SIAM J. COMPUT.
Vol. 31, No. 1, pp. 1–17

A POLYNOMIAL TIME APPROXIMATION SCHEME FOR GENERAL MULTIPROCESSOR JOB SCHEDULING*

JIANER CHEN† AND ANTONIO MIRANDA‡

Abstract. Recently, there have been considerable interests in the multiprocessor job scheduling problem, in which a job can be processed in parallel on one of several alternative subsets of processors. In this paper, a polynomial time approximation scheme is presented for the problem in which the number of processors in the system is a fixed constant. This result is the best possible because of the strong NP-hardness of the problem and is a significant improvement over the past results: the best previous result was an approximation algorithm of ratio $7/6 + \epsilon$ for 3-processor systems based on Goemans's algorithm for a restricted version of the problem.

Key words. job scheduling, approximation algorithm, polynomial time approximation scheme, multiprocessor processing

AMS subject classifications. 68Q20, 68Q25, 90B35, 90C27, 90C39

PII. S0097539798348110

1. Introduction.

One of the assumptions made in classical scheduling theory is that a job is always executed by one processor at a time. With advances in parallel algorithms, this assumption may no longer be valid for job systems. For example, in semiconductor circuit design workforce planning, a design project is to be processed by a group of people. The project contains n jobs, and each job can be worked on by one of a set of alternatives, where each alternative consists of one or more persons in the group working simultaneously on the particular job. The processing time of each job depends on the subgroup of people being assigned to handle the job. Note that the same person may belong to several different subgroups. Now the question is how we can schedule the jobs so that the project can be finished as early as possible. Other applications include (i) the berth allocation problem [23], where a large vessel may occupy several berths for loading and unloading, (ii) diagnosable microprocessor systems [22], 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 (this gives an example for a system in which processors may have different types), and (iv) scheduling a sequence of meetings where each meeting requires a certain group of people [11]. In the scheduling literature [17], these kinds of problems are called *multiprocessor job scheduling* problems.

Among the others, two types of multiprocessor job scheduling problems have been extensively studied [7, 24]. The first type is the $P_m|fix|C_{\max}$ problem, in which the subset of processors and the processing time for parallel processing each job are fixed. The second type is a more general version, the $P_m|set|C_{\max}$ problem, in which each job may have a number of alternative processing modes and each processing

*Received by the editors December 2, 1998; accepted for publication (in revised form) December 22, 2000; published electronically May 31, 2001. A preliminary version of this paper appeared in *Proceedings of the 31st Annual ACM Symposium on Theory of Computing* (STOC'99), Atlanta, 1999, pp. 418–427.

http://www.siam.org/journals/sicomp/31-1/34811.html

†Department of Computer Science, Texas A&M University, College Station, TX 77843-3112 (chen@cs.tamu.edu). This author was supported in part by the National Science Foundation under grants CCR-9613805 and CCR-0000206.

‡Department of Computer Science, Bucknell University, Lewisburg, PA 17837 (amiranda@eg.bucknell.edu).

mode specifies a subset of processors and the job processing time on that particular processor subset. The objective for both problems is to construct a scheduling of minimum makespan on the m-processor system for a given list of jobs. The jobs are supposed to be nonpreemptive.

Approximability of the multiprocessor job scheduling problems has been studied. The $P_2|set|C_{\max}$ problem is a generalized version of the classical job scheduling problem on a 2-processor system [13]; thus it is NP-hard. Hoogeveen, van de Velde, and Veltman [18] showed that the $P_3|fix|C_{\max}$ problem (thus also the $P_3|set|C_{\max}$ problem) is NP-hard in the strong sense; thus it does not have a fully polynomial time approximation scheme unless P = NP (see also [4, 5]). Blazewicz et al. [4] developed a polynomial time approximation algorithm of ratio 4/3 for the problem $P_3|fix|C_{\max}$, which was improved later by Dell'Olmo, Speranza, and Tuza [10], who gave a polynomial time approximation algorithm of ratio 5/4 for the same problem. Both algorithms are based on the study of a special type of schedulings called *normal schedulings*. Goemans [14] further improved the algorithms by giving a polynomial time approximation algorithm of ratio 7/6 for the $P_3|fix|C_{\max}$ problem. More recently, Amoura et al. [1] developed a polynomial time approximation scheme for the problem $P_m|fix|C_{\max}$ for every fixed integer m.

Approximation algorithms for the $P_m|set|C_{\max}$ problem were not as successful as that for the $P_m|fix|C_{\max}$ problem. Bianco et al. [3] presented a polynomial time approximation algorithm for the $P_m|set|C_{\max}$ problem whose approximation ratio is bounded by m. Chen and Lee [8] improved their algorithm by giving a polynomial time approximation algorithm for the $P_m|set|C_{\max}$ problem with an approximation ratio $m/2 + \epsilon$. Miranda [25] showed that the problem $P_3|set|C_{\max}$ can be approximated in polynomial time with a ratio $7/6 + \epsilon$. Before the present paper, it was unknown whether there is a polynomial time approximation algorithm with ratio c for the problem $P_m|set|C_{\max}$, where c is a constant independent of the number m of processors in the system.

In this paper, we present a polynomial time approximation scheme for the problem $P_m|set|C_{\max}$. Our algorithm combines the techniques developed by Amoura et al. [1], who split jobs into large jobs and small jobs, and the techniques developed by Dell'Olmo, Speranza, and Tuza [10] and Goemans [14] on normal schedulings, plus the standard dynamic programming and scaling techniques. More precisely, based on a classification of large jobs and small jobs, we introduce the concept of (m, ϵ)-*canonical schedulings*, which can be regarded as a generalization of the normal schedulings. We show that for any job list, there is an (m, ϵ)-canonical scheduling whose makespan is very close to the optimal makespan. Then we show how this (m, ϵ)-canonical scheduling can be approximated. Combining these two steps gives us a polynomial time approximation scheme for the $P_m|set|C_{\max}$ problem.

Our result is the best possible in the following sense: because the problem $P_m|set|C_{\max}$ is NP-hard in the strong sense, it is unlikely that our algorithm can be further improved to a fully polynomial time approximation scheme [13]. Moreover, the polynomial time approximation scheme cannot be extended to the more general problem $P|set|C_{\max}$, in which the number m of processors in the system is given as a parameter in the input: it can be shown that there is a constant $\delta > 0$ such that the problem $P|set|C_{\max}$ has no polynomial time approximation algorithms whose approximation ratio is bounded by n^δ [25].

The paper is organized as follows. Section 2 gives necessary background and preliminaries for the problem. In section 3 we introduce (m, ϵ)-canonical schedulings

and study their properties. Section 4 presents the polynomial time approximation scheme for the problem $P_m|set|C_{\max}$ and section 5 concludes with some remarks and further research directions.

2. Preliminaries. We assume readers' familiarity with the basic concepts in approximation theory [13], such as approximation algorithms, approximation ratios, polynomial time approximation schemes, and fully polynomial time approximation schemes.

The $P_m|set|C_{\max}$ problem is a scheduling problem minimizing the makespan for a set of jobs, each of which may have several alternative processing modes. More formally, an instance $\mathcal{J}$ of the problem $P_m|set|C_{\max}$ is a list of jobs: $\{J_1, J_2, \ldots, J_n\}$, where each job J_i is associated with a list of alternative *processing modes*: $J_i = [M_{i1}, \ldots, M_{ip_i}]$. Each processing mode (or simply *mode*) M_{ij} is specified by a pair (Q_{ij}, t_{ij}), where Q_{ij} is a subset of processors in the m-processor system and t_{ij} is an integer indicating the parallel processing time of the job J_i on the processor set Q_{ij}. In case there is no ambiguity, we also say that the processor set Q_{ij} is a mode for the job J_i. For each job $J_i = [M_{i1}, \ldots, M_{ip_i}]$, where $M_{ij} = (Q_{ij}, t_{ij})$, we let $\min_i$ be the minimum t_{ij} over all j, $1 \le j \le p_i$. The value $\min_i$ will be called the *minimum parallel processing time* for the job J_i.

Given a list $\mathcal{J} = \{J_1, \ldots, J_n\}$ of jobs, a *scheduling* $\Gamma(\mathcal{J})$ of $\mathcal{J}$ on the m-processor system consists of two parts: (1) determination of a processing mode for each job J_i in $\mathcal{J}$ and (2) determination of the starting execution time for each job under the assigned mode so that at any moment, each processor in the system is used for (maybe parallel) processing at most one job (assuming that the system starts at time $\tau = 0$). The *makespan* of the scheduling $\Gamma(\mathcal{J})$ is the latest finishing time of a job in $\mathcal{J}$ under the scheduling $\Gamma(\mathcal{J})$. Let $Opt(\mathcal{J})$ denote the minimum makespan over all schedulings for $\mathcal{J}$. The $P_m|set|C_{\max}$ problem is for a given instance $\mathcal{J}$ to construct a scheduling of makespan $Opt(\mathcal{J})$ for $\mathcal{J}$.

Let P_m be the set of the m processors in the m-processor system. A collection $\{P'_1, \ldots, P'_k\}$ of k nonempty subsets of P_m is a *k-partition* of P_m if $P_m = \bigcup_{i=1}^{k} P'_i$ and $P'_i \cap P'_j = \emptyset$ for all $i \ne j$. A collection of subsets of P_m is a *partition* of P_m if it is a k-partition for some integer $k \ge 1$. The total number B_m of different partitions of the set P_m is called the mth *Bell number* [16]. It can be proved easily by induction that $B_m \le m!$.

Another combinatorial fact we need for analysis of our scheduling algorithm is the "cut-index" in a nonincreasing sequence of integers.

LEMMA 2.1. *Let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be a nonincreasing sequence of integers, let $m \ge 2$ be a fixed integer, and let $\epsilon > 0$ be an arbitrary real number. Then there is an index j_0 (with respect to m and ϵ) such that*

(1) *$j_0 = (3mB_m + 1)^k$, where $k \le \lfloor m/\epsilon \rfloor$ is an integer and*
(2) *for any subset $\mathcal{T}'$ of at most $3j_0mB_m$ integers t_q in $\mathcal{T}$ with $q > j_0$, we have*

$$\sum_{t_q \in \mathcal{T}'} t_q \le (\epsilon/m) \sum_{i=1}^{n} t_i.$$

Proof. To simplify expressions, let $b_m = 3mB_m + 1$. Decompose the sum $t_1 + t_2 + \cdots + t_n$ into subsums

$$A_1 = t_1 + \cdots + t_{b_m},$$
$$A_2 = t_{b_m+1} + \cdots + t_{b_m^2},$$

$$\begin{aligned}
&\cdots\cdots\\
A_j &= t_{b_m^{j-1}+1} + \cdots + t_{b_m^j},\\
&\cdots\cdots\\
A_h &= t_{b_m^{h-1}+1} + \cdots + t_n,
\end{aligned}$$

where $h = \lceil \log n / \log b_m \rceil$.

Since $\sum_{j=1}^{h} A_j = \sum_{i=1}^{n} t_i$, there are at most $\lfloor m/\epsilon \rfloor$ subsums A_j larger than $(\epsilon/m)\sum_{i=1}^{n} t_i$. Let A_{k+1} be the first subsum such that $A_{k+1} \le (\epsilon/m)\sum_{i=1}^{n} t_i$; then $k \le \lfloor m/\epsilon \rfloor$.

Let $j_0 = b_m^k = (3mB_m + 1)^k$. Since the sum of the first $b_m^{k+1} - b_m^k = 3j_0 mB_m$ integers t_q in $\mathcal{T}$ with $q > j_0 = b_m^k$ is bounded by $(\epsilon/m)\sum_{i=1}^{n} t_i$,

$$A_{k+1} = t_{b_m^k+1} + \cdots + t_{b_m^{k+1}} \le (\epsilon/m)\sum_{i=1}^{n} t_i,$$

and the sequence $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ is nonincreasing, we conclude that for any subset $\mathcal{T}'$ of $\mathcal{T}$ of at most $3j_0 mB_m$ integers t_q with $q > j_0$, we must have

$$\sum_{t_q \in \mathcal{T}'} t_q \le (\epsilon/m)\sum_{i=1}^{n} t_i.$$

This completes the proof. □

For the nonincreasing sequence $\mathcal{T}$ of integers, we will denote by $j_{m,\epsilon}$ the smallest index that satisfies conditions (1) and (2) in Lemma 2.1. The index $j_{m,\epsilon}$ will be called the *cut-index* for the sequence $\mathcal{T}$.

3. On (m, ϵ)-canonical schedulings. In this section, we first assume that the mode assignment for each job in the instance $\mathcal{J}$ is decided and discuss how we schedule the jobs in $\mathcal{J}$ under the mode assignment to the processor set P_m. By this assumption, the job list $\mathcal{J}$ is actually an instance for the $P_m|fix|C_{\max}$ problem (recall that the $P_m|fix|C_{\max}$ problem is the problem $P_m|set|C_{\max}$ with the restriction that every job in an instance has only one processing mode).

Let $\mathcal{J} = \{J_1, \ldots, J_n\}$ be an instance for the $P_m|fix|C_{\max}$ problem, where each job J_i requires a fixed set Q_i of processors for parallel execution with processing time t_i for $i = 1, 2, \ldots, n$. Without loss of generality, assume that the processing time sequence $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ is nonincreasing.

For the fixed number m of processors in the system and for an arbitrarily given real number $\epsilon > 0$, let $j_{m,\epsilon}$ be the cut-index for the sequence $\mathcal{T}$ as defined in Lemma 2.1. That is, $j_{m,\epsilon} = (3mB_m + 1)^k$, where k is an integer bounded by $\lfloor m/\epsilon \rfloor$, and for any subset $\mathcal{T}'$ of at most $3j_{m,\epsilon} mB_m$ integers t_q in $\mathcal{T}$ with $q > j_{m,\epsilon}$, we have $\sum_{t_q \in \mathcal{T}'} t_q \le (\epsilon/m)\sum_{i=1}^{n} t_i$. We split the job set $\mathcal{J}$ into two subsets

$$\mathcal{J}_L = \{J_i \mid i \le j_{m,\epsilon}\}, \qquad \mathcal{J}_S = \{J_i \mid i > j_{m,\epsilon}\}. \tag{3.1}$$

The jobs in $\mathcal{J}_L$ will be called *large jobs* and the jobs in $\mathcal{J}_S$ will be called *small jobs*.

Let $\Gamma(\mathcal{J})$ be a scheduling for the job set $\mathcal{J}$. Consider the nondecreasing sequence $\{\tau_1, \tau_2, \ldots, \tau_h\}$ of integers, where $\tau_1 = 0$, $\tau_h = +\infty$, $h = 2j_{m,\epsilon} + 2$, and τ_i, $1 < i < h$, are the starting or finishing times of the $j_{m,\epsilon}$ large jobs in $\Gamma(\mathcal{J})$. A *small job block* χ in $\Gamma(\mathcal{J})$ consists of a subset $P' \subseteq P_m$ of processors and a time interval $[\tau_p, \tau_{p+1}]$,

$1 \le p \le h-1$, such that the subset $P_m - P'$ of processors are exactly those that are executing large jobs in the time interval $[\tau_p, \tau_{p+1}]$. The value $\tau_{p+1} - \tau_p$ will be called the *height* and the processor set P' will be called the *type* of the small job block χ.

Therefore, the subset P' of processors associated with the small job block χ are those processors that are either idle or used for executing small jobs in the time interval $[\tau_p, \tau_{p+1}]$. Note that the small job block χ can be of height 0 when $\tau_p = \tau_{p+1}$. The small job block of time interval $[\tau_{h-1}, +\infty]$, where τ_{h-1} is the latest finish time of a large job, will be called the "last small job block." Note that the last small job block has type P_m.

Let χ be a small job block associated with a processor set P' and a time interval $[\tau_p, \tau_{p+1}]$. The small job block at any time moment τ in the time interval $[\tau_p, \tau_{p+1}]$ can be characterized uniquely as a collection $[Q_1, \ldots, Q_s]$ of pairwise disjoint subsets of the processor set P' such that at the time τ, for $i = 1, \ldots, s$, all processors in the subset Q_i are used for parallel execution on the same small job (thus, the subset $P' - \bigcup_{i=1}^{s} Q_i$ is the subset of idle processors at time τ). The collection $[Q_1, \ldots, Q_s]$ will be called the *type* of the time moment τ. A *layer* in the small job block χ is a maximal time interval $[\tau', \tau'']$ in $[\tau_p, \tau_{p+1}]$ such that all time moments τ between τ' and τ'' are of the same type. The *type* of the layer is equal to the type of any time moment in the layer and the *height* of the layer is $\tau'' - \tau'$.

Let L_1 and L_2 be two layers in the small job block χ of types $[Q_1, \ldots, Q_s]$ and $[R_1, \ldots, R_t]$, respectively. We say that layer L_1 *covers* layer L_2 if $\{R_1, \ldots, R_t\} \subseteq \{Q_1, \ldots, Q_s\}$. In particular, if L_1 and L_2 are two consecutive layers in the small job block χ such that layer L_2 starts right after layer L_1 finishes and L_1 covers L_2, then layer L_2 is actually a continuation of the layer L_1 with some of the small jobs finished.

DEFINITION 3.1. *A* floor σ *in the small job block* χ *is a sequence* $\{L_1, L_2, \ldots, L_z\}$ *of consecutive layers such that* (1) *for* $i = 2, \ldots, z$, *layer* L_i *starts right after layer* L_{i-1} *finishes, and layer* L_{i-1} *covers layer* L_i*; and* (2) *all small jobs interlacing layer* L_1 *start in layer* L_1 *and all small jobs interlacing layer* L_z *finish in layer* L_z.

An example of a floor is given in Figure 3.1(a). Note that a small job block may not have any nonempty floor at all, as shown in Figure 3.1(b).

Remark 1. There are a few important properties of floors in a small job block. Suppose that the layer L_1 starts at time τ' while layer L_z finishes at time τ''. Then by property (2) in the definition, no small jobs cross the floor boundaries τ' and τ''. Therefore, the floor σ can be regarded as a single job that uses the processor set P', starts at time τ', and finishes at time τ''. The *height* of the floor σ is defined to be $\tau'' - \tau'$, which is equal to the sum of the heights of the layers L_1, ..., L_z. Second, since all floors in the small job block χ are for the same processor subset P' and there are no small jobs crossing the starting and finishing times of any floors, the floors in the same small job block χ can be rearranged in any order but can still fit into the small job block without exceeding the height of the small job block. Finally, property (1) in the definition ensures that no matter how the small jobs in a floor are rearranged, a simple greedy algorithm is sufficient to refit the small jobs into the floor without exceeding the floor height. The greedy algorithm is based on the idea of the well-known Graham's list scheduling algorithm for the classical job scheduling problem [15].

DEFINITION 3.2. *Let* $\mathcal{J}$ *be an instance of the problem* $P_m|fix|C_{\max}$ *and let* π *be any permutation of the jobs in* $\mathcal{J}$. *The* list scheduling algorithm *based on the ordering* π *is to schedule each job* J_i *of mode* Q_i *in* $\mathcal{J}$, *following the ordering of* π, *at the earliest time when the processor subset* Q_i *becomes available.*

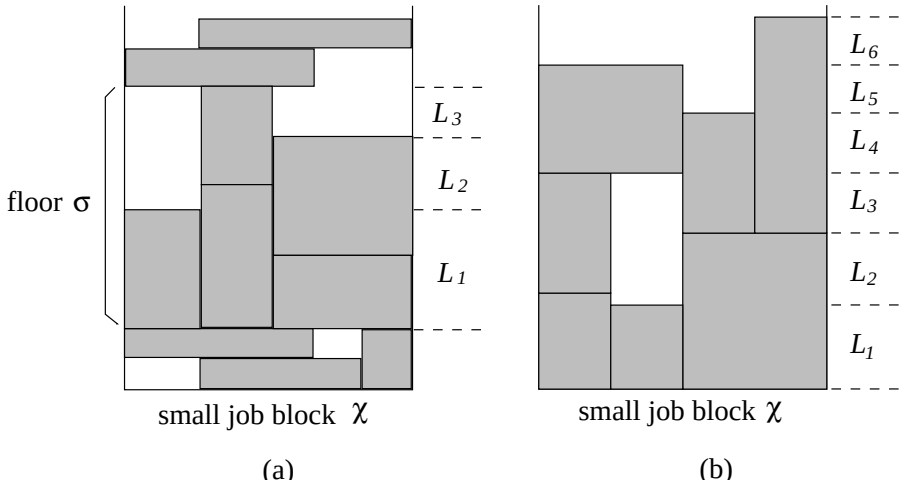

FIG. 3.1. (a) *a floor* $\{L_1, L_2, L_3\}$; (b) *a small job block with no floor.*

.

LEMMA 3.3. *Let $\mathcal{J}_\sigma$ be the set of small jobs in the floor σ. The list scheduling algorithm based on any ordering π of the jobs in $\mathcal{J}_\sigma$ will always reconstruct the floor σ.*

Proof. Suppose that the first layer in the floor σ is of type $[Q_1, \ldots, Q_s]$. Every job in $\mathcal{J}_\sigma$ must have a mode Q_i for some i, and no processor subset Q_i can become idle before its final completion time. The jobs of each mode Q_i in $\mathcal{J}_\sigma$ can be executed by the processor subset Q_i in any order without changing the completion time of Q_i. Since the list scheduling algorithm starts each job at its earliest possible time (thus no subset Q_i can become idle before its final completion time), the completion time for each subset Q_i will not be changed. Therefore, the list scheduling algorithm will construct a floor with exactly the same layers. □

DEFINITION 3.4. *Let $[Q_1, \ldots, Q_s]$ be a partition of the processor subset P'. We say that* we can assign the type $[Q_1, \ldots, Q_s]$ *to a floor $\sigma = \{L_1, \ldots, L_z\}$ if the type of the layer L_1 is a subcollection of $\{Q_1, \ldots, Q_s\}$.*

It is possible that several different types can be assigned to the same floor as long as the type of the floor is a subcollection of the assigned floor types. For example, let $[Q_1, \ldots, Q_s]$ be a partition of the processor subset P'. If the first layer L_1 in a floor σ is of type $[Q_3, \ldots, Q_s]$, then we can assign either type $[Q_1, Q_2, Q_3, \ldots, Q_s]$ or type $[Q_1 \cup Q_2, Q_3, \ldots, Q_s]$ to the floor σ.

DEFINITION 3.5. *A small job block χ is a* tower *if it is constituted by a sequence of floors such that we can assign types to the floors so that no two floors in the tower χ are of the same type.*

Note that since each floor type is a partition of the processor subset P', a tower contains at most B_m floors, where $B_m \leq m!$, the mth Bell number, is the number of different partitions of a set of m elements.

In our discussion, we will be concentrating on schedulings of a special form in the following sense.

DEFINITION 3.6. *Let $\mathcal{J}$ be an instance of the problem $P_m|fix|C_{\max}$, which is divided into large job set $\mathcal{J}_L$ and small job set $\mathcal{J}_S$ as given in* (3.1) *for a fixed integer $m > 2$ and a fixed constant $\epsilon > 0$. A scheduling $\Gamma(\mathcal{J})$ of $\mathcal{J}$ is* (m, ϵ)-canonical *if every*

small job block in $\Gamma(\mathcal{J})$ *is a tower.*

Remark 2. Note that in an (m, ϵ)-canonical scheduling, no small jobs cross the boundary of a tower. Therefore, a tower of height t and associated with a processor set Q can be simply regarded as a job of mode (Q, t).

We first show that an (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ of $\mathcal{J}$ can be constructed by the list scheduling algorithm when large jobs and towers in $\Gamma(\mathcal{J})$ are given in a proper order.

LEMMA 3.7. *Let* $\Gamma(\mathcal{J})$ *be an* (m, ϵ)*-canonical scheduling for the job set* $\mathcal{J}$*. Let* π *be the sequence of the large jobs and towers in* $\Gamma(\mathcal{J})$*, ordered in terms of their starting times in* $\Gamma(\mathcal{J})$*. Then the list scheduling algorithm based on the ordering* π*, which regards each tower as a single job, constructs a scheduling of* $\mathcal{J}$ *with makespan not larger than that of* $\Gamma(\mathcal{J})$*.*

Proof. Let $\mathcal{J}_i = \{J_1, \ldots, J_i\}$ be any prefix of the ordered sequence π, where each J_j is either a large job or a tower. Let $\Gamma(\mathcal{J}_i)$ be the scheduling of $\mathcal{J}_i$ obtained from $\Gamma(\mathcal{J})$ by removing all large jobs and towers that are not in $\mathcal{J}_i$ and let $\Gamma'(\mathcal{J}_i)$ be the scheduling by the list scheduling algorithm on the jobs in $\mathcal{J}_i$. By induction, it is not difficult to prove that the completion time of *each* processor in $\Gamma'(\mathcal{J}_i)$ is not larger than the completion time of the same processor in $\Gamma(\mathcal{J}_i)$. For $\mathcal{J}_i = \mathcal{J}$, this implies that the makespan of the scheduling constructed by the list scheduling algorithm based on the ordering π is not larger than the makespan of the (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$. □

Thus, once the ordering of large jobs and towers is decided, it is easy to construct a scheduling that is not worse than the given (m, ϵ)-canonical scheduling. In the following, we will prove that for any instance $\mathcal{J}$ for the problem $P_m|fix|C_{\max}$, there is an (m, ϵ)-canonical scheduling whose makespan is very close to the optimal makespan.

THEOREM 3.8. *Let* $\mathcal{J}$ *be an instance for the problem* $P_m|fix|C_{\max}$*. Then for any* $\epsilon > 0$*, there is an* (m, ϵ)*-canonical scheduling* $\Gamma(\mathcal{J})$ *of* $\mathcal{J}$ *such that the makespan of* $\Gamma(\mathcal{J})$ *is bounded by* $(1 + \epsilon)Opt(\mathcal{J})$*.*

Proof. Let $\Gamma_1(\mathcal{J})$ be an optimal scheduling of makespan $Opt(\mathcal{J})$ for $\mathcal{J}$. We construct an (m, ϵ)-canonical scheduling for $\mathcal{J}$ based on the optimal scheduling $\Gamma_1(\mathcal{J})$. Let $\mathcal{J}_L$ and $\mathcal{J}_S$ be the set of large jobs and the set of small jobs in $\mathcal{J}$, respectively, according to the definition in (3.1). Consider a small job block χ in the scheduling $\Gamma_1(\mathcal{J})$.

Assume that the small job block χ is associated with a processor set P' of r processors, $r \leq m$, and a time interval $[\tau_p, \tau_{p+1}]$. Let $[T_1, \ldots, T_y]$ be the list of all partitions of the processor set P', where $y = B_r \leq B_m$. We divide the layers in the small job block χ into groups, each corresponding to a partition of P', as follows. A layer of type T' is put in the group corresponding to a partition T_j if T' is a subcollection of T_j. Note that a layer type T' may be a subcollection of more than one partition of P'. In this case, we put the layer arbitrarily into one and only one of the groups to ensure that each layer belongs to only one group.

For each partition T_j of P', we construct a *floor frame* σ_j whose type is T_j and height is equal to the sum of heights of all layers belonging to the group corresponding to the partition T_j. Note that so far we have not yet actually assigned any small jobs to any floor frames σ_1, …, σ_y. Moreover, since each layer belongs to exactly one of the groups, it is easy to see that the sum $\sum_{j=1}^{y} height(\sigma_j)$ of the heights of the floor frames σ_1, …, σ_y is equal to the sum of the heights of all layers in the small job block χ, which is equal to the height of the small job block χ.

The construction for the floor frames for the last small job block in $\Gamma_1(\mathcal{J})$ is

slightly different: we group layers only in which not all processors are idle. Thus, the sum of the heights of all floor frames in the last small job block is equal to $Opt(\mathcal{J})-\tau_0$, where τ_0 is the latest finish time for some large job in the scheduling $\Gamma_1(\mathcal{J})$.

After the construction of the floor frames for each small job block in the scheduling $\Gamma_1(\mathcal{J})$, we assign the small jobs in $\mathcal{J}_S$ to the floor frames using the following greedy method. For each small job J that requires a parallel processing by a processor subset Q, we assign J to an arbitrary floor frame σ in a small job block as long as the floor frame σ satisfies the following conditions: (1) the type of the floor frame σ contains the subset Q and (2) adding the job J to σ does not exceed the height of the floor frame σ (if there are more than one floor frames satisfying these conditions, arbitrarily pick one of them). Note that we assign a job to a floor frame only when the mode of the job is contained in the type of the floor frame. Therefore, this assignment will never leave a "gap" between two jobs in the same floor frame.

The above assignment of small jobs in $\mathcal{J}_S$ to floor frames stops when none of the small jobs left in $\mathcal{J}_S$ can be assigned to any of the floor frames according to the above rules. Now each floor frame becomes a floor.

For each small job block χ in $\Gamma_1(\mathcal{J})$, let S_χ be the set of floor frames in χ. Since the height of a resulting floor is not larger than the height of the corresponding floor frame, the sum of the heights of the floors resulting from the floor frames in S_χ is not larger than the height of the small job block χ. Therefore, we can put all these floors into the small job block χ (in an arbitrary order) to make χ a tower. Doing this for all small job blocks in $\Gamma_1(\mathcal{J})$ gives an (m,ϵ)-canonical scheduling $\Gamma_2(\mathcal{J}_L \cup \mathcal{J}'_S)$ for the job set $\mathcal{J}_L \cup \mathcal{J}'_S$, where $\mathcal{J}'_S$ is the set of small jobs that have been assigned to the floor frames in the above procedure. The makespan of the scheduling $\Gamma_2(\mathcal{J}_L \cup \mathcal{J}'_S)$ is bounded by $Opt(\mathcal{J})$. Now the only thing left is that we still need to schedule the small jobs that have not been assigned to any floor frames. Let $\mathcal{J}''_S = \mathcal{J}_S - \mathcal{J}'_S$ be the set of small jobs that are not assigned to any floor frames by the above procedure. We want to demonstrate that there are not many jobs in the set $\mathcal{J}''_S$.

By the definition, the number of small job blocks in the scheduling $\Gamma_1(\mathcal{J})$ is $2j_{m,\epsilon}+1 \le 3j_{m,\epsilon}$. Since each small job block is associated with at most m processors, the number of floor frames constructed in each small job block is bounded by B_m. Therefore, the total number of floor frames we constructed from the scheduling $\Gamma_1(\mathcal{J})$ is bounded by $3B_m j_{m,\epsilon}$. Moreover, each floor type is a collection of at most m processor subsets.

If the set $\mathcal{J}''_S$ contains more than $3mB_m j_{m,\epsilon}$ small jobs, then there must be a subset Q of processors such that the number of small jobs of mode Q in $\mathcal{J}''_S$ is larger than the number of the constructed floor frames whose type contains the subset Q. Let $\{\sigma_1, \ldots, \sigma_d\}$ be the set of floor frames whose type contains the subset Q.

By our assignment rules, assigning any job of mode Q in $\mathcal{J}''_S$ to a floor frame in $\{\sigma_1, \ldots, \sigma_d\}$ would exceed the height of the corresponding floor frame. Since there are more than d small jobs of mode Q in $\mathcal{J}''_S$, the sum of processing times of all small jobs of mode Q in $\mathcal{J}_S$ is larger than $\sum_{i=1}^{d} height(\sigma_i)$. On the other hand, by our construction of the floor frames in each small job block χ, the sum of the heights of the floor frames in χ whose type contains Q should not be smaller than the sum of the heights of the layers in χ whose type contains Q. Summarizing this over all small job blocks, we conclude that the sum $\sum_{i=1}^{d} height(\sigma_i)$ is not smaller than the sum of processing times of all small jobs of mode Q in $\mathcal{J}_S$ (since each small job of mode Q must be contained in consecutive layers whose type contains Q). This derives a contradiction. The contradiction shows that there are at most $3mB_m j_{m,\epsilon}$ small jobs

in the set $\mathcal{J}''_S$.

Now we assign the small jobs in $\mathcal{J}''_S$ to the floor frames in the last small job block in the scheduling $\Gamma_2(\mathcal{J}_L \cup \mathcal{J}'_S)$. For each small job J of mode Q in $\mathcal{J}''_S$, we arbitrarily assign J to a floor frame whose type contains Q in the last small job block, even if this assignment exceeds the height of the floor frame. Note that the last small job block is associated with the whole processor set P_m, so for any mode Q, there must be a floor frame in the last small job block whose type contains the processor subset Q. This procedure stops with all small jobs in $\mathcal{J}''_S$ assigned to floor frames in the last small job block. It is easy to see that the resulting scheduling is an (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ of the original job set $\mathcal{J}$. Moreover, since the makespan of the scheduling $\Gamma_2(\mathcal{J}_L \cup \mathcal{J}'_S)$ is bounded by $Opt(\mathcal{J})$, the makespan of the (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ is bounded by

$$Opt(\mathcal{J}) + \sum_{J \in \mathcal{J}''_S} t(J),$$

where $t(J)$ is the parallel processing time of the small job J. Since there are at most $3mB_m j_{m,\epsilon}$ small jobs in the set $\mathcal{J}''_S$, by Lemma 2.1,

$$\sum_{J \in \mathcal{J}''_S} t(J) \le (\epsilon/m) \sum_{i=1}^{n} t_i.$$

It is easy to see that $Opt(\mathcal{J}) \ge (\sum_{i=1}^{n} t_i)/m$. Therefore, the makespan of the (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ is bounded by $(1+\epsilon)Opt(\mathcal{J})$. This completes the proof of the theorem. □

Before we close this section, we introduce one more definition.

DEFINITION 3.9. *Let σ be a floor of type $[Q_1, \ldots, Q_s]$ and height l, where Q_1, ..., Q_s are pairwise disjoint subsets of processors in the processor set P_m. Then each subset Q_i plus the height l is called a* room of type Q_i *in the floor σ.*

4. The approximation scheme. Now we come back to the original problem $P_m|set|C_{\max}$. Recall that an instance $\mathcal{J}$ of the problem $P_m|set|C_{\max}$ is a set of jobs $\{J_1, J_2, \ldots, J_n\}$, where each job J_i is given by a list of alternative processing modes $[M_{i,1}, \ldots, M_{i,p_i}]$ in which each processing mode $M_{i,j} = (Q_{i,j}, t_{i,j})$ specifies the parallel processing time $t_{i,j}$ of the job J_i on the subset $Q_{i,j}$ of processors in the m-processor system.

In order to describe our polynomial time approximation scheme for the problem, let us first discuss why this problem is more difficult than the classical job scheduling problem.

In the classical job scheduling problem, each job is executed by one processor in the system. Therefore, the order of executions of jobs in each processor is not crucial: the running time of the processor is simply equal to the sum of the processing times of the jobs assigned to the processor. Therefore, the decision of which job should be assigned to which processor, in any order, will uniquely determine the makespan of the resulting scheduling. This makes it possible to use a dynamic programming approach that extends a scheduling for a subset of jobs to that for a larger subset.

The situation in the general multiprocessor job scheduling problem $P_m|set|C_{\max}$, on the other hand, is more complicated. In particular, the makespan of a scheduling depends not only on the assignment of processing modes to jobs but also on the *order* in which the jobs are executed. Therefore, the techniques of extending a scheduling

for a subset of jobs in the classical job scheduling problem are not directly applicable here.

Theorem 3.8 shows that there is an (m, ϵ)-canonical scheduling whose makespan is very close to the optimal makespan. Therefore, constructing a scheduling whose makespan is not larger than the makespan of a good (m, ϵ)-canonical scheduling will give a good approximation to the optimal schedulings.

Nice properties of an (m, ϵ)-canonical scheduling are that within the same tower, the order of the floors does not affect the height of the tower and that within the same floor, the order of the small jobs does not affect the height of the floor (see Remarks 1 and 2 in the previous section). Therefore, the only factor that affects the heights of towers and floors is the assignments of jobs to towers and floors. This makes it become possible, at least for small jobs, to apply the techniques in classical job scheduling problems to our current problem. This is described as follows.

First, suppose that we can somehow divide the job set $\mathcal{J}$ into large job set $\mathcal{J}_L$ and small job set $\mathcal{J}_S$. Let us start with an (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ of the job set $\mathcal{J}$. The scheduling $\Gamma(\mathcal{J})$ gives a nondecreasing sequence $\{\tau_0, \tau_1, \ldots, \tau_{p+1}\}$ of integers, where $\tau_0 = 0$, $\tau_{p+1} = +\infty$, $p = 2j_{m,\epsilon}$, and τ_i, $0 < i < p + 1$, are the starting or finishing times of the $j_{m,\epsilon}$ large jobs in $\mathcal{J}_L$. Let the $p + 1$ corresponding towers be $\{\chi_0, \chi_1, \ldots, \chi_p\}$, where the tower χ_j consists of a subset P'_j of processors and the time interval $[\tau_j, \tau_{j+1}]$.

We suppose that the subset P'_j of processors associated with each tower χ_j is known and that the large jobs and towers of the scheduling $\Gamma(\mathcal{J})$ are ordered into a sequence π in terms of their starting times. However, we assume that the assignment of small jobs to the rooms of the scheduling $\Gamma(\mathcal{J})$ is unknown. We show how this information can be recovered.

For each tower χ_j associated with the processor set P'_j, the number of floors in the tower χ_j is $q_j = B_r \leq B_m$, where r is the number of processors in the set P'_j. Let $\sigma_{j,1}$, ..., σ_{j,q_j} be the floors of all possible different types in the tower χ_j. For each floor $\sigma_{j,q}$, let $\gamma_{j,q,1}$, ..., $\gamma_{j,q,r_{jq}}$ be the rooms in the floor $\sigma_{j,q}$, where $r_{jq} \leq m$. Therefore, the configuration of the small jobs in the (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ can be specified by a $((2j_{m,\epsilon} + 1)B_m m)$-tuple

$$[t_{0,1,1}, \ldots, t_{j,q,r}, \ldots, t_{2j_{m,\epsilon},B_m,m}],$$

where $t_{j,q,r}$ specifies the running time of the room $\gamma_{j,q,r}$ (for index $\{j, q, r\}$ for which the corresponding room $\gamma_{j,q,r}$ does not exist, we can simply set $t_{j,q,r} = -1$).

Suppose that an upper bound T_0 for the running time of rooms is derived; then we can use a Boolean array D of $(2j_{m,\epsilon} + 1)B_m m + 1$ dimensions to describe the configuration of a subset of small jobs in a scheduling

$$D[0..n_S;\ \underbrace{0..T_0,\ \ldots\ldots,\ 0..T_0}_{(2j_{m,\epsilon}+1)B_m m}],$$

where $n_S = n - j_{m,\epsilon}$ is the number of small jobs in $\mathcal{J}$ such that

$$D[i; t_{0,1,1}, \ldots, t_{j,q,r}, \ldots, t_{2j_{m,\epsilon},B_m,m}] = \textsc{True}$$

if and only if there is a scheduling on the first i small jobs to the floors in $\Gamma(\mathcal{J})$ such that the running time of the room $\gamma_{j,q,r}$ is $t_{j,q,r}$ (recall that the running time of a room is dependent only on the assignment of small jobs to the room and independent

of the order in which the small jobs are executed in the room). Initially, all array elements in the array $D[\cdots\cdots]$ have value FALSE.

Suppose that a configuration of a scheduling for the first $i-1$ small jobs is given:

$$D[i-1; t_{0,1,1}, \ldots, t_{j,q,r}, \ldots, t_{2j_{m,\epsilon},B_m,m}] = \text{TRUE}. \tag{4.1}$$

We say that the ith small job J_i' under mode Q_i is *addable* to a room $\gamma_{j,q,r}$ in the configuration in (4.1) if the room $\gamma_{j,q,r}$ is of type Q_i and adding the job J_i' to the room does not exceed the upper bound T_0 of the running time of the room $\gamma_{j,q,r}$.

Now we are ready to present our dynamic programming algorithm for scheduling small jobs into the rooms in the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$. The algorithm is given in Figure 4.1.

Note that the algorithm SCHEDULE-SMALL may not return an (m,ϵ)-canonical scheduling for the job set $\mathcal{J}$. In fact, there is no guarantee that the height of the towers constructed in the algorithm does not exceed the height of the corresponding towers in the original (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$. We first show below that the scheduling constructed by the algorithm SCHEDULE-SMALL has its makespan bounded by the makespan of the original (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$.

The following lemma can be proved by induction on the index i.

LEMMA 4.1. *For all i, $0 \le i \le n_S$, the array element $D[i; \ldots, t_{j,q,r}, \ldots] =$ TRUE if and only if there is a way to assign modes to the first i small jobs and arrange them into the rooms such that the room $\gamma_{j,q,r}$ has running time $t_{j,q,r}$ for all $\{j,q,r\}$.*

Lemma 4.1 gives us directly the following corollary.

COROLLARY 4.2. *If the sequence π of large jobs and towers is ordered in terms of their starting times in the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$, then the algorithm* SCHEDULE-SMALL *constructs a scheduling for the job set $\mathcal{J}$ with makespan bounded by the makespan of the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$.*

Proof. Note that the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ gives a way to assign and arrange all small jobs in $\mathcal{J}_S$ into the rooms. According to Lemma 4.1, the corresponding array element in the array D must have value TRUE:

$$D[n_S; \ldots, t_{j,q,r}, \ldots] = \text{TRUE}.$$

For this array element, step 3 of the algorithm will construct the towers that have exactly the same types and heights as their corresponding towers in the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$. (This may not give exactly the same assignment of small jobs to rooms. However, the running times of the corresponding rooms must be exactly the same.) Now since the sequence π is given in the order sorted by the starting times of the large jobs and towers in the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$, by Lemma 3.7, the call in step 3 to the list scheduling algorithm based on the order π and this configuration will construct a scheduling whose makespan is not larger than the makespan of the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$.

Finally, since step 4 of the algorithm returns the scheduling of the minimum makespan constructed in step 3, we conclude that the algorithm returns a scheduling whose makespan is not larger than the makespan of $\Gamma(\mathcal{J})$. □

We analyze the algorithm SCHEDULE-SMALL.

LEMMA 4.3. *Let T_0 be the upper bound used by the algorithm* SCHEDULE-SMALL *on the running time of the rooms. Then the running time of the algorithm* SCHEDULE-SMALL *is bounded by $O(n2^m\lambda_{m,\epsilon}T_0^{\lambda_{m,\epsilon}})$, where $\lambda_{m,\epsilon} = (2j_{m,\epsilon}+1)B_m m$.*

Proof. The number n_S of small jobs in $\mathcal{J}_S$ is bounded by the total number n of jobs in $\mathcal{J}$; each small job may have at most $2^m - 1 \le 2^m$ different modes. Also, as we

Algorithm. SCHEDULE-SMALL
Input: The set $\mathcal{J}_S$ of small jobs and an order π of the large jobs and towers in $\Gamma(\mathcal{J})$.
Output: A scheduling for the job set $\mathcal{J}$.
1. $D[0;0,\ldots,0] = \text{TRUE}$;
2. **for** $i = 1$ **to** n_S **do**
   for each mode Q_{ij} of the small job J'_i with processing time t_{ij}
   for each $D[i-1;\ldots,t_{j,q,r},\ldots] = \text{TRUE}$ such that the job J'_i under mode Q_{ij} is addable to the room $\gamma_{j,q,r}$
   $D[i;\ldots,t_{j,q,r}+t_{ij},\ldots] = \text{TRUE}$;
3. **for** each $D[n_S;\ldots,t_{j,q,r},\ldots] = \text{TRUE}$
   call the list scheduling algorithm based on the order π to construct a scheduling for $\mathcal{J}$ in which the room $\gamma_{j,q,r}$ has running time $t_{j,q,r}$ for all $t_{j,q,r} \geq 0$;
4. return the scheduling constructed in step 3 with the minimum makespan.

FIG. 4.1. *Scheduling small jobs in floors.*

indicated before, the number of rooms is bounded by $\lambda_{m,\epsilon} = (2j_{m,\epsilon}+1)B_m m$. Since the running time for each room is bounded by T_0, for each fixed i, there cannot be more than $T_0^{\lambda_{m,\epsilon}}$ elements $D[i-1;*,\ldots,*]$. Finally, for each $D[i-1;\ldots,t_{j,q,r},\ldots] = \text{TRUE}$, we can check each of the $\lambda_{m,\epsilon}$ component values $t_{j,q,r}$ to decide if the job J'_i under mode Q_{ij} is addable to the room $\gamma_{j,q,r}$. In conclusion, the running time of step 2 in the algorithm SCHEDULE-SMALL is bounded by

$$O(n \cdot 2^m \cdot T_0^{\lambda_{m,\epsilon}} \cdot \lambda_{m,\epsilon}).$$

We will also attach the mode assignment and room assignment of the job J'_i to each element $D[i;\ldots,t_{j,q,r},\ldots] = \text{TRUE}$. With this information, from a given configuration $D[n_S;\cdots,t_{j,q,r},\cdots] = \text{TRUE}$, a corresponding scheduling for the small jobs in the rooms can be easily constructed by backtracking the dynamic programming procedure and its makespan can be computed in time $\lambda_{m,\epsilon}$. Therefore, step 3 of the algorithm takes time

$$O(n \cdot T_0^{\lambda_{m,\epsilon}} \cdot \lambda_{m,\epsilon}).$$

In conclusion, the running time of the algorithm SCHEDULE-SMALL is bounded by $O(n2^m\lambda_{m,\epsilon}T_0^{\lambda_{m,\epsilon}})$, where $\lambda_{m,\epsilon} = (2j_{m,\epsilon}+1)B_m m$. □

We now discuss how an upper bound T_0 for the running time of rooms can be derived. Given an instance $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of the problem $P_m|set|C_{\max}$ and a positive real number $\epsilon > 0$, where each job J_i is specified by a list of alternative processing modes, $J_i = [M_{i1}, \ldots, M_{ip_i}]$ and $M_{ij} = (Q_{ij}, t_{ij})$. Recall that $\min_i = \min\{t_{ij} \mid 1 \leq j \leq p_i\}$. Then the sum $T_0 = \sum_{i=1}^{n} \min_i$ is obviously an upper bound on the makespan of the (m,ϵ)-canonical schedulings for $\mathcal{J}$. (T_0 is the makespan of a straightforward scheduling that assigns each job J_i the mode corresponding to $\min_i$, then starts each job J_i when the previous job J_{i-1} finishes. Therefore, if no (m,ϵ)-canonical scheduling has makespan better than T_0, we simply return this straightforward scheduling.) In particular, the value T_0 is an upper bound for the running time for all rooms. Moreover, since the job set $\mathcal{J}$ takes at least T_0 amount of "work" (the *work* taken by a job is equal to the parallel processing time multiplied by the number of processors involved in this processing) and the system has m processors,

the value T_0 also provides a lower bound for the optimal makespan $Opt(\mathcal{J})$:

$$Opt(\mathcal{J}) \geq T_0/m.$$

In order to apply algorithm SCHEDULE-SMALL, we first need to decide how the set $\mathcal{J}$ is split into large job set $\mathcal{J}_L$ and small job set $\mathcal{J}_S$, what are the modes for the large jobs, what are the types for the towers, and what is the ordering π for the large jobs and towers on which the list scheduling algorithm can be applied. According to Lemma 2.1, the number of large jobs is of form $j_{m,\epsilon} = (3mB_m + 1)^k$ for some integer $k \leq \lfloor m/\epsilon \rfloor$ and by the definition, the number of towers is $2j_{m,\epsilon} + 1$. When m and ϵ are fixed, the number of large jobs and the number of towers are both bounded by a constant. Therefore, we can use any brute force method to exhaustively try all possible cases.

To achieve a polynomial time approximation scheme for the problem $P_m|set|C_{\max}$, we combine the standard scaling techniques [20] with the concept of (m, ϵ)-canonical schedulings as follows.

Let $\mathcal{J} = \{J_1, \ldots, J_n\}$ be an instance of the $P_m|set|C_{\max}$ problem, where $J_i = [M_{i1}, \ldots, M_{ip_i}]$ and $M_{ij} = (Q_{ij}, t_{ij})$. We let $K = \epsilon \cdot T_0/(nm)$ and construct another instance $\mathcal{J}' = \{J'_1, \ldots, J'_n\}$ for the problem, where $J'_i = [M'_{i1}, \ldots, M'_{ip_i}]$ and $M'_{ij} = (Q_{ij}, \lfloor t_{ij}/K \rfloor)$. In other words, the jobs in $\mathcal{J}'$ are identical to those in $\mathcal{J}$ except that all processing times t_{ij} are replaced by $\lfloor t_{ij}/K \rfloor$. We say that the job set $\mathcal{J}'$ is obtained from the job set $\mathcal{J}$ by *scaling the processing times by* K. We apply the algorithm described above to the instance $\mathcal{J}'$ to construct a scheduling for $\mathcal{J}'$ from which a scheduling for $\mathcal{J}$ is induced. The formal algorithm is presented in Figure 4.2.

We explain how step 5 converts the scheduling $\Gamma_0(\mathcal{J}')$ for the job set $\mathcal{J}'$ into a scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$. We first multiply the processing time and the starting time of each job J'_i in the scheduling $\Gamma_0(\mathcal{J}')$ by K (but keeping the processing mode). That is, for the job J'_i of mode Q_{ij} and processing time $\lfloor t_{ij}/K \rfloor$ that starts at time τ_i in $\Gamma_0(\mathcal{J}')$, we replace it by a job J''_i of mode Q_{ij} and processing time $K \cdot \lfloor t_{ij}/K \rfloor$ and let it start at time $K\tau_i$. This is equivalent to proportionally "expanding" the scheduling $\Gamma_0(\mathcal{J}')$ by a factor K. Now on this expansion of the scheduling $\Gamma_0(\mathcal{J}')$, following the order in terms of their finish times, we do "correction" on processing times by increasing the processing time of each job J''_i from $K \cdot \lfloor t_{ij}/K \rfloor$ to t_{ij}. (Note that this increase in processing time may cause many jobs in the scheduling to delay their starting time by $(t_{ij} - K \cdot \lfloor t_{ij}/K \rfloor)$ units. In particular, this increase may cause the makespan of the scheduling to increase by $(t_{ij} - K \cdot \lfloor t_{ij}/K \rfloor)$ units.) After the corrections on the processing time for all jobs in $\mathcal{J}$, we obtain a scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$.

LEMMA 4.4. *For fixed $m \geq 2$ and $\delta > 0$, the running time of the algorithm* APPROX-SCHEME *for the problem $P_m|set|C_{\max}$ is bounded by $O(n^{\lambda_{m,\epsilon}+j_{m,\epsilon}+1})$, where $\epsilon = \delta/2$, $j_{m,\epsilon} \leq (3mB_m + 1)^{\lfloor m/\epsilon \rfloor}$ and $\lambda_{m,\epsilon} = (2j_{m,\epsilon} + 1)B_m m$.*

Proof. Since the integer k is bounded by $\lfloor m/\epsilon \rfloor$, the number j_0 of large jobs in $\mathcal{J}'_L$ is bounded by $j_{m,\epsilon} \leq (3mB_m+1)^{\lfloor m/\epsilon \rfloor}$. Therefore, there are at most $\binom{n}{j_{m,\epsilon}} = O(n^{j_{m,\epsilon}})$ ways to choose the large job set $\mathcal{J}'_L$. Since each job may have up to $2^m - 1 < 2^m$ alternative mode assignments, the total number of mode assignments to each large job set $\mathcal{J}'_L$ is bounded by $(2^m)^{j_{m,\epsilon}} = 2^{mj_{m,\epsilon}}$. Each tower is associated with a subset of the processor set P_m of m processors. Thus, each tower may be associated with $2^m - 1 \leq 2^m$ different subsets of P_m. Therefore, the number of different sequences of up to $2j_{m,\epsilon}+1$ towers is bounded by $(2^m)^{2j_{m,\epsilon}+1} = 2^{2mj_{m,\epsilon}+m}$. Finally, the number of permutations of the j_0 large jobs and $2j_0+1$ towers is $(3j_0+1)!$. Summarizing all these

Algorithm. APPROX-SCHEME
Input: An instance $\mathcal{J}$ for the problem $P_m|set|C_{\max}$ and $\delta > 0$.
Output: A scheduling of $\mathcal{J}$.
1. $\epsilon = \delta/2$; $T_0 = \sum_{i=1}^{n} \min_i$; $K = \epsilon \cdot T_0/(nm)$;
2. let $\mathcal{J}'$ be the job set obtained by scaling the job set $\mathcal{J}$ by K;
3. **for** $k = 0$ **to** $\lfloor m/\epsilon \rfloor$ **do**
   $j_0 = (3mB_m + 1)^k$;
3.1. **for** each subset $\mathcal{J}'_L$ of j_0 jobs in $\mathcal{J}'$
3.2. **for** each mode assignment A to the jobs in $\mathcal{J}'_L$
3.3. **for** each possible sequence of $2j_0 + 1$ towers
3.4. **for** each ordering π of the j_0 jobs in $\mathcal{J}'_L$ and the $2j_0 + 1$ towers
$\mathcal{J}'_S = \mathcal{J}' - \mathcal{J}'_L$;
call SCHEDULE-SMALL on small job set $\mathcal{J}'_S$ and the ordering π to construct a scheduling for the job set $\mathcal{J}'$ (use $T'_0 = \lceil T_0/K \rceil$ as the upper bound for the running time of rooms);
4. let $\Gamma_0(\mathcal{J}')$ be the scheduling constructed in step 3 with the minimum makespan;
5. replace each job J'_i in $\Gamma_0(\mathcal{J}')$ by the corresponding job J_i to obtain a scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$;
6. return the job scheduling $\Gamma_0(\mathcal{J})$.

FIG. 4.2. *The approximation scheme.*

together, we conclude that the number of times that the algorithm SCHEDULE-SMALL is called is bounded by

$$O(\lfloor m/\epsilon \rfloor \cdot n^{j_{m,\epsilon}} \cdot 2^{mj_{m,\epsilon}} \cdot 2^{2mj_{m,\epsilon}+m} \cdot (3j_{m,\epsilon} + 1)!). \tag{4.2}$$

When the algorithm SCHEDULE-SMALL is applied to the job set $\mathcal{J}'_S$, the upper bound on the running time of the rooms is

$$T'_0 = \lceil T_0/K \rceil = \lceil (nm)/\epsilon \rceil \leq (nm)/\epsilon + 1.$$

According to Lemma 4.3, each call to the algorithm SCHEDULE-SMALL takes time

$$O(n2^m \lambda_{m,\epsilon}(T'_0)^{\lambda_{m,\epsilon}}) = O(n2^m \lambda_{m,\epsilon}((nm)/\epsilon + 1)^{\lambda_{m,\epsilon}}), \tag{4.3}$$

where $\lambda_{m,\epsilon} = (2j_{m,\epsilon} + 1)B_m m$.

Combining (4.2) and (4.3) and noting that m and δ thus ϵ are fixed constants, we conclude that the running time of the algorithm APPROX-SCHEME is bounded by $O(n^{\lambda_{m,\epsilon}+j_{m,\epsilon}+1})$. □

Now we are ready to present our main theorem.

THEOREM 4.5. *The algorithm* APPROX-SCHEME *is a polynomial time approximation scheme for the problem* $P_m|set|C_{\max}$.

Proof. As proved in Lemma 4.4, the algorithm APPROX-SCHEME runs in polynomial time when m and δ are fixed constants. Therefore, we need only to show that the makespan of the scheduling $\Gamma_0(\mathcal{J})$ constructed by the algorithm APPROX-SCHEME for an instance $\mathcal{J}$ of the problem $P_m|set|C_{\max}$ is at most $(1 + \delta)$ times the optimal makespan $Opt(\mathcal{J})$ for the instance $\mathcal{J}$. Again let $\epsilon = \delta/2$.

Let $\Gamma(\mathcal{J})$ be an optimal scheduling of makespan $Opt(\mathcal{J})$. Under the scheduling $\Gamma(\mathcal{J})$, the mode assignments of the jobs are fixed. Thus, this particular mode assignment makes us able to split the job set $\mathcal{J}$ into large job set $\mathcal{J}_L$ and small job set $\mathcal{J}_S$ in terms of job processing time. According to Theorem 3.8, there is an (m, ϵ)-canonical

scheduling $\Gamma_1(\mathcal{J})$ for the instance $\mathcal{J}$, under the same mode assignments, such that the makespan of $\Gamma_1(\mathcal{J})$ is bounded by $(1+\epsilon)Opt(\mathcal{J})$.

Consider a room $\gamma_{j,q,r}$ in the (m,ϵ)-canonical scheduling $\Gamma_1(\mathcal{J})$. Suppose that J_{p_1}, ..., J_{p_q} are the small jobs assigned to the room $\gamma_{j,q,r}$ by the scheduling $\Gamma_1(\mathcal{J})$. Then $\sum_{i=1}^{q} t_{p_i} \leq T_0$, where t_{p_i} is the processing time for the job J_{p_i} under $\Gamma_1(\mathcal{J})$, which is the same as under $\Gamma(\mathcal{J})$. Thus we must have

$$\sum_{i=1}^{q} \lfloor t_{p_i}/K \rfloor \leq \sum_{i=1}^{q} t_{p_i}/K \leq T_0/K \leq T_0'.$$

Therefore, under the same mode assignments (with processing time t_{ij} replaced by $\lfloor t_{ij}/K \rfloor$) and the same room assignments, the corresponding scheduling $\Gamma_1(\mathcal{J}')$ for the job set $\mathcal{J}'$ has no rooms with running time exceeding T_0'. Thus, by Lemma 4.1, when step 3 of the algorithm Approx-Scheme loops to the stage in which the large job set and their mode assignments, the tower types, and the ordering of the large jobs and the towers all match that in the scheduling $\Gamma_1(\mathcal{J}')$, the array element $D[n_S; \ldots, t_{j,q,r}, \ldots]$ corresponding to the room configurations of the scheduling $\Gamma_1(\mathcal{J}')$ must have value True. Thus, a scheduling $\Gamma_1'(\mathcal{J}')$ based on this configuration is constructed and its makespan is calculated. Note that the scheduling $\Gamma_1'(\mathcal{J}')$ may not be exactly the scheduling $\Gamma_1(\mathcal{J}')$. However, they must have exactly the same makespan.

Since step 4 of the algorithm Approx-Scheme picks the scheduling $\Gamma_0(\mathcal{J}')$ that has the smallest makespan over all schedulings for $\mathcal{J}'$ constructed in step 3, we conclude that the makespan of the scheduling $\Gamma_0(\mathcal{J}')$ is not larger than the makespan of the scheduling $\Gamma_1'(\mathcal{J}')$, thus not larger than the makespan of the scheduling $\Gamma_1(\mathcal{J}')$.

As we described in the paragraph before Lemma 4.4, to obtain the corresponding scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$, we first expand the scheduling $\Gamma_0(\mathcal{J}')$ by K (i.e., multiplying the job processing times and starting times in $\Gamma_0(\mathcal{J}')$ by K). Let the resulting scheduling be $\Gamma_0(\mathcal{J}'')$. Similarly we expand the scheduling $\Gamma_1(\mathcal{J}')$ by K to obtain a scheduling $\Gamma_1(\mathcal{J}'')$. The makespan of the scheduling $\Gamma_0(\mathcal{J}'')$ is not larger than the makespan of the scheduling $\Gamma_1(\mathcal{J}'')$ since they are obtained by proportionally expanding the schedulings $\Gamma_0(\mathcal{J}')$ and $\Gamma_1(\mathcal{J}')$, respectively, by the same factor K.

Moreover, the makespan of $\Gamma_1(\mathcal{J}'')$ is not larger than the makespan of the (m,ϵ)-canonical scheduling $\Gamma_1(\mathcal{J})$. To see this, observe that these two schedulings use the same large job set under the same mode assignment, the same small job set under the same mode assignment and room assignment, and the same order of large jobs and towers. The only difference is that the processing time t_{ij} of each job J_i in $\Gamma_1(\mathcal{J})$ is replaced by a possibly smaller processing time $K \cdot \lfloor t_{ij}/K \rfloor$ of the corresponding job J_i'' in $\Gamma_1(\mathcal{J}'')$. In consequence, we conclude that the makespan of the scheduling $\Gamma_0(\mathcal{J}'')$ is not larger than the makespan of the (m,ϵ)-canonical scheduling $\Gamma_1(\mathcal{J})$, which is bounded by $(1+\epsilon)Opt(\mathcal{J})$.

Finally, to obtain the scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$, we make corrections on the processing times of the jobs in the scheduling $\Gamma_0(\mathcal{J}'')$. More precisely, we replace the processing time $K \cdot \lfloor t_{ij}/K \rfloor$ for job J_i'' by t_{ij}, which is the processing time of the job J_i in the job set $\mathcal{J}$. Correcting the processing time for each job J_i'' in $\Gamma_0(\mathcal{J}'')$ may make the makespan of the scheduling increase by

$$t_{ij} - K \cdot \lfloor t_{ij}/K \rfloor < K.$$

Therefore, after the corrections of processing times for all jobs in $\mathcal{J}''$, the makespan of the finally resulting scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$, constructed by the algorithm

Approx-Scheme, is bounded by

$$\begin{aligned}\text{the makespan of } \Gamma_1(\mathcal{J}) + n \cdot K &\leq (1+\epsilon)Opt(\mathcal{J}) + \epsilon T_0/m \\ &\leq (1+2\epsilon)Opt(\mathcal{J}) \\ &= (1+\delta)Opt(\mathcal{J}).\end{aligned}$$

Here we have used the fact that $Opt(\mathcal{J}) \geq T_0/m$.

This completes the proof of the theorem. □

5. Conclusion and remarks. In this paper, we have developed a polynomial time approximation scheme for the $P_m|set|C_{\max}$ problem for any fixed constant m. The result is achieved by combinations of the recent techniques developed in the area of multiprocessor job schedulings plus the classical dynamic programming and scaling techniques. This result is a significant improvement over the previous results on the problem: no previous approximation algorithms for the problem $P_m|set|C_{\max}$ have their approximation ratio bounded by a constant that is independent of the number m of processors in the system. Our result also confirms a conjecture made by Amoura et al. [1]. In the following we make a few remarks on further work on the problem.

The multiprocessor job scheduling problem seems an intrinsically difficult problem. For example, if the number m of processors in the system is given as a variable in the input, then the problem becomes highly nonapproximable: there is a constant δ such that no polynomial time approximation algorithm for the problem can have an approximation ratio smaller than n^δ unless P = NP [25]. Observing this plus the difficulties in developing good approximation algorithms for the problem, people had suspected whether the $P_m|set|C_{\max}$ problem for some fixed m should be MAX-NP hard [8]. The present paper completely eliminates this possibility [2].

Our study shows that there are very "normalized" schedulings whose makespan is close to the optimal ones and that these "good" normalized schedulings can be constructed systematically. We are interested in investigating the tradeoff between the degree of this kind of normalization and the time complexity of approximation algorithms.

The current form of our polynomial time approximation scheme may not be practically useful, yet. Even for a small integer m and a reasonably small constant ϵ, the time complexity of our algorithm is bounded by a polynomial of very high degree. More recently, Jansen and Porkolab [21] use the approach of Amoura et al. [1] and are able to develop a linear time approximation scheme for the $P_m|set|C_{\max}$ problem, which still does not seem practical because of the huge constant factor in the complexity of the algorithm.

We are especially interested in developing more practical polynomial time approximation algorithms for systems with small number of processors, such as $P_4|set|C_{\max}$. In particular, we would like to develop practical approximation algorithms for the $P_m|set|C_{\max}$ problem with approximation ratio better than $m/2$, which is still the best known bound for the problem [8]. Some progress has recently been made toward this direction for systems of four processors [19].

Acknowledgment. The authors would like to acknowledge the helpful and stimulating discussions with Nancy Amato, Don Friesen, Klaus Jansen, Chung-Yee Lee, Lorant Porkolab, and Roberto Solis-Oba. Frank Ruskey has helped in the formula for the Bell numbers. The authors would especially like to thank two anonymous referees whose comments and suggestions have greatly improved the presentation of the paper.

REFERENCES

[1] A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, *Scheduling independent multiprocessor tasks*, in Proceedings of the 5th Annual European Symposium on Algorithms, Graz, Austria, Lecture Notes in Comput. Sci. 1284, Springer-Verlag, Berlin, 1997, pp. 1–12.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

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

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

[5] J. Blazewicz, P. Dell'Olmo, M. Drozdowski, and M. Speranza, *Erratum, Scheduling multiprocessor tasks on three dedicated processors*, Inform. Process. Lett., 49 (1994), pp. 269–270.

[6] J. Blazewicz, M. Drozdowski, and J. Weglarz, *Scheduling multiprocessor tasks to minimize scheduling length*, IEEE Trans. Comput., 35 (1986), pp. 389–393.

[7] J. Blazewicz, W. Drozdowski, and J. Weglarz, *Scheduling multiprocessor tasks—a survey*, Internat. J. Microcomput. Appl., 13 (1994), pp. 89–97.

[8] J. Chen and C.-Y. Lee, *General multiprocessor tasks scheduling*, Naval Res. Logist., 46 (1999), pp. 57–74.

[9] J. Chen and A. Miranda, *A polynomial time approximation scheme for general multiprocessor job scheduling*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC'99), Atlanta, 1999, pp. 418–427.

[10] P. Dell'Olmo, M. G. Speranza, and Zs. Tuza, *Efficiency and effectiveness of normal schedules on three dedicated processors*, Discrete Math., 164 (1997), pp. 67–79.

[11] G. Dobson and U. Karmarkar, *Simultaneous resource scheduling to minimize weighted flow times*, Oper. Res., 37 (1989), pp. 592–600.

[12] J. Du and J. Y.-T. Leung, *Complexity of scheduling parallel task systems*, SIAM J. Discrete Math., 2 (1989), pp. 473–487.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[14] M. X. Goemans, *An approximation algorithm for scheduling on three dedicated machines*, Discrete Appl. Math., 61 (1995), pp. 49–59.

[15] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1581.

[16] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1994.

[17] L. A. Hall, *Approximation algorithms for scheduling*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS, Boston, 1997, pp. 1–45.

[18] J. A. Hoogeveen, S. L. van de Velde, and B. Veltman, *Complexity of scheduling multiprocessor tasks with prespecified processor allocations*, Discrete Appl. Math., 55 (1994), pp. 259–272.

[19] J. Huang, J. Chen, and S. Chen, *A simple linear time approximation algorithm for multiprocessor job scheduling on four processors*, in Proceedings of the 11th Annual International Symposium on Algorithms and Computation (ISAAC 2000), Lecture Notes in Comput. Sci. 1969, Springer-Verlag, New York, 2000, pp. 60–71.

[20] O. H. Ibarra and C. E. Kim, *Fast approximation algorithms for the Knapsack and sum of subset problems*, J. Assoc. Comput. Mach., 22 (1975), pp. 463–468.

[21] K. Jansen and L. Porkolab, *General multiprocessor task scheduling: Approximate solutions in linear time*, in Proceedings of the 6th Workshop on Algorithms and Data Structures (WADS '99), Lecture Notes in Comput. Sci. 1663, Springer-Verlag, New York, Berlin, 1999, pp. 110–121.

[22] H. Krawczyk and M. Kubale, *An approximation algorithm for diagnostic test scheduling in multicomputer systems*, IEEE Trans. Comput., 34 (1985), pp. 869–872.

[23] C.-Y. Lee and X. Cai, *Scheduling multiprocessor tasks without prespecified processor allocations*, IIE Transactions, to appear.

[24] C.-Y. Lee, L. Lei, and M. Pinedo, *Current trends in deterministic scheduling*, Ann. Oper. Res., 70 (1997), pp. 1–42.

[25] A. Miranda, *Approximation Algorithms in Multiprocessor Task Scheduling*, Ph.D. thesis, Department of Computer Science, Texas A&M University, College Station, TX, 1998.