Proceedings of the Federated Conference on Computer Science and Information Systems pp. 577–584
DOI: 10.15439/2016F441
ACSIS, Vol. 8. ISSN 2300-5963

Minimizing the Number of Late Multi-Task Jobs on Identical Machines in Parallel

Lingxiang Li
Hunan University of Science and Engineering
Yongzhou
Hunan, P.R.China

Haibing Li
BNP Paribas,
787 Seventh Ave
New York City, NY 10019, USA

Hairong Zhao
Purdue University Northwest
2200 169th Street, IN 46323, USA
Email: hairong@pnw.edu

***Abstract*—We consider the problem of scheduling multi-task jobs on identical machines in parallel. Each multi-task job consists of one or more tasks. Each job has a release date and a due date. A task of a job can be processed by any one of the machines. Multiple machines can process the tasks of a job concurrently. The completion time of a job is the time at which all its individual tasks have been completed. A job is late if it is completed after its due date. We study the problem of minimizing the total number of late jobs. We show that while some special cases are solvable, the general problem is NP-hard and there exists no polynomial time ρ-approximation algorithm, for any $\rho > 1$. We present a general algorithm for the problem and derive from it six heuristics whose performance is evaluated by experimental results.**

I. Introduction

THE PROBLEM under consideration is scheduling multi-task jobs on identical machines in parallel. It can be stated as follows: Assume there are m identical machines and n jobs. Each job j $(j = 1, 2, \dots, n)$, which is available at time r_j and has a due date d_j, consists of k_j $(1 \leq k_j \leq k)$ individual tasks (or operations), where k is the maximum number of tasks that a job may have. Each task $l = 1, 2, \dots, k_j$ of job j, denoted by o_{lj}, can be processed by any one of the machines, and its processing time is denoted by p_{lj}. The individual tasks of a job can be assigned to multiple machines so that they can be processed concurrently. When a machine switches over from one task to another, no setup is required. The completion time of job j, denoted by C_j, is the time at which all individual tasks of job j have been completed. If we let C_{lj} denote the completion time of task o_{lj}, it is clear that $C_j = \max_{1 \leq l \leq k_j} \{C_{lj}\}$. For the ease of description, we also let $P_{i,j}$ and $C_{i,j}$ denote the total processing time and the completion time of job j on machine i, respectively. By definition, $C_j = \max_{1 \leq i \leq m} \{C_{i,j}\}$. A job is late if $C_j > d_j$, and by standard notation, $U_j = 1$ if $C_j > d_j$ and $U_j = 0$ otherwise. We are interested in minimizing the total number of late jobs $\sum U_j$. Let $jbs\ n$ denote n jobs and $tsk\ k$ denote the maximum number of tasks that a job may have. The problem is denoted by $Pm \mid jbs\ n, tsk\ k, r_j \mid \sum U_j$, where m, n, and k can be either fixed or arbitrary. If any of these is not fixed, it is removed from the notation. For example, $P \mid jbs\ 10, tsk \mid \sum U_j$ denotes that m and k are arbitrary but the number of jobs n is 10. If $r_j = 0$ for all jobs, r_j is removed from the notation as well.

The above problem is a more general description of the fully flexible case of customer order scheduling models described in [1], so it is not limited to any specific application contexts, e.g. manufacturing environments. In addition to the application examples surveyed in [1], we yet give another real-life application example in software project management, with the objective to minimize $\sum U_j$. It is not unusual that in a software development team, new projects with various due dates are requested from business lines. A development manager usually creates a parent task for each new project, and creates multiple child tasks (for example, independent modules or loosely coupled modules as a result of well-designed software architecture) associated with the parent task so that multiple software developers in the development team can work on the project simultaneously. A parent task (project) is completed if and only if all child tasks are completed. All software developers (assuming that they have the same skills at the same proficiency levels after certain cross-training) in the team can work on all child tasks. The challenge for the development manager is to find a good schedule for the team, to minimize the number of parent tasks (projects) that cannot be completed before their due dates, so that the relationship and partnership between the development team and the business teams can be positively built up.

Some past work has been done for this problem with the objective to minimize the total weighted completion time $\sum w_j C_j$ and its un-weighted version. Even when $w_j = 1$ for all j, the problem with an arbitrary k is ordinary NP-hard for any fixed $m \geq 2$ and strongly NP-hard when m is arbitrary (see Blocher and Chhajed [2]). On the other hand, when $k = 1$, the problem becomes the classical problem $P \mid\mid \sum w_j C_j$ which is strongly NP-hard, and for any fixed $m \geq 2$, the problem is equivalent to $Pm \mid\mid \sum w_j C_j$ which is ordinary NP-hard (see [3]). In the aspect of algorithms, when $w_j = 1$, Blocher and Chhajed [2] presented six heuristics with empirical analysis of the performance of the heuristics. One of the heuristics was also studied by Yang [4], [5] where it was shown a worst-case performance bound of $7/6$ for $m = 2$. For arbitrary m, two classes of nine heuristics with proven worst-case performance bounds of either $(2-1/m)$ or m were studied by Leung, Li and Pinedo [6].

To the best of our knowledge, no past work has ever been done for this problem with the objective to minimize $\sum U_j$.

In this paper, we are interested in both the complexity and the algorithm aspects of the problem. The remainder of the paper is organized as follows. In Section II, we present some preliminary results regarding some properties of optimal schedules. In Section III, we show that some cases are NP-hard, while some other cases are polynomially solvable. Then, after showing the non-approximability for the general case without release dates, we present in Section IV a general algorithm scheme for the problem and derive from it six heuristics, whose performance is evaluated by experimental results in Section V. Finally, we present conclusions in Section VI.

II. Preliminary Results

We first look into some properties of an optimal schedule for problem $P \mid jbs, tsk \mid \sum U_j$:

Lemma 2.1 (Optimal Property): For problem $P \mid jbs, tsk \mid \sum U_j$, there exists an optimal schedule in which:

a) the tasks (if more than two) of a job are assigned consecutively on each machine;
b) the early jobs are scheduled in nondecreasing order of their due dates on each machine.

Proof: **a)** Suppose that there exists an optimal schedule in which some tasks of job j assigned on machine i are not consecutive, we keep the last task of job j where it is, but make necessary interchange to shift all its other tasks backward so that they become consecutive, thus the completion time of job j remains unchanged. However, other jobs, whose tasks are shift forward due to the interchanges, would be completed earlier. Thus, no new late jobs are introduced, and the resulting schedule remains optimal.

b) Suppose that in an optimal schedule there exist two early jobs, j_1 and j_2, and a machine i, such that $C_{i,j_2} < C_{i,j_1}$ but $d_{j_1} < d_{j_2}$. We assume the tasks of both jobs are scheduled consecutively according to **a**). We remove job j_2, and push forward all jobs before j_1 (and j_1 itself), to fill the hole produced by removing j_2, then place j_2 right after j_1. Clearly, all jobs except j_2 are completed earlier. As for job j_2, its new completion time $C'_{i,j_2} = C_{i,j_1} \leq d_{j_1} < d_{j_2}$. Thus, job j_2 is still completed on time, and the resulting schedule remains optimal. ∎

Note that in an optimal schedule, the tasks of a job are not necessarily to be assigned across all machines. Some machines may be assigned with multiple tasks of the job, while some other machines may not be assigned with any tasks of the same job. To illustrate this, consider an example as follows: $m = 2, n = 2$, $p_{11} = p_{21} = 2, d_1 = 4$, $p_{12} = 4, d_2 = 5$. Clearly, in an optimal schedule for this instance, the only task of job 2 must be assigned to one machine, while the two tasks of job 1 must be assigned to another machine. This yields a schedule with no late jobs.

For any instance I of $P \mid jbs, tsk \mid \sum U_j$, to derive a lower bound for it, we can construct an instance I' of problem $1 \mid\mid \sum U_j$ which can be solved in $O(n \log n)$ by Moore-Hodgson's algorithm [7]: For each job j, construct a job j for I' with processing time $p'_j = \sum_l p_{lj}/m$ and due date $d'_j = d_j$. Let S_{OPT} and S'_{OPT} denote an optimal schedule for instance I and I', respectively. Then, we have the following lower bound which might be useful for the design of a branch-and-bound algorithm, or for evaluating the performance of heuristic algorithms by experimental analysis as what we will show later.

Lemma 2.2 (Lower Bound): For any instance I of problem $P \mid jbs, tsk \mid \sum U_j$, and its corresponding instance I' of problem $1 \mid\mid \sum U_j$ constructed in the way described above, the optimal objective value for I has the following lower bound:

$$\sum_j U_j(S_{OPT}) \geq \sum_j U_j(S'_{OPT}).$$

Proof: Consider an optimal schedule S_{OPT} for instance I, let S_e denote the sub-schedule of all early jobs in S_{OPT}. We construct a schedule S' for instance I' as follows: **a)** For the jobs in S_e, schedule the corresponding jobs of I' in nondecreasing order of d'_j which equals to d_j, let the sub-schedule be S'_e; **b)** Append the rest jobs to the end of S'_e in arbitrary order.

We shall show that all jobs in S'_e are on time as well. Without loss of generality, we assume that the early jobs in S_e are indexed by $1, 2, \ldots, |S_e|$. Consider any partial schedule for jobs $1, 2, \ldots, j^*$ in S_e where $1 \leq j^* \leq |S_e|$. Since job j^* is early, we have $C_{j^*} = \max_{1 \leq i \leq m} \left\{ \sum_{j=1}^{j^*} P_{i,j} \right\} \leq d_{j^*}$. Due to the fact that this partial schedule may not be aligned up at the end of each machine, we have $\max_{1 \leq i \leq m} \left\{ \sum_{j=1}^{j^*} P_{i,j} \right\} \geq \sum_{j=1}^{j^*} \sum_{l=1}^{k_j} p_{lj}/m = \sum_{j=1}^{j^*} p'_j = C'_{j^*}$, it follows that $C'_{j^*} \leq d_{j^*} = d'_{j^*}$, implying that job j^* is early in S'_e. Thus, $\sum_j U_j(S') \leq \sum_j U_j(S_{OPT})$. The lower bound follows due to the fact that $\sum_j U_j(S') \geq \sum_j U_j(S'_{OPT})$. ∎

III. Complexity Results

In this section, we investigate the cases that are either NP-hard or polynomially solvable. The goal is to establish a borderline between the hard cases and the polynomially solvable ones.

A. NP-hard Cases

Before we proceed further, we first introduce the following NP-complete problems (see Garey and Johnson [8]) that will be used for reduction later:

Definition 1 (Partition Problem): Given a list $A = (a_1, a_2, \ldots, a_n)$ of n positive integers, can A be partitioned into two subsets A_1 and A_2 such that $A_1 \cup A_2 = A$ and $\sum_{a_j \in A_1} a_j = \sum_{a_j \in A_2} a_j = B = \frac{1}{2} \sum_{a_j \in A} a_j$?

Definition 2 (3-Partition Problem): Given a list $A = (a_1, a_2, \ldots, a_{3m})$ of $3m$ positive integers such that $\sum_j a_j = mB, B/4 < a_j < B/2$ for each $1 \leq j \leq 3m$, is there a partition A into m subsets $A_1, A_2, \ldots, A_m$ such that $\cup_i^m A_i = A$ and $\sum_{a_j \in A_i} a_j = B$ for each $1 \leq i \leq m$?

Note that even though these two problems are closely related, the **Partition** problem is NP-complete in the ordinary sense, while the **3-Partition** problem is strongly NP-complete.

To show the NP-hardness of several cases, we first start with two restricted cases.

Theorem 3.1: Problem $P \mid jbs, tsk\ 1, d_j = d \mid \sum U_j$ is NP-hard in the strong sense.

Proof: We shall show that the 3-Partition problem is reducible to problem $P \mid jbs, tsk\ 1, d_j = d \mid \sum U_j$. Given an instance of $A = (a_1, a_2, \ldots, a_{3m})$ of 3-Partition, we construct an instance of $P \mid jbs, tsk\ 1, d_j = d \mid \sum U_j$ as follows: There are m machines and $3m$ jobs such that $p_{1j} = a_j$ and $d_j = B$ for each $1 \leq j \leq 3m$. The transformation clearly takes polynomial time. The decision version of the scheduling problem asks if there exists a schedule such that $\sum U_j = 0$?

If the 3-Partition instance has a "Yes" solution, we let the partition be $A_1, A_2, \ldots, A_m$. For each A_i $(1 \leq i \leq m)$, we schedule on machine i the three jobs constructed from the three elements which are in A_i. Thus, we have a schedule such that the finish time on each machine is exactly B, implying that no job is late, i.e., $\sum U_j = 0$.

Conversely, if the scheduling problem instance has a schedule such that $\sum U_j = 0$, it implies that the finish time on each machine has to be exactly B. Due to $p_{1j} = a_j$ and $B/4 < a_j < B/2$, each machine must have 3 jobs exactly, otherwise, the finish time with less/more jobs on a machine would be strictly less/larger than B. Let A_i be the triplet corresponding to the 3 jobs scheduled on each machine $1 \leq i \leq m$, then $A_1, A_2, \ldots, A_m$ is a "Yes" solution to the 3-Partition instance. ■

Theorem 3.2: Problem $Pm \mid jbs, tsk\ 1, d_j = d \mid \sum U_j$ is NP-hard in the ordinary sense for every fixed $m \geq 2$.

Proof: It is sufficient to consider the special case for $m = 2$. We shall show that the Partition problem is reducible to $P2 \mid jbs, tsk\ 1, d_j = d \mid \sum U_j$. Given an instance of the Partition problem, we construct an instance of $P2 \mid jbs, tsk\ 1, d_j = d \mid \sum U_j$ as follows: Let there be n jobs, $p_{1j} = a_j, d_j = B$ for each job $1 \leq j \leq n$. The decision version of the scheduling problem asks if there exists a schedule such that $\sum U_j = 0$?

It is easy to see that the Partition problem instance has a "Yes" solution if and only if the $P2 \mid jbs, tsk\ 1, d_j = d \mid \sum U_j$ instance has a schedule such that $\sum U_j = 0$. ■

Theorem 3.1 and Theorem 3.2 immediately imply the NP-hardness of their general cases, respectively:

Theorem 3.3: As generalization of the cases with common due dates,

a) both problem $P \mid jbs, tsk\ 1 \mid \sum U_j$ and problem $P \mid jbs, tsk \mid \sum U_j$ are strongly NP-hard;
c) both problem $Pm \mid jbs, tsk\ 1 \mid \sum U_j$ and problem $Pm \mid jbs, tsk \mid \sum U_j$ are NP-hard in the ordinary sense for every fixed $m \geq 2$.

On the other hand, when it is restricted to only one job, which has arbitrary number of tasks, the special cases are still NP-hard, as shown below:

Theorem 3.4: Problem $P \mid jbs\ 1, tsk \mid \sum U_j$ is NP-hard in the strong sense.

Proof: We shall show that the 3-Partition problem is also reducible to $P \mid jbs\ 1, tsk \mid \sum U_j$. Given any 3-Partition instance, we construct an instance of the scheduling problem as follows: Let there be 1 job with $3m$ tasks such that $p_{l1} = a_l$ for each $l = 1, 2, \ldots, 3m$; and let $d_1 = B$. The decision version of the scheduling problem asks if there exists a schedule such that $U_1 = 0$?

Similar argument as described in Theorem 3.1 shows that the 3-Partition instance has a "Yes" solution if and only if the scheduling instance has a schedule such that $U_1 = 0$. ■

Theorem 3.5: Problem $Pm \mid jbs\ 1, tsk \mid \sum U_j$ is NP-hard in the ordinary sense for every fixed $m \geq 2$.

Proof: It is sufficient that we consider the special case $m = 2$. Again, a simple reduction from the Partition problem shows that the problem is NP-hard in the ordinary sense for $m = 2$. ■

With the presence of release dates, all the NP-hard cases presented above would be harder. Further, we show that the problem $1 \mid jbs, tsk, r_j \mid \sum U_j$ is NP-hard in the strong sense.

Theorem 3.6: Problem $1 \mid jbs, tsk, r_j \mid \sum U_j$ is NP-hard in the strong sense.

Proof: Since problem $1 \mid jbs, tsk\ 1, r_j \mid \sum U_j$ is equivalent to $1 \mid r_j \mid \sum U_j$ which is strongly NP-hard (due to that $1 \mid r_j \mid \sum L_{max}$ is strongly NP-hard [9] and L_{max} is reducible to U_j [10], [11], [12]), thus its general version $1 \mid jbs, tsk, r_j \mid \sum U_j$ is also NP-hard in the strong sense. ■

B. Polynomially Solvable Cases

We start with the single-machine cases:

Theorem 3.7: The following problems:

a) $1 \mid jbs, tsk\ k \mid \sum U_j$; and
b) $1 \mid jbs, tsk \mid \sum U_j$.

can be solved in polynomial time.

Proof: As a direct result of **a)** in Lemma 2.1, by aggregating the tasks of each job j into a single task with processing time $\sum_l p_{lj}$, both $1 \mid jbs, tsk\ k \mid \sum U_j$ and $1 \mid jbs, tsk \mid \sum U_j$ can be solved in polynomial time by Moore-Hodgson's algorithm [7]. ■

Now we consider a special case in which the tasks of all jobs have identical processing times:

Theorem 3.8: Problem $P \mid jbs, tsk, p_{lj} = p \mid \sum U_j$ can be solved in $O(n \log n + \sum k_j)$ time.

Proof: We can find the optimal schedule in two steps: (1) identify the early job set E; and (2) schedule the early jobs in E. To find the early jobs, we can do the following:

a) Sort and reindex the jobs such that $d_1 \leq d_2 \leq \ldots \leq d_n$.
b) $E = \emptyset$, $sumK = 0$
c) For each $1 \leq j \leq n$
   $E = E \cup \{j\}$, $sumK = sumK + k_j$
   If $\lceil \frac{sumK}{m} \rceil * p > d_j$
      Let $i = \operatorname{argmax}_{x \in E} k_x$
      $E = E \setminus \{i\}$, $sumK = sumK - k_i$

To schedule the early jobs, we simply take the tasks of the early jobs from E in non-decreasing order of the due dates, and assign them one by one to the machines 1, 2, $\cdots$, m, then 1 2, $\cdots$ m again, and so on.

First we show that our schedule is optimal. Without loss of generality, we assume that for any job j, we have $\lceil \frac{k_j}{m} \rceil * p \leq d_j$. Otherwise, it must be late in any schedule.

Notice that in step (1) the jobs are processed in non-decreasing order of their due dates and in (c), if $\lceil \frac{sumK}{m} \rceil * p >$

d_j, we remove the job with the largest number of tasks (thus maximum processing time) from E. This guarantees that if job $j \in E$ after step (1) , then $\lceil \frac{\sum_{i \in E, i \leq j} k_i}{m} \rceil * p \leq d_j$. By the way we schedule the tasks in step (2), we have $C_j = \lceil \frac{\sum_{i \in E, i \leq j} k_i}{m} \rceil * p$, thus $C_j \leq d_j$. So all jobs in E are scheduled on time. Also notice that $|E|$ must be maximum due to the fact that the largest job is chosen to be tardy in (c).

For the time complexity, step (1) can be implemented in $O(n \lg n)$ time if we use priority queue to maintain the jobs in E, and step (2) can be implemented in $\sum k_j$. ■

With the presence of release dates, some special cases are still polynomially solvable. Before we proceed further, we first show that (as we are not aware of any proof in the literature), the classical problem $1 \mid r_j, d_j = d \mid \sum U_j$ can be solved in polynomial time, even though the general case $1 \mid r_j \mid \sum U_j$ is strongly NP-hard.

Theorem 3.9: Problem $1 \mid r_j, d_j = d \mid \sum U_j$ can be solved in $O(n \log n)$ time.

Proof: Consider the following algorithm:

a) For each job with $r_j + p_j > d$, simply mark it as late and exclude it from the next steps.
b) For the remaining jobs, define a new deadline $d'_j = d - r_j$.
c) Treat time d as time 0, and schedule the jobs with new deadlines and release times 0 backwards by applying Hodgson-Moore algorithm.

The correctness of the algorithm lies in the fact that the modified problem is equivalent to the original one and Hodgson-Moore algorithm is optimal for the modified problem. ■

Theorem 3.10: Problem $1 \mid jbs, tsk, r_j, d_j = d \mid \sum U_j$ can be solved in $O(n \log n + \sum k_j)$ time.

Proof: The key observation is that, due to the common due date, there exists an optimal schedule for any problem instance in which the tasks of each early job are scheduled consecutively. Otherwise, shifting the separated tasks (except the first one) of the job forward so that they are consecutive, and shifting the in-between tasks of other jobs backward would not violate their release dates and the common due date, and the schedule remains feasible and optimal. Thus, by aggregating all tasks of each job as a single task with processing time of $\sum_l p_{lj}$, problem $1 \mid jbs, tsk, r_j, d_j = d \mid \sum U_j$ can be polynomially solved by an equivalent $1 \mid r_j, d_j = d \mid \sum U_j$ problem according to Theorem 3.9. Since aggregating the tasks takes $O(\sum k_j)$ time, the algorithm runs in $O(n \log n + \sum k_j)$ time. ■

Theorem 3.11: Problem $1 \mid jbs, tsk, r_j, p_{lj} = 1 \mid \sum U_j$ can be solved in $O(n^5)$ time.

Proof: Consider the following algorithm:

a) For any instance I of problem $1 \mid jbs, tsk, r_j, p_{lj} = 1 \mid \sum U_j$, construct an instance I' of the classical preemptive scheduling problem $1 \mid r_j, pmtn \mid \sum U_j$ with $p'_j = \sum_l p_{lp}, r'_j = r_j, d'_j = d_j$.
b) Solve I' by Lawler's Dynamic Programming algorithm in $O(n^5)$ time [11], [13].
c) Construct the schedule for I exactly from the optimal schedule for I', by mapping the jobs one by one.

Apparently, the obtained schedule is optimal and the running time is dominated by Lawler's algorithm which runs in $O(n^5)$ time. ■

Since the cases with release dates are harder than the ones with no release dates, we focus on the design and analysis of algorithms for the latter cases in the next two sections.

IV. Heuristic Algorithms for Problem $P \mid jbs, tsk \mid \sum U_j$

Due to the strong NP-hardness of problem $P \mid jbs, tsk \mid \sum U_j$, it would be of interest to see if there exists any good approximation algorithm for it. Unfortunately, the following negative result shows that there exists no such approximation algorithm unless $P = NP$.

Theorem 4.1: Unless $P = NP$, there exists no polynomial time ρ-approximation algorithm ($1 < \forall \rho < \infty$) for both problem $P \mid jbs, tsk \mid \sum U_j$ and problem $Pm \mid jbs, tsk \mid \sum U_j$ (for any fixed $m \geq 2$).

Proof: It is sufficient to consider a special case of the problem, i.e., $P \mid jbs\ 1, tsk \mid \sum U_j$, for which we have only 1 job with arbitrary number of tasks to be scheduled on m machines where m is not fixed. Given any problem instance I, since it has only 1 job, being either late or on time, the objective value returned by $\mathcal{A}$ must be $\sum U_j \in \{0, 1\}$. We consider the following four cases:

Case1. The optimal objective value is 0 and $\mathcal{A}$ returns 0, which is optimal.
Case2. The optimal objective value is 1 and $\mathcal{A}$ returns 1, which is optimal.
Case3. The optimal objective value is 0 and $\mathcal{A}$ returns 1. The performance ratio is ∞.
Case4. The optimal objective value is 1 and $\mathcal{A}$ returns 0, which is impossible.

First of all, Case 4 can be excluded for $\mathcal{A}$, since it contradicts to the optimality of the optimal objective value. As for other cases, we claim that there exists at least one problem instance so that Case 3 is true for $\mathcal{A}$. Otherwise, assume that "only" Case 1 and Case 2 are true for $\mathcal{A}$, it simply implies that $\mathcal{A}$ is optimal. Since $\mathcal{A}$ is also polynomial by assumption, it would imply $P = NP$. Therefore, our claim must be true unless $P = NP$, implying that the algorithm $\mathcal{A}$ must be unbounded due to Case 3.

Similar arguments apply to $Pm \mid jbs, tsk \mid \sum U_j$ for its special case when $m = 2$. ■

An observation on the non-approximability of $P \mid jbs\ 1, tsk \mid \sum U_j$ and $Pm \mid jbs\ 1, tsk \mid \sum U_j$ is that, each of them intrinsically consists of a NP-hard subproblem to solve (i.e., the C_{max} problem on parallel machines) and yet its objective value is limited to only two numbers, i.e., either 0 or 1. Thus, there is no much freedom for any algorithm to approximate within.

We first present a general algorithm scheme to identify and schedule a set of early jobs, and then derive some specific algorithms from it by customizing the task sorting criterion and the machine selection criterion.

General-Scheme GS **for** $P \mid jbs, tsk \mid \sum U_j$

Input. A set of n multi-task jobs; the number of machines m.
Output. A set of early jobs E and their schedule S_e.

Sort the n jobs such that $d_1 \leq d_2 \leq \ldots \leq d_n$.
for each job $j = 1, 2, \ldots, n$
  $\ll$***sort its tasks by certain criterion***$\gg$
  assuming the sorted order is o_{1j}, o_{2j}, ..., $o_{k_j j}$

Let $E = \emptyset$, and S_e be an empty schedule
$j = 1$, $firstTry = true$.
While $j \leq n$
  $late = false$
  $l = 1$.
  while ($l \leq k_j$ and $late = false$)
    $\ll$***select machine*** i^* ***by certain criterion***$\gg$
    assign task o_{lj} to machine i^* in S_e.
    if o_{lj} is late
      late = true.
      remove all tasks o_{1j}, o_{2j}, ..., o_{lj} from S_e.
      if $firstTry = true$
        let $j^* = \text{argmax}_{k \in E \cup \{j\}} \{\sum_l p_{lk}\}$
        if $j^* = j$, then $j = j + 1$.
        else
          remove all tasks of j^* from S_e.
          $firstTry = false$.
      else $j = j + 1$, $firstTry = true$.
    else
      if $l = k_j$
        if $firstTry = true$, then $E = E \cup \{j\}$.
        else $E = E \setminus \{j^*\} \cup \{j\}$.
        $j = j + 1$, $firstTry = true$.
      $l = l + 1$.
return S_e

It should be noted that, when $k = 1$ and $m = 1$, the above general algorithm schema would work in the same way as Moore-Hodgson's algorithm for $1 \mid\mid \sum U_j$. Thus, we can regard it as generalization of Moore-Hodgson's algorithm. To derive a specific algorithm from the above general scheme, we need to specify two criteria as marked within $\ll \ldots \gg$. The first criterion to be specified in Step 1 is for sorting the individual tasks of each job $j = 1, 2, \ldots, n$, while the second one in Step 2 is for choosing a machine to process the task under consideration.

Intuitively, we consider two **Task Sorting Criteria**:

- **Arbitrary Order.** No sorting, just keep the original ordering of the tasks as given in input. Thus, it takes no extra running time.
- **Non-increasing Order.** Sort the tasks of job j in non-increasing order of their processing times such that $p_{1j} \geq p_{2j} \geq \ldots p_{k_j j}$ for each $j = 1, 2, \ldots, n$. It takes $O(k \log k)$ time for each job.

To assign a task to a machine, we consider three **Machine Choosing Criteria**:

Smallest Load. Choose the machine with the smallest load. This is used in the well-known Longest Processing Time first rule (LPT) and List Scheduling algorithm (LS) for problem $P \mid\mid C_{max}$ [16].

First Fit. Choose the first machine which can process the task before the job's due date. The First-Fit algorithms were originally designed for the Bin Packing problem [17].

Best Fit. Choose the machine with the largest load but can still process the task before the job's due date.The Best-Fit algorithms were also originally designed for the Bin Packing problem.

TABLE I
SIX ALGORITHMS DERIVED FROM THE TWO TYPES OF CRITERIA

Algorithm	Machine Choosing	Sorting	Task Assignment
GS_{LS}	Smallest Load	Arbitrary	LS
GS_{LPT}	Smallest Load	Non-increasing	LPT
GS_{FF}	First Fit	Arbitrary	First Fit
GS_{FFD}	First Fit	Non-increasing	First Fit Decreasing
GS_{BF}	Best Fit	Arbitrary	Best Fit
GS_{BFD}	Best Fit	Non-increasing	Best Fit Decreasing

Naturally, combination of these two types of criteria produces six different concrete algorithms, which are enumerated in Table I.

In essence, each algorithm derived from the above general algorithm scheme combines the Earliest Due Date first (EDD) rule [18] (at job level), and either an algorithm for problem $P \mid\mid C_{max}$ [16] or an algorithm for the Bin Packing problem [17] (at task level).

Clearly, all algorithms derived from the general scheme run in polynomial time. It is not surprising that, due to Theorem 4.1, the performance ratio of these algorithms is not bounded. To illustrate this by a simple example, we consider the following instance: $n = 1, m = 4, k = 9, p_{11} = 7, p_{21} = 4, p_{31} = p_{41} = 5, p_{51} = p_{61} = 6, p_{71} = 7, p_{81} = p_{91} = 4, d_1 = 12$. The "trial" assignment of tasks by all these six algorithms, as illustrated by Table II, would result in schedules in which the job is late. Thus, all these heuristic algorithms return $\sum_j U_j = 1$. However, in an optimal schedule, as illustrated by the last column in the same table, the job is on time, and the objective value is 0. Thus, the performance ratio of all these heuristic algorithms is ∞ .

Even though the above worst-case example shows that the heuristic algorithms could perform arbitrarily bad, in practice, we expect that their average performance could be much better. To this end, we evaluate these algorithms by experimental results in the next section.

V. EXPERIMENTAL EVALUATION

To evaluate the above heuristic algorithms, we choose the number of jobs $n = 500$, and the number of machines $m = 20$. Problem instances of varying hardness are generated according

TABLE II
TRIAL ASSIGNMENT OF TASKS FOR THE WORST-CASE EXAMPLE

Machine	GS_{LS}	GS_{LPT}	GS_{FF}	GS_{FFD}	GS_{BF}	GS_{BFD}	OPT
1	7, 4, 4	7, 4, 4	7, 4	7, 5	7, 4	7, 5	7, 5
2	4, 6, 4	7, 4	5, 5,	7, 5	5, 5, 4	7, 5	7, 5
3	5, 6	6, 5	6, 6	6, 4	6, 6	6, 4, 4	6, 6
4	5, 7	6, 5	7, 4, 4	6, 4, 4	7, 4	6, 4	4, 4, 4

to different characteristics of the due dates, in a similar way described in Leung, Li and Pinedo [19].

First of all, for each job $j = 1, 2, \ldots, n$, the number of tasks k_j is randomly generated from the uniform distribution $[1, 10m]$. Then, for each task $l = 1, 2, \ldots, k_j$, p_{lj} is generated from the uniform distribution $[1, 100]$. Finally, after all jobs are generated, for each job $j = 1, 2, \ldots, n$, its due date d_j is generated from the following uniform distribution:

$$[P(1 - \delta_1/2 - \delta_2), P(1 + \delta_1/2 - \delta_2)],$$

where

$$P = \sum_{j=1}^{n} \sum_{l=1}^{k_j} p_{lj}/m,$$

and δ_1 and δ_2 determines the range in which the due dates lie and adjusts the tightness of the due dates, respectively. Also, in generating d_j, we ensure that

$$d_j \geq \max\left\{\sum_l p_{lj}/m, \max_l\{p_{lj}\}\right\}.$$

Otherwise, job j would always be late.

We set $\delta_1 = 0.2, 0.4, 0.6, 0.8, 1.0$ and $\delta_2 = 0.2, 0.4, 0.6, 0.8, 1.0$. For each combination of δ_1 and δ_2, 100 instances are generated. Thus, there are 2500 instances in total. The algorithms are implemented in Java. The running environment is Windows 7 64-bit Operating System running on a dual core (2.50GHz + 2.50GHz) PC with 4GB RAM memory.

To compare the algorithms, for each generated instance I_i $(i = 1, 2, \ldots, 100)$, we also construct the corresponding single-machine instance I'_i as described in Lemma 2.2. The instance I'_i is solved optimally by Moore-Hodgson's algorithm, and then the result, denoted by $LB(I'_i)$, is used as a reference objective value (lower bound) to evaluate the objective value produced by a heuristic algorithm A for I_i, denoted by $\sum_j U_j(A, I_i)$. Table III shows the collective results for all six algorithms. Each algorithm A has two columns, namely $\overline{\varepsilon}$ and $\overline{t}$, which are defined as follows. For each setting of δ_1 and δ_2, $\overline{\varepsilon}$ is defined for A as:

$$\overline{\varepsilon} = \frac{1}{100}\sum_{i=1}^{100}\left(\sum_j U_j(A, I_i) - LB(I'_i)\right);$$

and let $t(A, I_i)$ denote the running time (in milliseconds) of algorithm A on solving instance I_i, $\overline{t}$ is defined for A as:

$$\overline{t} = \frac{1}{100}\sum_{i=1}^{100} t(A, I_i).$$

From the table, we have the following findings:

- The objective values produced by all six algorithms are actually very close to the lower bound values, the gaps are mostly less than 3, which means that the algorithms perform close to an optimal algorithm for these randomly generated instances.
- The frequencies that the six algorithms achieve the lowest $\overline{\varepsilon}$ are $(5, 9 \mid 11, 14 \mid 13, 22)$ corresponding to their order listed in the table. Thus, in terms of Machine Choosing Criterion, the algorithms based on Best-Fit criterion performs better than those based on First-Fit criterion, which in turn are better than those based on Smallest-Load criterion. In terms of Task Sorting Criterion, the algorithms based on non-increasing task sorting criterion perform better than those without task sorting.
- In terms of running time, the First-Fit based algorithms run faster than those based on Best-Fit criterion, which in turn run faster than algorithms based on Smallest-Load criterion.
- Interestingly, task sorting in initialization actually does not increase the running time, but helps reduce the running time. This could be due to that task sorting helps produce better results and hence results in less iterations for Step 2 and Step 3.
- Regarding the sensitivity of algorithms' performance to the hardness of problem instances, overall, $\overline{\varepsilon}$ increases when δ_2 increases. The explanation is that higher δ_2 results in tighter due dates generated for the instances. Hence, the number of late jobs is expected to be higher, and the gap between the heuristic result and lower bound result is expected to increase accordingly. On the other hand, $\overline{t}$ also increases when δ_2 increases. Indeed, when the number of late jobs increases with higher δ_2, more iterations are required by Step 2 and Step 3 of the algorithms.

The above findings are sufficient to give us an overview of the performance of the algorithms and provide guidelines for us to choose the best ones among them for practical use. Considering both solution quality and running time, we recommend that algorithm GS_{BFD} is the best choice.

VI. CONCLUSIONS

In this paper, we studied the problem of minimizing the total number of late multi-task jobs on identical and flexible machines in parallel. We first investigated the complexity aspect of the problem. As summarized in Table IV, complexity

TABLE III
COMPARISON OF THE SIX ALGORITHMS IN TERM OF $\overline{\varepsilon}$ AND $\overline{t}$

		GS_{LS}		GS_{LPT}		GS_{FF}		GS_{FFD}		GS_{BF}		GS_{BFD}	
δ_1	δ_2	$\overline{\varepsilon}$	$\overline{t}$	$\overline{\varepsilon}$	$\overline{t}$	$\overline{\varepsilon}$	$\overline{t}$	$\overline{\varepsilon}$	$\overline{t}$	$\overline{\varepsilon}$	$\overline{t}$	$\overline{\varepsilon}$	$\overline{t}$
0.2	0.2	0.56	60	0.47	55	0.46	40	0.45	40	0.44	41	0.45	40
0.2	0.4	0.57	170	0.46	155	0.46	121	0.47	115	0.47	123	0.47	118
0.2	0.6	0.65	283	0.57	258	0.56	210	0.56	199	0.56	215	0.56	204
0.2	0.8	0.65	360	0.51	331	0.51	277	0.51	264	0.51	285	0.51	271
0.2	1.0	3.03	379	2.39	352	1.77	295	1.6	284	1.53	304	1.51	292
0.4	0.2	0.83	389	0.83	365	0.83	302	0.83	296	0.83	312	0.83	304
0.4	0.4	0.67	479	0.53	446	0.52	365	0.52	356	0.52	377	0.52	366
0.4	0.6	0.53	583	0.41	542	0.41	446	0.41	436	0.41	461	0.41	447
0.4	0.8	0.82	657	0.51	612	0.51	508	0.47	501	0.48	526	0.47	513
0.4	1.0	3.03	680	2.54	637	1.83	530	1.65	526	1.53	550	1.5	538
0.6	0.2	0.0	687	0.0	648	0.0	536	0.0	536	0.0	556	0.0	549
0.6	0.4	0.57	745	0.5	703	0.5	576	0.5	577	0.5	598	0.5	591
0.6	0.6	0.58	835	0.49	785	0.49	643	0.48	647	0.48	667	0.48	662
0.6	0.8	2.05	888	1.51	836	1.21	687	1.05	695	1.06	713	1.01	710
0.6	1.0	3.14	911	2.71	862	2.06	709	1.88	719	1.72	736	1.69	735
0.8	0.2	0.0	919	0.0	873	0.0	715	0.0	729	0.0	743	0.0	746
0.8	0.4	0.62	931	0.55	888	0.54	724	0.53	742	0.53	753	0.53	759
0.8	0.6	0.73	987	0.5	942	0.5	766	0.5	789	0.5	797	0.49	807
0.8	0.8	2.22	1018	1.63	974	1.3	791	1.19	818	1.16	824	1.13	836
0.8	1.0	2.95	1036	2.42	994	1.88	807	1.71	837	1.62	841	1.61	856
1.0	0.2	0.0	1043	0.0	1005	0.0	813	0.0	847	0.0	848	0.0	867
1.0	0.4	0.0	1050	0.0	1016	0.0	819	0.0	857	0.0	855	0.0	878
1.0	0.6	1.37	1061	0.9	1031	0.68	829	0.55	870	0.62	865	0.57	892
1.0	0.8	2.24	1072	1.64	1045	1.18	838	1.08	883	1.05	876	1.02	906
1.0	1.0	2.89	1083	2.34	1058	1.79	847	1.67	896	1.53	886	1.49	919

results were established for some cases that are either NP-hard or polynomially solvable. Due to the NP-hardness of the general case, we then investigated its approximability. Unfortunately, the result was negative, as we showed that, unless $P = NP$, there exists no ρ-approximation algorithm $(1 < \forall\rho < \infty)$ even for the case with no release dates. Thus, we designed a general algorithm scheme and derived from it six heuristic algorithms whose performance was evaluated by experimental results. The findings from the experimental results provided guidelines for choosing the best algorithm among them for practical use, and we recommended algorithm GS_{BFD} as the best choice.

We did not consider setup times, preemption and weights for the problem. It will be interesting to study the problem with these additional constraints. Even for release dates, we only considered the single machine cases. Hopefully, the heuristics presented in this paper can be extended to the parallel machine cases with release dates. We did not consider an exact algorithm in this paper either. It seemed that the design of an exact algorithm with intelligent search of an optimal solution is not trivial at all, even though we looked into some properties and derived a lower bound for optimal schedule. Indeed, although it has been shown that there exists an optimal schedule which complies with the EDD rule. However, the subproblem to assign the individual tasks to the parallel machines is NP-hard. This not only makes it hard for the design of an exact algorithm with intelligent search, but also makes it non-trivial for the design of effective local search heuristics or meta-heuristics. All of these are worthy of further research for the problem.

REFERENCES

[1] J.-T. Leung, H. Li, and M. Pinedo, "Order scheduling models: an overview," in *Multidisciplinary Scheduling: Theory and Applications*, G. Kendall, E. K. Burke, S. Petrovic, and M. Gendreau, Eds. Springer, 2005, pp. 37–53, http://dx.doi.org/10.1007/0-387-27744-7_3.

[2] J. Blocher and D. Chhajed, "The customer order lead-time problem on parallel machines," *Naval Research Logistics*, vol. 43, pp. 629–654, 1996, http://dx.doi.org/10.1002/(SICI)1520-6750(199608)43:5<629::AID-NAV3>3.0.CO;2-7.

[3] J. Bruno, E. Coffman, and R. Sethi, "Scheduling independent tasks to reduce mean finishing time," *Communications of the ACM*, vol. 17, no. 7, pp. 382–387, 1974, http://dx.doi.org/10.1145/361011.361064.

[4] J. Yang, "Scheduling with batch objectives," Ph.D. dissertation, Industrial and Systems Engineering Graduate Program, The Ohio State University, Columbus, Ohio, 1998.

[5] J. Yang and M. Posner, "Scheduling parallel machines for the customer order problem," *Journal of Scheduling*, vol. 8, no. 1, pp. 49–74, 2005, http://dx.doi.org/10.1007/s10951-005-5315-5.

[6] J.-T. Leung, H. Li, and M. Pinedo, "Approximation algorithms for minimizing total weighted completion time of orders on identical machines in parallel," *Naval Research Logistics*, vol. 53, no. 4, pp. 243–260, 2006, http://dx.doi.org/10.1002/nav.20138.

[7] J. Moore, "An n job, one machine sequencing algorithm for minimizing the number of late jobs," *Management Science*, vol. 15, pp. 102–109, 1968, http://dx.doi.org/10.1287/mnsc.15.1.102.

[8] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. New York: W.H.Freeman, 1979.

[9] J. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977, http://dx.doi.org/10.1016/S0167-5060(08)70743-X.

[10] E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys, "Sequencing and scheduling: algorithms and complexity," in *Handbooks in Operations Research and Management Science*, 1993, pp. 445–522.

[11] P. Brucker, *Scheduling Algorithms, Fifth Edition*. Berlin: Springer, 2007.

[12] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer, 2008.

[13] E. Lawler, "A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs," *An-*

TABLE IV
COMPLEXITY RESULTS

Problem	Complexity
$1 \mid jbs, tsk, r_j \mid \sum U_j$	NP-hard in the strong sense
$Pm \mid jbs, tsk \mid \sum U_j$	NP-hard in the ordinary sense for $m \geq 2$
$P \mid jbs, tsk \mid \sum U_j$	NP-hard in the strong sense
$1 \mid jbs, tsk \mid \sum U_j$	Solvable by Moore-Hodgson's algorithm
$1 \mid jbs, tsk, r_j, d_j = d \mid \sum U_j$	Solvable in $O(n \log n + \sum k_j)$ time
$1 \mid jbs, tsk, r_j, p_{lj} = 1 \mid \sum U_j$	Solvable by Lawler's algorithm in $O(n^5)$ time
$P \mid jbs, tsk, p_{lj} = p \mid \sum U_j$	Solvable in $O(n \log(n) + \sum k_j)$ time

nals of Operations Research, vol. 26, no. 1, pp. 125–133, 1990, http://dx.doi.org/10.1007/BF02248588.

[14] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[15] C. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," in *Journal of Computer and System Sciences*, Vol. 43(3), pp. 425–440, 1991, http://dx.doi.org/10.1016/0022-0000(91)90023-X.

[16] R. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal of Applied Mathematics*, vol. 17, pp. 263–269, 1969, http://dx.doi.org/10.1137/0117039.

[17] E. Coffman, M. Garey, and D. Johnson, "Approximation algorithms for bin packing: a survey," in *Approximation Algorithms for NP-hard Problems*, D. Hochbaum, Ed. PWS Publishers, 1997, pp. 46–93.

[18] J. Jackson, "Scheduling a production line to minimize maximum tardiness," Management Science Research Project, UCLA, Tech. Rep. 43, 1955.

[19] J.-T. Leung, H. Li, and M. Pinedo, "Scheduling orders for multiple product types with due date related objectives," *European Journal of Operational Research*, vol. 168, no. 2, pp. 370–389, 2006, http://dx.doi.org/10.1016/j.ejor.2004.03.030.