General Multiprocessor Task Scheduling

Jianer Chen,[1] Chung-Yee Lee[2]
[1] *Department of Computer Science, Texas A&M University, College Station, Texas, USA 77843-3112*

[2] *Department of Industrial Engineering, Texas A&M University, College Station, Texas, USA 77843-3131*

Received July 1997; revised June 1998; accepted 25 August 1998

Abstract: Most papers in the scheduling field assume that a job can be processed by only one machine at a time. Namely, they use a one-job-on-one-machine model. In many industry settings, this may not be an adequate model. Motivated by human resource planning, diagnosable microprocessor systems, berth allocation, and manufacturing systems that may require several resources simultaneously to process a job, we study the problem with a one-job-on-multiple-machine model. In our model, there are several alternatives that can be used to process a job. In each alternative, several machines need to process simultaneously the job assigned. Our purpose is to select an alternative for each job and then to schedule jobs to minimize the completion time of all jobs. In this paper, we provide a pseudopolynomial algorithm to solve optimally the two-machine problem, and a combination of a fully polynomial scheme and a heuristic to solve the three-machine problem. We then extend the results to a general m-machine problem. Our algorithms also provide an effective lower bounding scheme which lays the foundation for solving optimally the general m-machine problem. Furthermore, our algorithms can also be applied to solve a special case of the three-machine problem in pseudopolynomial time. Both pseudopolynomial algorithms (for two-machine and three-machine problems) are much more efficient than those in the literature. © 1999 John Wiley & Sons, Inc. Naval Research Logistics 46: 57–74, 1999

INTRODUCTION

Due to the popularity of just-in-time and total quality management concepts, scheduling has played an important role in satisfying customer's expectation for on-time delivery. In the last four decades, many papers have been published in the scheduling area. There is a common assumption in the scheduling literature of a one-job-on-one-machine pattern. That is, at a given time, each job can be processed on only one machine. In many industry applications, this may not be an adequate model. Namely, a job may be processed simultaneously by several machines. For example, in semiconductor circuit design workforce planning, a design project is to be processed by m persons (a team of people). The project contains n tasks, and each task can be worked on by one of a set of alternatives, where each alternative contains one or more persons

Correspondence to: C.-Y. Lee
Contract grant sponsor: National Science Foundation; contract grant numbers: CCR-9613805 and DMI-9610229

 CCC 0894-069X/99/010057-18

working simultaneously on that particular task. Task 1 can either be handled by person 1 and person 2 together, by person 1 and person 3 together, or just handled by person 1 alone. The processing time of each task depends on the group being assigned to handle the task. A group is formed when a set of people is working on a particular task, but a person may not belong to a fixed group all the time. Our goal is to assign these n tasks to m persons in order to minimize the project finishing time. Other applications can be found in (i) the berth allocation problems, where a large vessel may occupy several berths for loading and/or unloading, (ii) diagnosable microprocessor systems (Krawczyk and Kubale [20]), where a job must be performed on parallel processors in order to detect faults, (iii) manufacturing, where a job may need machines, tools, and people simultaneously, and (iv) scheduling a sequence of meetings where each meeting requires a certain group of people (Dobson and Karmarker [13]). In all the above examples, one job may need to be processed by several machines simultaneously. In literature we call this *multiprocessor task scheduling* (Drozdowski [15]) or a *one-job-on-multiple-machine* problem (Lee and Cai [24]).

We are interested in the problem with a one-job-on-multiple-machine pattern. To describe the problem concisely we introduce the notation first.

There are n jobs to be processed on m machines, and

J_i : job i, $i = 1, \ldots, n$,
M_j : machine j, $j = 1, \ldots, m$,
C_i : completion time for J_i,
C_{max} : makespan $= \text{Max}\{C_i: i = 1, \ldots, n\}$,
$N(i)$: the number of alternative machine sets to which job J_i can be assigned,
$t_{i,I}$: the processing time of job J_i if it is assigned to the processors in set I, where I is a set of machines (for example, $t_{i,12}$ is the processing time of J_i assigned to Processors 1 and 2).

In this paper, *job* and *task* will be used interchangeably, and *machine* and *processor* will also be used interchangeably.

For example, we may have four jobs to be processed by three machines. The alternative machine sets that can process each job are shown in the following matrices. J_1 can be processed by one of six alternatives ($N(1) = 6$): $\{M_1,M_2,M_3\}$, $\{M_1,M_2\}$, $\{M_1,M_3\}$, $\{M_2,M_3\}$, $\{M_2\}$ or $\{M_3\}$, with corresponding processing times $t_{1,123} = 2$, $t_{1,12} = 3$, $t_{1,13} = 3$, $t_{1,23} = 3$, $t_{1,2} = 6$, and $t_{1,3} = 8$. Therefore, if J_1 is processed by the first alternative: $\{M_1,M_2,M_3\}$ (represented by the first column in the first matrix where an entry of 1 in row j means that M_j belongs to alternative 1), then the processing time is $t_{1,123} = 2$. Our purpose is to schedule jobs with a particular objective function.

$$J_1: \begin{bmatrix} M_1: & 1 & 1 & 1 & 0 & 0 & 0 \\ M_2: & 1 & 1 & 0 & 1 & 1 & 0 \\ M_3: & 1 & 0 & 1 & 1 & 0 & 1 \\ t_{1,I}: & 2 & 3 & 3 & 3 & 6 & 8 \end{bmatrix}, \quad J_2: \begin{bmatrix} M_1: & 1 & 0 & 1 & 0 \\ M_2: & 1 & 1 & 0 & 1 \\ M_3: & 1 & 1 & 0 & 0 \\ t_{2,I}: & 4 & 5 & 8 & 6 \end{bmatrix},$$

$$J_3: \begin{bmatrix} M_1: & 0 & 1 & 0 & 0 \\ M_2: & 1 & 0 & 1 & 0 \\ M_3: & 1 & 0 & 0 & 1 \\ t_{3,I}: & 5 & 8 & 8 & 8 \end{bmatrix}, \quad J_4: \begin{bmatrix} M_1: & 1 & 1 & 0 & 0 \\ M_2: & 1 & 0 & 1 & 0 \\ M_3: & 0 & 0 & 0 & 1 \\ t_{4,I}: & 2 & 4 & 4 & 4 \end{bmatrix}.$$

In general, for the m-machine problem, the maximum possible number of alternative sets for any job is $\binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{m}$. Actually, for notational convenience we can always assume that $N(i) = \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{m}$ for all i and let $t_{iI} = \infty$ for those I such that job i cannot be processed in parallel by processors with indices in I. In the above example, $t_{2,12}$, $t_{2,13}$, and $t_{2,3}$ are all equal to ∞. In this paper, we are particularly interested in $m = 2$ and 3. Hence for all i, we have $N(i) = 3$ and 7 for $m = 2$ and 3 respectively.

There are two special cases that have been studied in the scheduling literature. In the first special case, $N(i) = 1$ for all i. Namely, for each job a specifically *fixed* set of machines is assigned to it. We call this problem a *fixed* multiprocessor task scheduling problem. For example, if only $t_{1,1}$, $t_{2,23}$, $t_{3,23}$, and $t_{4,3}$ are finite numbers in the above example, then our problem becomes a fixed multiprocessor task scheduling problem. Krawczyk and Kubale [20], Kubale [21], Hoogeveen, van de Velde, and Veltman [18], Brucker [9], and Kramer [19] study the problem with the objective of minimizing the *makespan* under different machine configurations and constraints. Dobson and Karmarkar [13], Dobson and Khosla [14], Hoogeveen, van de Velde, and Veltman [18], Brucker [9], and Cai, Lee, and Li [12] study the problem with objective function related to *weighted mean flow time.* Bianco et al. [2], Brucker [9], Bianco et al. [3], and Lee, Lei, and Pinedo [26] examine the problem with an objective related to *lateness.* Most of the papers above study parallel-machine problems while Brucker and Kramer [10, 11] study other machine configurations such as flow shop, open shop and job shop.

The second class of problems assumes that each job may require a fixed *number* of processors working simultaneously, yet the machines required are *not specified.* In the example above, if only $t_{1,12}$, $t_{1,13}$, $t_{1,23}$, $t_{2,123}$, $t_{3,1}$, $t_{3,2}$, $t_{3,3}$, $t_{4,1}$, $t_{4,2}$, and $t_{4,3}$ are finite numbers, then it is equivalent to the problem where J_1 needs to be processed by two machines (any two machines simultaneously with processing time 3), J_2 by three machines simultaneously (with processing time 4) and jobs J_3 and J_4 by only one machine (with processing times 8 and 4, respectively). We call this problem a *sized* multiprocessor task scheduling problem. Blazewicz, Weglarz, and Drabowski [8], Blazewicz, Drabowski, and Weglarz [5], Du and Leung [16], and Blazewicz et al. [7] study the problem of minimizing *makespan.* Lee and Cai [24] study the problem with an objective of minimizing *weighted mean flow time.* Plehn [27] and Lee and Cai [24] study the problem with a *lateness-related* objective function.

For the general *set* multiprocessor task scheduling problem, Bianco et al. [1] provide a dynamic programming algorithm with time complexities $O(nT_0^3)$ for the $m = 2$ case, and $O(nT_0^4)$ for a special case of $m = 3$, where T_0 is a finite upper bound of the makespan which will be defined formally later. They also provide a heuristic for the general m-machine problem with error bound $C/C^* \leq m$. Please see Veltmann, Lageweg, and Lenstra [28], Blazewicz, Drozdowski, and Weglarz [6], Drozdowski [15], and Lee, Lei, and Pinedo [25] for detailed surveys.

To refer to the problem under study more precisely, we follow the standard notation used in scheduling literature. We use $Pm|\text{set}_j|C_{max}$ to denote the general problem of minimizing the makespan of multiprocessor tasks in the m-parallel-machine problem, where each job can be processed by a set of alternatives, and each alternative contains one or more machines simultaneously. Also, we use $Pm|\text{fix}_j|C_{max}$ to denote the first special case where the alternative assigned to each job has been fixed in advance.

The paper is organized in the following way. In Section 1, we study the problem characteristics and provide some optimality properties. Section 2 discusses the two-machine problem. In particular, we provide a pseudopolynomial algorithm with running time $O(nT_0)$ to solve the

problem optimally. Section 3 discusses the three-machine problem. We provide a pseudopolynomial algorithm with running time $O(nT_0^2)$ to find an effective lower bound for the optimal makespan and to solve optimally a special case of the problem. Both of our pseudopolynomial algorithms significantly improve previous results in the literature. We also provide a combination of a fully polynomial scheme and a heuristic method, with time complexity $O(n^3/\varepsilon^2)$ and error bound $(3/2)(1 + \varepsilon)$, to solve the general three-machine problem. Section 4 extends our results to the m-machine problem. Finally, Section 5 concludes with a summary and a discussion of some future research topics.

1. PROBLEM CHARACTERISTICS AND OPTIMALITY PROPERTIES

In solving $Pm|\text{set}_j|C_{max}$ problem, we need to assign a machine set from $N(i)$ alternatives for each job i, $i = 1, \ldots, n$. We also need to decide the schedule for jobs (i.e., the order in which the jobs are executed). We call these two decisions: ***assignment*** and ***schedule.*** Note that these two decisions can be made in parallel or in sequence. We can assign a machine set to each job first, and then make scheduling decision for the jobs. It is also possible to make the assignment and schedule decisions for a particular job before we consider other jobs. However, doing assignments for all jobs before proceeding with a schedule decision has the advantage of simplicity. We will follow this approach in this paper and call it ***two-phase method.*** Phase I is an assignment decision while Phase II is a scheduling decision.

Since we will not consider scheduling in making the assignment decision, it is important to choose the right objective function for the assignments. We will introduce an objective function for the assignment decision problem so that an optimal assignment in terms of this objective function provides an effective *lower bound* for the original $Pm|\text{set}_j|C_{max}$ problem.

For a given assignment A, we use $\boldsymbol{G_I(A)}$ to denote the set of jobs that are assigned to the parallel execution of all processors with index in the set I. For example, for a given assignment A, $\boldsymbol{G_1(A)}$ consists of the jobs that are assigned to Processor-1, $\boldsymbol{G_{12}(A)}$ consists of the jobs that are assigned to be processed by Processor-1 and Processor-2 in parallel, and $\boldsymbol{G_{123}(A)}$ consists of the jobs that are assigned to be processed by Processor-1, Processor-2, and Processor-3 in parallel. Define

$$T_j(A) = \sum_{I \in \Omega(j)} \sum_{J_i \in G_I(A)} t_{i,I},$$

where $\Omega(j)$ denotes the set of index sets that contain machine j. For example, in two-machine problem, $\Omega(1) = \{\{1\}, \{12\}\}$ and $\Omega(2) = \{\{2\}, \{12\}\}$. Given assignment A, $T_j(A)$ is the sum of processing times of all jobs for which processor j will participate in processing. In calculating $T_j(A)$ we haven't considered the schedule decision, and hence $T_j(A)$ *does not include any idle time that may be incurred when the schedule is implemented.*

For example, for the two-machine problem,

$$T_1(A) = \sum_{J_i \in G_1(A)} t_{i,1} + \sum_{J_i \in G_{12}(A)} t_{i,12},$$

$$T_2(A) = \sum_{J_i \in G_2(A)} t_{i,2} + \sum_{J_i \in G_{12}(A)} t_{i,12}.$$

For the three-machine problem,

$$T_1(A) = \sum_{J_i \in G_1(A)} t_{i,1} + \sum_{J_i \in G_{12}(A)} t_{i,12} + \sum_{J_i \in G_{13}(A)} t_{i,13} + \sum_{J_i \in G_{123}(A)} t_{i,123},$$

and $T_2(A)$ and $T_3(A)$ are defined similarly.

Let

$$T_A = \max_{j=1,\ldots,m} Tj(A).$$

Our assignment problem is to find an assignment A such that T_A is minimized. Namely,

$$\text{(PA)} \quad \min_A \left\{ T_A = \max_{j=1,\ldots,m} \left\{ \sum_{I \in \Omega(j)} \sum_{J_i \in G_I(A)} t_{i,I} \right\} \right\}.$$

We call the above assignment problem Problem PA. In Problem PA, we are interested in finding an assignment A to minimize T_A. When A^* is an optimal solution to Problem PA, then clearly T_{A^*} is a *lower bound* of the original problem $Pm|set_j|C_{max}$. Namely, $T_{A^*} \leq T^*$, where T^* is the optimal value of the original problem. Although T_{A^*} may not be the optimal objective value for the original problem, we show that it is an effective lower bound.

In this paper, when we say *optimal assignment,* we mean *optimal* solution to problem PA, i.e., it may not be optimal to the original problem. In some special cases in which we can find a schedule, based on a given assignment, that results in no idle time in each machine, the lower bound we obtain from the assignment decision will also be an optimal value for our original problem. Examples of these special cases include the *two-machine problem and a special case of three-machine problem.*

In the two-machine problem, T_{A^*} is actually an *optimal* value for the original problem, $P2|set_j|C_{max}$, as the following schedule will yield a feasible schedule with makespan T_{A^*}:

Scheduling Decision for the Two-Machine Problem: "Given the assignment A^*, first schedule all the jobs in $G_{12}(A^*)$ on Processor-1 and Processor-2 for parallel execution and then schedule all jobs in $G_1(A^*)$ on Processor-1 and all jobs in $G_2(A^*)$ on Processor-2."

Under this schedule, there is no idle time between jobs on each machine. Hence the completion time will be equal to T_{A^*}. Thus, it is an optimal solution to the original problem $P2|set_j|C_{max}$. As we will show in Section 2, a pseudopolynomial algorithm can optimally solve the assignment decision. Hence we can get the optimal solution to the original problem in pseudopolynomial time.

For the m-machine ($m > 2$) problem, the situation is much more complicated, except for a special case of the three-machine problem, as we will show later. The problem has been proven to be NP-hard in the strong sense even for the special case $P3|fixed_j|C_{max}$. Namely, it is impossible to solve the original problem optimally by a pseudopolynomial algorithm (Garey and Johnson [17], p. 95). Even if we solve the assignment problem optimally, the optimal value of the assignment is only a lower bound to the original problem. Furthermore, the schedule decision is based on the given assignment we obtained in the assignment decision, hence we cannot determine whether we have identified an optimal solution to the original problem if the

final makespan of the schedule is greater than T_{A^*}. Nevertheless, T_{A^*} does provide an effective lower bound. In Section 3, we will provide a polynomial heuristic, based on T_{A^*}, with time complexity $O(n^3/\varepsilon^2)$ and error bound of $(1 + \varepsilon)(3/2)$.

2. TWO-MACHINE PROBLEM: P2|SET$_J$|C_{MAX}

It is clear that the P2|set$_j$|C_{max} problem is NP-hard because a special case of it, the classical two-parallel-machine problem P2$\|C_{max}$, is already NP-hard. As we discussed above, once we solve Problem PA optimally, we can then easily find an optimal schedule for the original problem. Hence in this section, we will focus on solving problem PA for the two-machine problem. Note that there are at most three alternatives for each job [$N(i) \leq 3$ for all i]. Hence we assume that each job J_i is associated with a triple $(t_{i,1}, t_{i,2}, t_{i,12})$, where $t_{i,I}$ was defined above. We will let the processing time of a particular alternative be infinite if there is no such alternative for that job. The P2|set$_j$|C_{max} problem is to schedule the jobs so that the makespan is minimized.

Let $t_{min}(i) = \min\{t_{i,1}, t_{i,2}, t_{i,12}\}$. We assume that $t_{min}(i) < \infty$ for all i. Define

$$T_0 = \sum_{i=1}^{n} t_{min}(i).$$

Bianco et al. [1] provide a pseudopolynomial dynamic programming algorithm with time complexity $O(nT_0^3)$ to solve the problem. In this section, we provide a much more efficient algorithm with time complexity $O(nT_0)$ to solve the problem optimally.

LEMMA 1: Let T^{opt} be the makespan for an optimal schedule for the jobs $J_1, \ldots, J_n$ in the P2|set$_j$|C_{max} problem. Then $T_0/2 \leq T^{opt} \leq T_0$.

PROOF: Consider the following assignment A: for $i = 1, \ldots, n$, assign job J_i to the alternative with processing time $t_{min}(i)$. It is obvious that the assignment will have a makespan $T \leq T_0$, and hence $T^{opt} \leq T \leq T_0$. For each job J_i, the minimum processing time is at least $t_{min}(i)$ on at least one machine; any processor assigned to J_i has to spend time at least $t_{min}(i)$ in order to process job J_i. Therefore, the minimum total sum of processing times for any assignment to two machines is at least T_0. Thus, a lower bound of the optimal makespan is $T_0/2$, which is obtained by evenly distributing T_0 to two machines. Hence $T^{opt} \geq T_0/2$. □

Now we are ready to provide a pseudo-polynomial algorithm to solve our problem optimally. The following dynamic programming algorithm aims to partition the jobs $J_1, \ldots, J_n$ into three groups $G_1(A)$, $G_2(A)$, $G_{12}(A)$ such that the value T_A is minimized.

Dynamic Programming Algorithm (DP1). Define the value of the function $f(i,x)$ for $i \geq 1$, $0 \leq x < \infty$ to be the minimum y such that there is an assignment S for jobs $J_1, \ldots, J_i$, for which the total running time on Processor-1 is x, and the total running time on Processor-2 is y, where we assume that x and y are finite numbers. If there does not exist an assignment for jobs $J_1, \ldots, J_i$ such that the total running time on Processor-1 is x and the total running time on Processor-2 is finite, then we define $f(i, x) = \infty$.

Suppose $f(i,x) = y$. There are three possible ways to assign job i to achieve $f(i,x) = y$: (1) Assign job i to Processor-1, (2) assign it to Processor-2, or (3) assign it to parallel execution on

Processor-1 and Processor-2. Suppose $f(i,x) = y$ is achieved by assigning job i to Processor-1. Since this assignment increases the running time of Processor-1 by t_{i1} and does not increase the running time of Processor-2, we must have $f(i - 1, x - t_{i1}) = y$. Similar analysis can be applied for the other two possible assignments. Hence by the principle of optimality, we know that

$$f(i,x) = \min\{f(i - 1, x - t_{i,1}), f(i - 1, x) + t_{i,2}, f(i - 1, x - t_{i,12}) + t_{i,12}\}$$

Now we are ready to develop recursive equations to solve the problem.

Initial conditions:

$$f(1,0) = t_{1,2},$$

$$f(1,t_{1,1}) = 0,$$

$$f(1,t_{1,12}) = t_{1,12},$$

$$f(1,x) = \infty \qquad \text{for all other } x.$$

Recursive equations:

$$f(i,x) = \min\begin{cases} f(i - 1, x - t_{i,1}), \\ f(i - 1, x) + t_{i,2}, \\ f(i - 1, x - t_{i,12}) + t_{i,12}, \end{cases}$$

where $i = 2, \ldots, n$, and $x = 0, \ldots, T_0$.

THEOREM 1: The optimal solution of P2|set$_j$|C_{max} is equal to $\min\{\max(x, f(n,x)): x = 0, \ldots, T_0\}$, and it can be found in time complexity $O(nT_0)$.

PROOF: Since $t_{min}(i) < \infty$ for all i, by Lemma 1 there must exist an optimal assignment for $J_1, \ldots, J_n$ such that the total running time on Processor-1 is x_0, the total running time on Processor-2 is y_0, $x_0 \le T_0$, $y_0 \le T_0$, and the corresponding completion time is $\max\{x_0, y_0\}$.

Note that y_0 is a feasible solution given that the total processing time on Processor-1 is x_0. We have $f(n,x_0) \le y_0$ by the definition of $f(n,x_0)$. Hence $\min\{\max(x, f(n,x)): x = 0, \ldots, T_0\} \le \max\{x_0, f(n,x_0)\} \le \max(x_0, y_0)$.

On the other hand, let x_1 be the value such that $\max\{x_1, f(n, x_1)\} = \min\{\max(x, f(n,x)): x = 0, \ldots, T_0\}$. It means that there exists a feasible solution such that the total running time on Processor-1 is x_1 and the total running time on Processor-2 is $f(n, x_1)$. Clearly, it is a feasible solution to P2|set$_j$|C_{max}. Hence $\min\{\max(x,f(n,x)): x = 0, \ldots, T_0\} = \max\{x_1, f(n, x_1)\} \ge \max\{x_0, y_0\}$.

By the recursive equation, it is easy to see that the optimal solution can be found in $O(nT_0)$ time and the actual assignment corresponding to this $f(n, x)$ can be constructed by backtracking from $f(n, x)$ in time $O(n)$. □

Now we can summarize our approach for solving P2|set$_j$|C_{max} optimally. First, apply the Dynamic Programming Algorithm (DP1) described above to solve Problem PA, denoting the optimal assignment as A^* and the corresponding objective value as T_{A^*}. Second, schedule all jobs in $G_{12}(A^*)$ on Processor-1 and Processor-2 for parallel execution and then schedule all jobs in $G_1(A^*)$ to Processor-1 and all jobs in $G_2(A^*)$ to Processor-2. The optimal objective value of P2|set$_j$|C_{max} is equal to T_{A^*} found by solving PA.

REMARK 1: We can develop a fully polynomial approximation scheme, similar to the one in the next section for the three-machine problem, to solve the problem P2|set$_j$|C_{max}.

3. THREE-MACHINE PROBLEM: P3|SET$_J$|C_{MAX}

As mentioned above, the three-machine problem is more complicated than the two-machine problem. It has been proven that the more restricted problem P3|fix$_j$|C_{max}, in which the assignment is fixed, is strongly NP-hard (see Blazewicz et al. [4] and Hoogeveen et al. [18]). Hence, our problem is also NP-hard in the strong sense. It is impossible to find a pseudopolynomial algorithm to solve the three-machine problem *optimally* (Garey and Johnson [17], p. 95). Hence our goal is to provide a polynomial heuristic approach to solve the problem and also provide an error bound analysis. Similarly to our analysis for the two-machine problem, we solve the problem by first deciding the assignment and then the schedule. In the assignment decision, a pseudopolynomial algorithm will be provided to find an optimal solution to problem PA, which will serve as a *lower* bound to our original problem. The assignment decision for the three-machine problem is similar to that of the two-machine problem. However, the scheduling decision for the three-machine problem is much more complicated than that for the two-machine problem.

In the following subsections, we will first provide a pseudo-polynomial algorithm to find an optimal assignment for Problem PA. Based on this approach, we then provide a fully polynomial approximation scheme for the Problem PA. The purpose of the fully polynomial approximation scheme is to combine it with a polynomial heuristic in the schedule decision and to solve the whole problem in polynomial time. Error bound analysis for the heuristic will be provided.

3.1. Pseudopolynomial Dynamic Programming Algorithm: Optimal Assignment Decision

Since there are at most seven possible alternatives for each job, we let $J_i = (t_{i,1}, t_{i,2}, t_{i,3}, t_{i,12}, t_{i,13}, t_{i,23}, t_{i,123})$, where t_{iI} was defined above. In this subsection, we will solve Problem PA for the three-machine problem.

$$\text{(PA)} \qquad \min_A \left\{ T_A = \max_{j=1,2,3} \left\{ \sum_{I \in \Omega(j)} \sum_{J_i \in G_I(A)} t_{i,I} \right\} \right\}.$$

The optimal value of problem PA, denoted as T_{A^*}, will serve as a *lower bound* for our original problem. Let $t_{min}(i) = \min\{t_{i,1}, t_{i,2}, t_{i,3}, t_{i,12}, t_{i,13}, t_{i,23}, t_{i,123}\}$. Furthermore, we define $T_0 = \sum_{i=1}^{n} t_{min}(i)$.

LEMMA 2: $T_0/3 \le T_{A^*} \le T_0$.

PROOF: Similar to that of Lemma 1. Omitted.

The following dynamic programming algorithm aims to find an optimal assignment to obtain a *lower bound*, T_{A*}, for the P3|set$_j$|C_{max} problem.

Dynamic Programming Algorithm (DP2). Define the function $f(i, x, y)$ as the minimum value of the total processing time assigned to Processor-3 (without counting possible idle time), given that there exists an assignment for $J_1, \ldots, J_i$ such that the total processing time is x for Processor-1 and y for Processor-2. The value $f(i,x,y)$ equals ∞ if there is no such assignment. Similar to Theorem 1, we can develop recursive equations to get the optimal assignment solution. There are seven alternatives in the recursive equations, as shown below.

Initial condition:

$$f(1,t_{1,1},0) = 0,$$

$$f(1,\ t_{1,12},\ t_{1,12}) = 0,$$

$$f(1,t_{1,13},0) = \begin{cases} t_{1,13}, & \text{if } t_{1,13} \neq t_{1,1}, \\ 0, & \text{otherwise,} \end{cases}$$

$$f(1,\ 0,\ t_{1,2}) = 0,$$

$$f(1,0,t_{1,23}) = \begin{cases} t_{1,23}, & \text{if } t_{1,23} \neq t_{1,2}, \\ 0, & \text{otherwise,} \end{cases}$$

$$f(1,0,0) = t_{1,3}$$

$$f(1,t_{1,123},t_{1,123}) = \begin{cases} t_{1,123} & \text{if } t_{1,123} \neq t_{1,12}, \\ 0, & \text{otherwise,} \end{cases}$$

$$f(1,x,y) = \infty \qquad \text{for all other } x \text{ and } y.$$

Recursive equations:

$$f(i,\ x,\ y) = \min \begin{cases} f(i-1,\ x-t_{i,1},\ y), \\ f(i-1,\ x,\ y-t_{i,2}), \\ f(i-1,\ x,\ y)+t_{i,3}, \\ f(i-1,\ x-t_{i,12},\ y-t_{i,12}), \\ f(i-1,\ x-t_{i,13},\ y)+t_{i,13}, \\ f(i-1,\ x,\ y-t_{i,23})+t_{i,23}, \\ f(i-1,\ x-t_{i,123},\ y-t_{i,123})+t_{i,123}, \end{cases}$$

where $i = 2, \ldots, n$, $x = 0, \ldots, T_0$, and $y = 0, \ldots, T_0$.

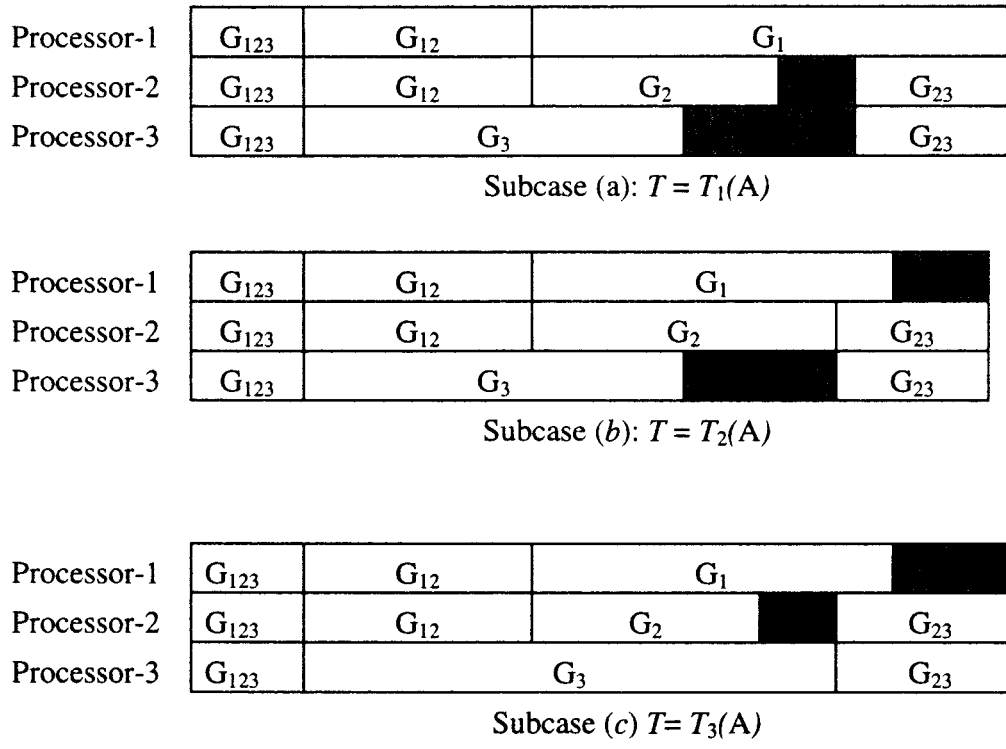

Figure 1. Detailed schedules for special P3|set$_j$|C_{max}: subcase (a): $T = T_1(A)$; subcase (b): $T = T_2(A)$; subcase (c): $T = T_3(A)$.

Optimal Solution: $\min\{\max(f(n,x,y), x, y): x = 0, \ldots, T_0, y = 0, \ldots, T_0\}$.

Time Complexity: $O(nT_0^2)$. Note that the actual assignment corresponding to this $f(n, x, y)$ can be constructed by backtracking from $f(n, x, y)$ in time $O(n)$.

Justification: There are seven possible allocations for each job J_i. Since in the optimum value T_{A*} for the assignment problem, we only care about the total processing time for each processor (without counting the possible idle time), these seven cases correspond to the seven terms in the recursive equation.

REMARK 2: It is interesting to note that the above pseudopolynomial dynamic programming algorithm can also be used to solve the special case studied by Bianco et al. [1]. They provide a pseudopolynomial algorithm with time complexity $O(nT_0^4)$ to solve a special case of the problem P3|set$_j$|C_{max} where at least one of $G_{12}(A)$, $G_{13}(A)$, and $G_{23}(A)$ is empty for any assignment (or has infinite processing time for each job in that set). They analyze 26 cases and use a dynamic programming algorithm to solve the problem. Without loss of generality, we assume that $G_{13}(A) = \varnothing$ for any A. For notational convenience, we denote such a problem as ***Special P3|set$_j$|C_{max}.*** As indicated in Bianco et al. [1], any schedule to such a problem can always be reduced to one of the schedules shown in Figure 1. In the following we show that DP2, which is much more efficient [with time complexity $O(nT_0^2)$] and is easier to understand, can be applied to solve such a special case.

LEMMA 3: DP2 can be applied to solve *Special* P3|set$_j$|C_{max} in $O(nT_0^2)$ time.

PROOF: As discussed above, T_{A^*} is a lower bound for T^*. Hence it suffices to show that given the solution of T_{A^*}, we can always construct a feasible solution to the original *Special* P3|set$_j$|C_{max} problem with $T_{A^*} = T^*$. Let A^* be the optimal assignment constructed by Algorithm DP2. Depending on whether T_{A^*} equals $T_1(A^*)$, $T_2(A^*)$, or $T_3(A^*)$, we construct a schedule illustrated in Figure 1 as subcases (a), (b), and (c), respectively. It is easy to verify that, in each case, the schedule has completion time $= T_{A^*}$. Namely, we have a feasible schedule with an objective value equal to a lower bound of the optimal value. Clearly it is an optimal solution. □

So far, we have shown how to provide a pseudopolynomial time algorithm with running time $O(nT_0^2)$ for solving Problem PA. In the next subsection, we show how this pseudopolynomial time algorithm provides the basis for a fully polynomial approximation scheme for solving Problem PA. This fully polynomial approximation scheme plus a polynomial heuristic for the schedule decision will yield a polynomial heuristic for the whole problem.

3.2. Fully Polynomial Scheme for the Assignment Decision

Let $\varepsilon > 0$ be any constant. Let $K = (\varepsilon T_0)/(3n)$. Define a new job set

$$J' = \{J'_i = (t'_{i,1}, t'_{i,2}, t'_{i,3}, t'_{i,12}, t'_{i,13}, t'_{i,23}, t'_{i,123}): i = 1, \ldots, n\},$$

where $t'_{i,j} = \lceil t_{i,j}/K \rceil$, $t'_{i,jk} = \lceil t_{i,jk}/K \rceil$, $t'_{i,123} = \lceil t_{i,123}/K \rceil$, and $\lceil x \rceil$ denotes the smallest integer that is not less than x. Also, let

$$\begin{aligned} t'_{min}(i) &= \min\{t'_{i,1}, t'_{i,2}, t'_{i,3}, t'_{i,12}, t'_{i,13}, t'_{i,23}, t'_{i,123}\} \\ &= \min\{\lceil t_{i,1}/K \rceil, \lceil t_{i,2}/K \rceil, \lceil t_{i,3}/K \rceil, \lceil t_{i,12}/K \rceil, \lceil t_{i,13}/K \rceil, \lceil t_{i,23}/K \rceil, \lceil t_{i,123}/K \rceil\} \\ &= \lceil t_{min}(i)/K \rceil \end{aligned}$$

and $T'_0 = \sum_{i=1}^{n} t'_{min}(i)$.

According to the previous discussion, the pseudopolynomial time algorithm DP2 constructs an optimal solution A^{apx} for the job set J' in Problem PA. Keeping the same job-processor assignments, A^{apx} is also an assignment for the original job set J. We use A^{apx} as an approximation to the optimal assignment for the original job set J.

THEOREM 2: Let $T_{A^{apx}}$ be the value for the assignment A^{apx} and T_{A^*} be the value for an optimal assignment A^* for the assignment problem PA, both for the original job set J. Then (i) the assignment A^{apx} can be constructed in time $O(n^3/\varepsilon^2)$ and (ii) $T_{A^{apx}}/T_{A^*} \leq 1 + \varepsilon$.

PROOF: The assignment A^{apx} can be constructed from the job set J', using the algorithm DP2 that runs in time $O(n(T'_0)^2))$, where

$$T'_0 = \sum_{i=1}^{n} t'_{min} = \sum_{i=1}^{n} \lceil t_{min}(i)/K \rceil$$

$$\leq \sum_{i=1}^{n} (t_{min}(i)/K + 1)$$

$$= \frac{1}{K} \sum_{i=1}^{n} t_{min}(i) + n$$

$$= \frac{T_0}{K} + n$$

$$= \frac{n(3 + \varepsilon)}{\varepsilon}.$$

Therefore, the assignment A^{apx} can be constructed in time $O(n(T_0')^2) = O(n^3/\varepsilon^2)$, which is $O(n^3)$ when ε is a fixed constant.

Now we analyze the performance ratio for the assignment A^{apx} for jobs $J_1, \ldots, J_n$. For the assignment A^{apx}, let $t_{i,1}^{apx}$, $t_{i,2}^{apx}$ and $t_{i,3}^{apx}$ be the processing times of job J_i assigned to Processor-1, Processor-2, and Processor-3, respectively, by A^{apx}. If A^{apx} assigns job J_i to Processor-1 alone, then $t_{i,1}^{apx} = t_{i,1}$ and $t_{i,2}^{apx} = t_{i,3}^{apx} = 0$; if A^{apx} assigns job J_i to Processor-2 alone, then $t_{i,1}^{apx} = t_{i,3}^{apx} = 0$ and $t_{i,2}^{apx} = t_{i,2}$; if A^{apx} assigns job J_i to Processor-1 and Processor-2 for parallel execution, then $t_{i,1}^{apx} = t_{i,12}$, $t_{i,2}^{apx} = t_{i,12}$, and $t_{i,3}^{apx} = 0, \ldots$, etc. Similarly, let $t_{i,1}^{*}$, $t_{i,2}^{*}$, and $t_{i,3}^{*}$ be the processing time of job J_i assigned to Processor-1, Processor-2, and Processor-3, respectively, by A^*. The value $T_{A^{apx}}$ for the schedule A^{apx} is

$$T_{A^{apx}} = \max\left\{ \sum_{i=1}^{n} t_{i,1}^{apx}, \sum_{i=1}^{n} t_{i,2}^{apx}, \sum_{i=1}^{n} t_{i,3}^{apx} \right\},$$

and the value T_{A^*} for the schedule A^* is

$$T_{A^*} = \max\left\{ \sum_{i=1}^{n} t_{i,1}^{*}, \sum_{i=1}^{n} t_{i,2}^{*}, \sum_{i=1}^{n} t_{i,3}^{*} \right\}.$$

We have

$$T_{A^*} = \max\left\{ \sum_{i=1}^{n} t_{i,1}^{*}, \sum_{i=1}^{n} t_{i,2}^{*}, \sum_{i=1}^{n} t_{i,3}^{*} \right\}$$

$$= \max\left\{ K \sum_{i=1}^{n} t_{i,1}^{*}/K, K \sum_{i=1}^{n} t_{i,2}^{*}/K, K \sum_{i=1}^{n} t_{i,3}^{*}/K \right\}$$

$$\geq \max\left\{ K \sum_{i=1}^{n} (\lceil t_{i,1}^{*}/K \rceil - 1), K \sum_{i=1}^{n} (\lceil t_{i,2}^{*}/K \rceil - 1), K \sum_{i=1}^{n} (\lceil t_{i,3}^{*}/K \rceil - 1) \right\}$$

$$= \max\left\{K \sum_{i=1}^{n} \lceil t^*_{i,1}/K \rceil,\ K \sum_{i=1}^{n} \lceil t^*_{i,2}/K \rceil,\ K \sum_{i=1}^{n} \lceil t^*_{i,3}/K \rceil\right\} - Kn$$

$$\geq \max\left\{K \sum_{i=1}^{n} \lceil t^{apx}_{i,1}/K \rceil,\ K \sum_{i=1}^{n} \lceil t^{apx}_{i,2}/K \rceil,\ K \sum_{i=1}^{n} \lceil t^{apx}_{i,3}/K \rceil\right\} - Kn.$$

This last inequality holds due to the fact that the schedule A^{apx} is an optimal assignment for the job set J' and the value $\max\{\sum_{i=1}^{n} \lceil t^{apx}_{i,1}/K \rceil, \sum_{i=1}^{n} \lceil t^{apx}_{i,2}/K \rceil, \sum_{i=1}^{n} \lceil t^{apx}_{i,3}/K \rceil\}$ is an optimal value for the job set J'.

Note that

$$K \max\left\{\sum_{i=1}^{n} \lceil t^{apx}_{i,1}/K \rceil,\ \sum_{i=1}^{n} \lceil t^{apx}_{i,2}/K \rceil,\ \sum_{i=1}^{n} \lceil t^{apx}_{i,3}/K \rceil\right\}$$

$$\geq K \max\left\{\sum_{i=1}^{n} t^{apx}_{i,1}/K,\ \sum_{i=1}^{n} t^{apx}_{i,2}/K,\ \sum_{i=1}^{n} t^{apx}_{i,3}/K\right\}$$

$$= \max\left\{\sum_{i=1}^{n} t^{apx}_{i,1},\ \sum_{i=1}^{n} t^{apx}_{i,2},\ \sum_{i=1}^{n} t^{apx}_{i,3}\right\}$$

$$= T_{A^{apx}}.$$

Thus we obtain

$$T_{A^*} + Kn \geq T_{A^{apx}} \text{ or } 1 + \frac{Kn}{T_{A^*}} \geq \frac{T_{A^{apx}}}{T_{A^*}}.$$

Note that A^* is an optimal assignment for the job set J. According to Lemma 2, $T_{A^*} \geq T_0/3$. Moreover, recall that $K = (\varepsilon T_0)/(3n)$. We obtain $(Kn)/T_{A^*} \leq \varepsilon$. This gives $T_{A^{apx}}/T_{A^*} \leq 1 + \varepsilon$. □

That is, the assignment A^{apx} is an ε-approximation to the optimal assignment for the job set J.

3.3. Approximating P3|set$_j$|C_{max}: Schedule Jobs

Given a job set $J = \{J_i = (t_{i,1}, t_{i,2}, t_{i,3}, t_{i,12}, t_{i,13}, t_{i,23}, t_{i,123}): i = 1, \ldots, n\}$, we first apply the $O(n^3/\varepsilon^2)$-time approximation algorithm described in the previous subsection to obtain an assignment A such that $T_A \leq T_{A^*}(1 + \varepsilon) \leq T^*(1 + \varepsilon)$, where T^* is the minimum makespan for the problem P3|set$_j$|C_{max} and $T_{A^*} \leq T^*$. The assignment A partitions the jobs into seven groups $G_1(A)$, $G_2(A)$, $G_3(A)$, $G_{12}(A)$, $G_{13}(A)$, $G_{23}(A)$, $G_{123}(A)$, where, as defined above, $G_I(A)$ consists of the jobs that are assigned by A to the parallel execution of all processors indexed in I. For notational convenience, we will delete A in $G_I(A)$ in this

subsection. Our next task is to schedule jobs on machines such that the makespan is as small as possible.

For each job group G_j, we let $T(G_j)$ be the total processing time of the system on the jobs in G_j. That is, $T(G_j) = \sum_{J_i \in G_j} t_{i,j}$ for $j = 1,2,3$. We define the notation $T(G_{jk})$ and $T(G_{123})$ similarly. Without loss of generality, we assume that $T(G_1) \geq T(G_2) \geq T(G_3)$ (otherwise, we simply reindex the processors). Based on assignment A, we apply a polynomial heuristic to schedule the jobs, and call the schedule S, which keeps the job-processor assignment of A but also schedules the jobs which may lead to idle times between jobs. Let T be the completion time of the final schedule. Although we cannot directly compare T to the optimal solution T^*, we are able to show that $T \leq (3/2)(1 + \varepsilon)T_{A^*}$, which implies that $T \leq (3/2)(1 + \varepsilon)T^*$.

Heuristic Schedule Algorithm 1:

1. Starting from time 0, execute the jobs in G_{123} on the three processors. Suppose that this process finishes at time s_{123}.
2. Starting from time s_{123}, process the jobs in G_{12} on Processor-1 and Processor-2. Suppose that this process finishes at time s_{12}.
3. Starting from time s_{12}, process the jobs in G_{13} on Processor-1 and Processor-3. Suppose that this process finishes at time s_{13}.
4. Starting from time s_{13}, process the jobs in G_{23} on Processor-2 and Processor-3. Suppose that this process finishes at time s_{23}.
5. Starting from time s_{13}, process the jobs in G_1 on Processor-1. Starting from time s_{23} process the jobs in G_2 on Processor-2 and process the jobs in G_3 on Processor-3.

THEOREM 3: Let T be the completion time for the schedule from the Heuristic Schedule Algorithm 1 and T^* be the optimal completion time for the original problem P3|set$_j$|C_{max}. Then $T \leq (3/2)(1 + \varepsilon)T^*$.

PROOF: Since the Heuristic Scheduling Algorithm 1 keeps the same assignment as that of A, and we have $T_A \leq T_{A^*}(1 + \varepsilon) \leq T^*(1 + \varepsilon)$, it suffices to show that $T \leq (3/2)T_A$. By the definition in the Heuristic Schedule Algorithm, $T = \max\{s_1, s_2, s_3\}$, where s_j is the completion time of the last job on machine j. Note that by our assumption, $T(G_1) \geq T(G_2) \geq T(G_3)$.

CASE i: $s_1 \geq s_2$. Hence $T = s_1$. Clearly $T = T_A$ [Fig. 2(a)] since $T = T_1(A)$ and Processor-1 has no idle time.

CASE ii: $s_1 < s_2$. Hence $T = s_2$ [Fig. 2(b)]. In this case, the sum of total processing times on the three machines equals

$$T_{total} = 3s_{123} + 2(s_{12} - s_{123}) + 2(s_{13} - s_{12}) + 2(s_{23} - s_{13}) + (s_1 - s_{13}) + (s_2 - s_{23}) + (s_3 - s_{23})$$

$$\geq 2s_{23} + 2(s_2 - s_{25}) \qquad \text{because } (s_1 - s_{13}) = T(G_1) \geq T(G_2) = (s_2 - s_{23})$$

$$= 2s_2 = 2T.$$

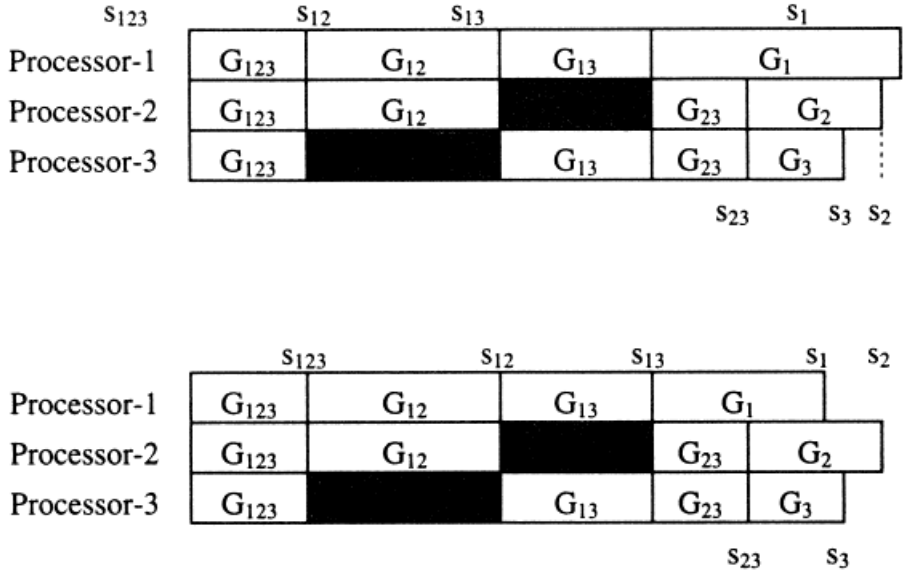

Figure 2. Schedule for P3|set$_j$|C_{max} with (a) $s_1 \geq s_2$ and (b) $s_1 < s_2$.

Since there are three processors, we have $T_A \geq T_{total}/3 \geq (2/3)T$. That is $T \leq (3/2)T_A$. This completes the analysis for the algorithm since, by our assumption $T(G_2) \geq T(G_3)$, we always have $s_2 \geq s_3$. □

REMARK 3: We point out that the ratio $T/T^* \leq (3/2)(1 + \varepsilon)$ is tight, based on our Heuristic Schedule Algorithm 1. To prove this, we show that there exists an instance for which the ratio T/T^* is arbitrarily close to 3/2. Consider a four-job three-machine problem with the following data. With the notation $J_i = \{t_{i,1}, t_{i,2}, t_{i,3}, t_{i,12}, t_{i,13}, t_{i,23}, t_{i,123}\}$, we have $t_{1,1} = 2 + 2\delta$, $t_{1,12} = 1$, $t_{2,2} = 2 + 2\delta$, $t_{2,23} = 1$, $t_{3,1} = 1 + \delta$, $t_{3,3} = 1$, $t_{4,3} = 1 + \delta$, and all other $t_{i,j}$, $t_{i,kl}$ as well as $t_{i,123}$ are ∞, where δ is a very small number. It is easy to verify that the assignment A^* for this instance has $T_{A^*} = 2 + \delta$ and for any other assignment A we have $T_A \geq 2 + 2\delta$. Therefore, if we let $\varepsilon < \delta/(2 + \delta)$, then an ε-approximation to the assignment A^* must be an optimal. Thus, if we apply the ε-approximation scheme to the assignment problem, we will get $G_{12} = \{J_1\}$, $G_{23} = \{J_2\}$, $G_1 = \{J_3\}$, and $G_3 = \{J_4\}$ with $T_{A^*} = 2 + \delta$. We then apply the Heuristic Schedule Algorithm 1 and get the schedule shown in Figure 3a with makespan $T = 3 + \delta$. However, the optimal solution of our problem is the schedule shown in Figure 3b with makespan $T^* = 2 + \delta$. Thus the ratio T/T^* approaches 3/2 since δ can be arbitrarily small.

REMARK 4: In Step 5 of Heuristic Schedule Algorithm 1, if there is an empty slot before s_{23} that is long enough for a job in G_2, then instead of starting from s_{23}, we will schedule that job to start in that available slot. Similarly, we will schedule jobs in G_3 into any available slot before s_{23}. Such a schedule will certainly result in a better makespan although it cannot improve the worst case error bound.

In summary, we have provided a combination of a fully polynomial approximation scheme with a polynomial heuristic schedule which is of running time $O(n^3/\varepsilon^2)$ and performance ratio $T/T^* \leq (3/2)(1 + \varepsilon)$ for the P3|set$_j$|C_{max} problem.

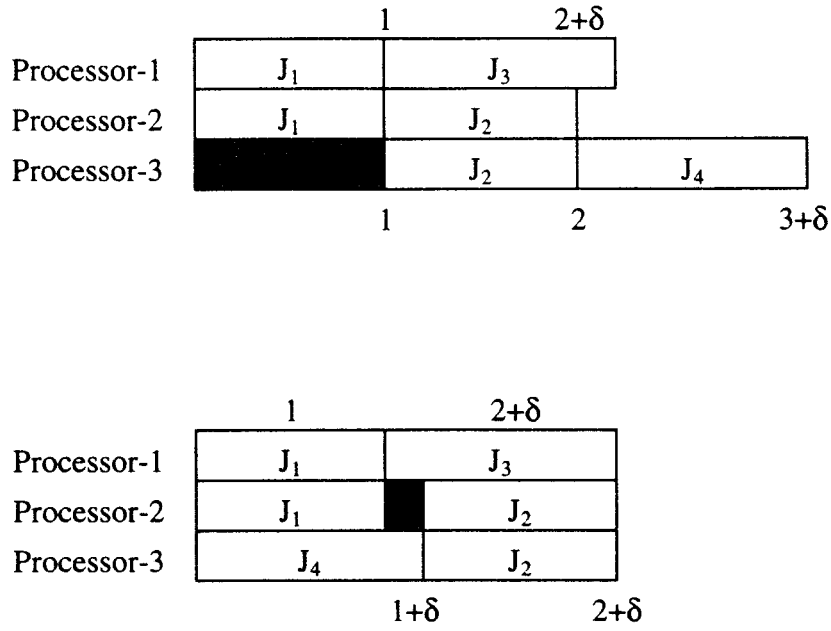

Figure 3. (a) Schedule obtained from the Schedule Algorithm; (b) optimal schedule.

4. *M*-MACHINE PROBLEM: P*M*|SET$_J$|C_{MAX}

For each fixed integer $m \geq 4$, we can extend the dynamic programming algorithm (DP2) to find a lower bound for the m-machine problem and then use the approximation scheme to find a solution within a $(1 + \varepsilon)$ error bound for the assignment. Let R be the maximum number of alternatives to which a job can be assigned, $R \leq \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{m}$. Since m is fixed, R is a fixed number too. Given that we have applied a fully polynomial scheme to obtain an assignment A such that $T_A \leq T_{A^*}(1 + \varepsilon) \leq T^*(1 + \varepsilon)$, where A^* is an optimal solution to the assignment decision problem PA, and T^* is the minimum makespan for the problem P$m|\text{set}_j|C_{max}$, we then apply the following polynomial heuristic to schedule those jobs, assuming that $T(G_1) \geq T(G_i)$ for all i.

Heuristic Schedule Algorithm 2:

1. Given a solution from the assignment step, divide jobs into following three sets.
   S_1: those jobs that need more than one machine to process and one of them must be processor-1.
   S_2: those jobs that need more than one machine to process and none of them is processor-1.
   S_3: jobs that require only one machine. Namely, $S_3 = G_1 \cup G_2 \cup \cdots \cup G_m$.
2. Schedule jobs in S_1 on the corresponding machines at the earliest machine available time and then jobs in S_2 and finally jobs in S_3 at the earliest machine available time.

THEOREM 4: Let T be the completion time for the final schedule of the Heuristic Schedule Algorithm 2 and T^* be the optimal completion time for the original problem $Pm|set_j|C_{max}$. Then $T \leq (m/2)(1 + \varepsilon)T^*$.

PROOF: Since the Heuristic Scheduling Algorithm 2 keeps the same assignment as that of A, and we have $T_A \leq T_{A^*}(1 + \varepsilon) \leq T^*(1 + \varepsilon)$, it suffices to show that $T \leq (m/2)T_A$. Let s_j be the completion time for the last job on machine j, then $T = \max\{s_1, s_2, \ldots, s_m\}$.

Note that Step 2 implies that there is no idle time on machine 1. Hence $s_1 \leq T_A$. If $T = s_1$, then $T = s_1 \leq T_A \leq T_{A^*}(1 + \varepsilon) \leq T^*(1 + \varepsilon)$, and we are done. Now suppose that $T = s_j > s_1$. Since each job in S_1 and S_2 needs more than one machine to process in parallel, and by our assumption, $T(G_1) \geq T(G_j)$, the sum of all processing times (if a job needs to be processed by k machines in parallel, then we count its processing time k times) is not less than $2s_j = 2T$. Hence $T_A \geq 2T/m$. We have $T \leq (m/2)T_A \leq (m/2)T^*(1 + \varepsilon)$. □

5. CONCLUSION AND FUTURE RESEARCH

The general multiprocessor task scheduling problem can be attacked in two steps, assignment and schedule. In the first step, we assign jobs to machines and then we schedule them in the second step. We provide a pseudo-polynomial algorithm for the assignment problem, which provides an effective lower bound for the original problem. These pseudo-polynomial time algorithms can also be used to solve the two-machine problem and a special case of three-machine problem *optimally*. Both algorithms are significantly more efficient than those in the literature. We also provide a fully polynomial approximation scheme to solve the assignment problem. In the scheduling step, we provide a heuristic algorithm with error bound $T/T^* \leq (m/2)(1 + \varepsilon)$ for the m-machine problem ($m \geq 2$).

Future research includes solving the problems with: (i) other objective functions such as mean flow time and tardiness related functions and (ii) job-precedence constraints and other job or machine constraints such as machine availability constraints (Lee [22, 23]). We are also interested in using integer programming and branch and bound schemes to solve this large scale problem optimally.

ACKNOWLEDGMENTS

This work is supported in part by NSF Grants CCR-9613805 and DMI-9610229. The authors are grateful to an anonymous Associate Editor and referee whose constructive comments have led to a substantial improvement in the presentation of the paper.

REFERENCES

[1] L. Bianco, J. Blazewicz, P. Dell'Olmo, and M. Drozdowski, Scheduling multiprocessor tasks on a dynamic configuration of dedicated processors, Ann Operations Res 58 (1995), 493–517.

[2] L. Bianco, J. Blazewicz, P. Dell'Olmo, and M. Drozdowski, Preemptive scheduling of multiprocessor task on the dedicated processor system subject to minimal lateness, Inf Process Lett 46 (1993), 109–113.

[3] L. Bianco, J. Blazewicz, P. Dell'Olmo, and M. Drozdowski, Preemptive multiprocessor task scheduling with release times and time windows, Ann Operations Res Sched 70 (1997), 43–57.

[4] J.K. Blazewicz, P. Dell'Olmo, M. Drozdowski, and M.G. Speranza, Scheduling multiprocessor tasks on three dedicated processors, Inf Process Lett 41 (1992), 275–280.

[5] J.K. Blazewicz, M. Drabowski, and J. Weglarz, Scheduling multiprocessor tasks to minimize schedule length, IEEE Trans Comput C-35 (1986), 389–393.
[6] J.K. Blazewicz, W. Drazdowski, and J. Weglarz, Scheduling multiprocess tasks—a survey, Microcomput Appl 13 (1994), 89–97.
[7] J. Blazewicz, M. Drozdowski, G. Schmidt, and D. de Werra, Scheduling independent two-processor tasks on a uniform duo-processor system, Discrete Appl Math 28 (1990), 11–20.
[8] J.K. Blazewicz, J. Weglarz, and M. Drabowski, Scheduling independent 2-processor tasks to minimize scheduling length, Inf Process Lett 18 (1984), 267–273.
[9] P. Brucker, Scheduling algorithm, Springer-Verlag, Berlin, 1995.
[10] P. Brucker and A. Kramer, Shop scheduling problems with multiprocessor tasks on dedicated processors, Ann Operations Res 57 (1995), 13–27.
[11] P. Brucker and A. Kramer, Polynomial algorithms for resource-constrained and multiprocessor task scheduling problems, Eur J Operational Res 90 (1996), 214–226.
[12] X. Cai, C.-Y. Lee, and C.L. Li, Scheduling multiprocessor tasks with prespecified processor allocations, Naval Res Logistics 45 (1998), 231–242.
[13] G. Dobson and U. Karmarkar, Simultaneous resource scheduling to minimize weighted flow times, Operations Res 37 (1989), 592–600.
[14] G. Dobson and I. Khosla, Simultaneous resource scheduling with batching to minimize weighted flow times, IIE Trans 27 (1995), 587–598.
[15] M. Drozdowski, Scheduling multiprocessor tasks—an overview, Eur J Operational Res 94 (1996), 215–230.
[16] J. Du and J.Y.-T. Leung, Complexity of scheduling parallel task systems, SIAM J Discrete Math 2 (1989), 473–487.
[17] M.R. Garey and D.S. Johnson, Computer and intractability: A guide to the theory of NP-completeness, Freeman, New York.
[18] J.A. Hoogeveen, S.L. van de Velde, and B. Deltman, Complexity of scheduling multiprocessor tasks with prespecified allocations, Discrete Appl Math 55 (1994), 259–272.
[19] A. Kramer, Scheduling multiprocessor tasks on dedicated processors, Ph.D. Thesis, FB Mathematik/Informatik, Universität Osnabrück.
[20] H. Krawczyk and M. Kubale, An approximation algorithm for diagnostic test scheduling in multicomputer systems, IEEE Trans Comput 34 (1985), 869–872.
[21] M. Kubale, The complexity of scheduling independent two-processor tasks on dedicated processors, Inf Process Lett 24 (1987), 141–147.
[22] C.-Y. Lee, Machine scheduling with an availability constraint, J Global Optim, Special Issue Optim Scheduling Appl 9 (1996), 395–416.
[23] C.-Y. Lee, Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint, Operations Res Lett 20 (1997), 129–139.
[24] C.-Y. Lee and X. Cai, Scheduling multiprocessor tasks without prespecified processor allocations, submitted for publication.
[25] C.-Y. Lee, L. Lei, and M. Pinedo, Current trends in deterministic scheduling, Ann Operations Res 70 (1997), 1–42.
[26] C.-L. Li, X. Cai, and C.-Y. Lee, Machine scheduling with multiple-job-on-one-machine pattern, IIE Trans 30, (1998), 433–446.
[27] J. Plehn, Preemptive scheduling of independent jobs with release times and deadlines on a hypercube, Inf Process Lett C-34 (1990), 161–166.
[28] B. Veltmann, B.J. Lageweg, and J.K. Lenstra, Multiprocessor scheduling with communication delays, Parallel Comput 16 (1990), 173–182.