optimization (Vol. 3), Kluwer, Boston, pp. 21–169.

Dosa, G., Epstein, L., 2008. Preemptive scheduling on a small number of hierarchical machines. Information and Computation 206 (5), 602–619.

Efraimidis, P.S., Spirakis, P.G., 2006. Approximation schemes for scheduling and covering on unrelated machines. Theoretical Computer Science 359 (1–3), 400–417.

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York.

Glass, C.A., Kellerer, H., 2007. Parallel machine scheduling with job assignment restrictions. Naval Research Logistics 54 (3), 250–257.

Glass, C.A., Mills, H.R., 2006. Scheduling unit length jobs with parallel nested machine processing set restrictions. Computers and Operations Research 33 (3), 620–638.

Goldberg, A.V., Tarjan, R.E., 1988. A new approach to the maximum-flow problem. Journal of the Association for Computing Machinery 35 (4), 921–940.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5, 287–326.

Harvey, N.J.A., Ladner, R.E., Lovasz, L., Tamir, T., 2006. Semi-matchings for bipartite graphs and load balancing. Journal of Algorithms 59 (1), 53–78.

Hong, K.S., Leung, J.Y.-T., 1992. On-line scheduling of real-time tasks. IEEE Transactions on Computers 41 (10), 1326–1331.

Horn, W.A., 1973. Minimizing average flow time with parallel machines. Operations Research 21 (3), 846–847.

Horowitz, E., Sahni, S., 1976. Exact and approximate algorithms for scheduling nonidentical processors. Journal of the Association for Computing Machinery 23 (2), 317–327.

Huo, Y., Leung, J.Y.-T., Wang, X., 2008. Preemptive scheduling algorithms with nested and inclusive processing set restrictions. Working paper. Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA.

Hwang, H.-C., Chang, S.Y., Hong, Y., 2004a. A posterior competitiveness for list scheduling algorithm on machines with eligibility constraints. Asia-Pacific Journal of Operational Research 21 (1), 117–125.

Hwang, H.-C., Chang, S.Y., Lee, K., 2004b. Parallel machine scheduling under a grade of service provision. Computers and Operations Research 31 (12), 2055–2061.

Jaffe, J.M., 1980. Bounds on the scheduling of typed task systems. SIAM Journal on Computing 9 (3), 541–551.

Jansen, K., 1994. Analysis of scheduling problems with typed task systems. Discrete Applied Mathematics 52 (3), 223–232.

Jansen, K., Porkolab, L., 2001. Improved approximation schemes for scheduling unrelated parallel machines. Mathematics of Operations Research 26 (2), 324–338.

Ji, M., Cheng, T.C.E., 2008. An FPTAS for parallel-machine scheduling under a grade of service provision to minimize makespan. Information Processing Letters, doi: 10.1016/j.ipl.2008.04.021

Jiang, Y., 2008. Online scheduling on parallel machines with two GoS levels. Journal of Combinatorial Optimization 16 (1), 28–38.

Jiang, Y.-W., He, Y., Tang, C.-M., 2006. Optimal online algorithms for scheduling on two identical machines under a grade of service. Journal of Zhejiang University SCIENCE A 7 (3), 309–314.

Jurisch, B., 1995. Lower bounds for the job-shop scheduling problem on multi-purpose machines. Discrete Applied Mathematics 58 (2), 145–156.

Kafura, D.G., Shen, V.Y., 1977. Task scheduling on a multiprocessor system with independent memories. SIAM Journal on Computing 6 (1), 167–187.

Kawaguchi, T., Kyan, S., 1986. Worst case bound of an LRF schedule for the mean weighted flow-time problem. SIAM Journal on Computing 15 (4), 1119–1129.

Kellerer, H., Woeginger, G.J., 1992. UET-scheduling with constrained processor allocations. Computers and Operations Research 19 (1), 1–8.

Lai, T.-H., Sahni, S., 1983. Nearly on-line scheduling of multiprocessor systems with memories. Journal of Algorithms 4 (4), 353–362.

Lai, T.-H., Sahni, S., 1984. Preemptive scheduling of a multiprocessor system with memories to minimize maximum lateness. SIAM Journal on Computing 13 (4), 690–704.

Lawler, E.L., Labetoulle, J., 1978. On preemptive scheduling of unrelated parallel processors by linear programming. Journal of the Association for Computing Machinery 25 (4), 612–619.

Lee, K., Leung, J.Y.-T., Pinedo, M.L., 2008. Scheduling jobs with equal processing times subject to machine eligibility constraints. Working paper. Department of Information, Operations and Management Sciences, Stern School of Business, New York University, New York, NY 10012, USA.

Lenstra, J.K., Shmoys, D.B., Tardos, E., 1990. Approximation algorithms for scheduling unrelated parallel machines. Mathematical Programming 46 (1–3), 259–271.

Li, C.-L., 2006. Scheduling unit-length jobs with machine eligibility restrictions. European Journal of Operational Research 174 (2), 1325–1328.

Li, C.-L., Wang, X., 2008. Scheduling parallel machines with inclusive processing set restrictions and job release times. Working paper. Department of Logistics, The Hong Kong Polytechnic University, Kowloon, Hong Kong.

Liao, L.-W., Sheen, G.-J., 2008. Parallel machine scheduling with machine availability and eligibility constraints. European Journal of Operational Research 184 (2), 458–467.

Lin, Y., Li, W., 2004. Parallel machine scheduling of machine-dependent jobs with unit-length. European Journal of Operational Research 156 (1), 261–266.

Liu, J.W.S., Liu, C.L., 1978. Performance analysis of multiprocessor systems containing functionally dedicated processors. Acta Informatica 10 (1), 95–104.

Logendran, R., Subur, F., 2004. Unrelated parallel machine scheduling with job splitting. IIE Transactions 36 (4), 359–372.

Low, C.P., 2002. An efficient retrieval selection algorithm for video servers with random duplicated assignment storage technique. Information Processing Letters 83 (6), 315–321.

Martel, C., 1983. Scheduling multiple processors with memory constraints. In: Ruschitzka, M., Christensen, M., Ames, W.F., Vichnevetsky, R. (Eds.), Parallel and Large-Scale Computers: Performance, Architecture, Applications, North-Holland, Amsterdam, pp. 67–75.

Martel, C., 1985. Preemptive scheduling to minimize maximum completion time on uniform processors with memory constraints. Operations Research 33 (6), 1360–1380.

Ou, J., Leung, J.Y.-T., Li, C.-L., 2008. Scheduling parallel machines with inclusive processing set restrictions. Naval Research Logistics 55 (4), 328–338.

Papadimitriou, C.H., Yannakakis, M., 1979. Scheduling interval-ordered tasks. SIAM Journal on Computing 8 (3), 405–409.

Park, J., Chang, S.Y., Lee, K., 2006. Online and semi-online scheduling of two machines under a grade of service provision. Operations Research Letters 34 (6), 692–696.

Pinedo, M.L., 2002. Scheduling: Theory, Algorithms, and Systems, 2nd Edition. Prentice-Hall, Upper Saddle River, NJ.

Shchepin, E.V., Vakhania, N., 2005. An optimal rounding gives a better approximation for scheduling unrelated machines. Operations Research Letters 33 (2), 127–133.

Sheen, G.-J., Liao, L.-W., 2007. Scheduling machine-dependent jobs to minimize lateness on machines with identical speed under availability constraints. Computers and Operations Research 34 (8), 2266–2278.

Sheen, G.-J., Liao, L.-W., Lin, C.-F., 2008. Optimal parallel machines scheduling with machine availability and eligibility constraints. The International Journal of Advanced Manufacturing Technology 36 (1–2), 132–139.

Sitters, R., 2001. Two NP-hardness results for preemptive minsum scheduling of unrelated parallel machines. Proceedings of the 8th International IPCO Conference, Lecture Notes in Computer Science 2081, Springer, Berlin, pp. 396–405.

Spyropoulos, C.D., Evans, D.J., 1985a. Generalized worst-case bounds for an homogeneous multiprocessor model with independent memories—Completion time performance criterion. Performance Evaluation 5 (4), 225–234.

Spyropoulos, C.D., Evans, D.J., 1985b. Analysis of the Q.A.D. algorithm for an homogeneous multiprocessor computing model with independent memories. International Journal of Computer Mathematics 17 (3–4), 237–255.

Su, L.-H., 2008. Scheduling on identical parallel machines to minimize total completion time with deadline and machine eligibility constraints. The International Journal of Advanced Manufacturing Technology, doi: 10.1007/s00170-007-1369-1

Vairaktarakis, G.L., Cai, X., 2003. The value of processing flexibility in multipurpose machines. IIE Transactions 35 (8), 763–774.

Yang, X., 2000. A class of generalized multiprocessor scheduling problems. Systems Science and Mathematical Sciences 13 (4), 385–390.

Zhou, P., Jiang, Y.-W., He, Y., 2007. Parallel machine scheduling problem with two GoS levels. Applied Mathematics: A Journal of Chinese Universities (Series A) 22 (3), 275–284 (in Chinese).

Table 1. Nonpreemptive offline worst-case results

Problem	Polynomial-time approximation	References
$R\|\mathcal{M}_j\|C_{\max}$	$(2-1/m)$-approximation	Shchepin and Vakhania (2005)
$P\|\mathcal{M}_j(inclusive), r_j\|C_{\max}$	PTAS	Li and Wang (2008)
$Rm\|\mathcal{M}_j\|C_{\max}$	FPTAS	Horowitz and Sahni (1976), Jansen and Porkolab (2001), Efraimidis and Spirakis (2006)
$Q\|\mathcal{M}_j, p_j=1\|\max\{f_j\}$	Optimal algorithm	Li (2006)
$P\|\mathcal{M}_j, p_j=p, r_j\|C_{\max}$	Optimal algorithm	Lee *et al.* (2008)

Table 2. Nonpreemptive online worst-case results

Problem	Competitive ratio	References
$P\|\mathcal{M}_j\|C_{\max}$	$\lceil \log_2 m \rceil + 1$	Azar *et al.* (1995)
$P\|\mathcal{M}_j(inclusive)\|C_{\max}$	$e+1 \approx 3.718$	Bar-Noy *et al.* (2001)
$P2\|\mathcal{M}_j(inclusive)\|C_{\max}$	$5/3$	Jiang *et al.* (2006), Park *et al.* (2006)
$P\|\mathcal{M}_j(nested), p_j=1, r_j\|C_{\max}$	1	Lee *et al.* (2008)
$P2\|\mathcal{M}_j, p_j=p, r_j\|C_{\max}$	$(1+\sqrt{5})/2$	Lee *et al.* (2008)
$P2\|\mathcal{M}_j(inclusive), p_j=p, r_j\|C_{\max}$	$\sqrt{2}$	Lee *et al.* (2008)

Table 3. Preemptive optimal offline algorithms

Problem	Computational complexity	References
$R\|\mathcal{M}_j, pmtn, r_j\|C_{\max}$	Polynomial time	Lawler and Labetoulle (1978)
$P\|\mathcal{M}_j, pmtn, r_j\|C_{\max}$	$O(mn^2k^2 \log n \log P)$, where k is the number of distinct due dates	Huo *et al.* (2008)
$P\|\mathcal{M}_j(inclusive), pmtn\|C_{\max}$	$O(n \log m)$	Martel (1983)
$P\|\mathcal{M}_j(nested), pmtn\|C_{\max}$	$O(n \log n)$	Huo *et al.* (2008)
$Q\|\mathcal{M}_j(inclusive), pmtn\|C_{\max}$	$O(mn \log^2 m)$	Martel (1983,1985)