DOI: 10.15439/2020F33
ISSN 2300-5963 ACSIS, Vol. 21

Multiprocessor Scheduling Problem with Release and Delivery Times

Natalia Grigoreva
St.Petersburg State University
Universitetskay nab. 7/9, St.Petersburg, Russia
Email: n.s.grig@gmail.com

***Abstract*—The multiprocessor scheduling problem is defined as follows: set of jobs have to be executed on parallel identical processors. For each job we know release time, processing time and delivery time. At most one job can be performed on every processor at a time, but all jobs may be simultaneously delivered. Preemption on processors is not allowed. The goal is to minimize the time, by which all tasks are delivered. Scheduling tasks among parallel processors is a NP-hard problem in the strong sense. The best known approximation algorithm is Jackson's algorithm, which generates the list schedule by selecting the ready job with the largest delivery time. This algorithm generates no delay schedules. We define an IIT (inserted idle time) schedule as a feasible schedule in which a processor can be idle at a time when it could begin performing a ready job. The paper proposes the approximation inserted idle time algorithm for the multiprocessor scheduling. We proved that deviation of this algorithm from the optimum is smaller then twice the largest processing time. To illustrate the efficiency of our approach we compared two algorithms on randomly generated sets of jobs.**

I. Introduction

We consider the problem of scheduling jobs with release and delivery times on parallel identical processors.

We consider a set of jobs $U = \{u_1, u_2, \dots, u_n\}$. For each job we know its processing time $t(u_i)$, its release time $r(u_i)$ the time at which the job is ready for performing and its delivery time $q(u_i)$. All data are integer. Set of jobs is performed on m parallel identical processors. Any processor can run any job and it can perform no more than one job at a time. Preemption is not allowed. The schedule defines the start time $\tau(u_i)$ of each job $u_i \in U$. The makespan of the schedule S is the quantity

$$C_{\max} = \max\{\tau(u_i) + t(u_i) + q(u_i) | u_i \in U\}.$$

The goal is to minimize $C_{\max}$, the time by which all jobs are delivered. Following the classification scheme proposed by Graham *et al.* [12], this problem is denoted by $P|r_i, q_i|C_{max}$.

The problem is equivalent to model $P|r_i|L_{\max}$ with due dates $d(u_i)$, rather than delivery times $q(u_i)$. The equivalence is shown by replacing each delivery time $q(u_i)$ by due date $d(u_i) = q_{\max} - q(u_i)$, where $q_{\max} = \max\{q(u_i) \mid u_i \in U\}$. In this problem the objective is to minimize the maximum lateness of jobs $L_{\max} = \max\{\tau(u_i) + t(u_i) - d(u_i) | u_i \in U\}$.

This problem relates to the scheduling problem [3], very similar problems can arise in different application fields [23]. The problem plays the main role in some important applications, for example, in the Resource Constrained Project Scheduling Problem [3], and it is NP-hard [27].

The single machine problem with release and delivery times is denoted by $1|r_j, q_j|C_{max}$ and it is NP-hard too [27]. The $1|r_j|q_j|C_{\max}$ is also a main component of several more complex scheduling problems, such that flowshop and jobshop scheduling [1], [8] and uses in real industrial application [8]. The problem $1|r_j, q_j|C_{\max}$ has been studied by many researches [5], [16], [22], [26].

The problem $P|r_j, q_j|C_{\max}$ is a generalization of the single-machine scheduling problem with release and delivery times $1|r_j, q_j|C_{\max}$. The problem arises as a strong relaxation of the multiprocessor flow shop problem [4]. The problem has been the subject of numerous papers, some of these works focus on problems with a precedence constrains [29].

Most of these studies have focused to obtain lower bounds [6], [18], the development of exact solution of the problem [7], [8] or a polynomial time approximation scheme (PTAS) [17], [21].

However, despite its practical importance, only Jackson's algorithm is used as a simple list heuristic algorithm for the $P|r_j, q_j|C_{\max}$.

The worst-case performance of Jackson's algorithm has been investigated by Gusfield [15] and Carlier [7]. Gusfield [15] examined Jackson's heuristic for the problem to minimize the maximum lateness of jobs with release times and due dates and proved that difference between the lateness given by Jackson's algorithm and the optimal lateness is bounded by $(2m-1)t_{max}/m$ and this bound is tight.

Carlier [7] proved that $C_{\max} - C_{opt} \leq 2t_{\max} - 2$, where $C_{\max}$ is the objective function of Jackson's rule schedule, and C_{opt} is the optimal makespan.

Gharbi and Haouari [11] proposed improved Jackson's algorithm which uses an $O(n \log n)$-time preprocessing procedure in order to reduce the number of jobs to be scheduled and investigated its worst-case performance.

The preprocessing procedure can be briefly described as follows. Let $j(k)$ is the job with the kth smallest release time. A condition which allows to define the start time of a job $j_0 \in \{j_1, j_2, ..., j_m\}$ at $r(j_0)$ in an optimal schedule is $r(j_0) + t(j_0) = \min\{r(j_k) + t(j_k) | k \in 1..m\} \leq r(j_{m+1})$. Then a job j_0 can be deleted from the set of jobs. This deleting rule is recursively applied to the new jobset $U \setminus \{j_0\}$. Let U_r be the set of jobs deleted according to this rule. Then the above deleting rule can be applied to the reversing problem (where by reversing the roles of the release and delivery times). Let U_q be the set of jobs deleted according to this second rule.

Therefore, the problem can be solved on a reduced job-set, denoted by UJ. Let S_{UJ} is a feasible schedule with makespan equal to $C_{max}(S_{UJ})$. Then the improved Jackson's algorithm constructs a complete schedule with makespan equal to $C_{\max} = \max\{C_{\max}(S_{UJ}), \max(r_j + t_j + q_j | j \in U_r \cup U_q)\}$.

Most of research in scheduling is devoted to the development of nondelay schedule. A nondelay schedule has been defined by Baker[2] as a feasible schedule in which a processor cannot be idle at a time when it could start performing a ready job. Kanet and Sridharam [19] defined an inserted idle time schedule (IIT)as a feasible schedule in which a processor can idle, if there is the ready job and reviewed the literature with problem setting where IIT scheduling may be required. Most of papers considered problem with single processor. It is known that an optimal schedule can be IIT schedule. Therefore,it is important to develop algorithms that can build IIT schedule.

In [13] we considered multiprocessor scheduling problem with precedence constrained and proposed the branch and bound algorithm, which use an inserted idle time algorithm for m parallel identical processors.

In [14] we investigated the inserted idle time algorithm for single machine scheduling with release times and due dates.

The goal of this paper is to propose an approximation IIT algorithm for $P|r_j, q_j|C_{max}$ problem and investigate its worst-case performance.

In order to confirm the effectiveness of our approach we tested our algorithms on randomly generated examples.

First in section 2, we propose an approximation IIT algorithm named MDT/IIT (maximum delivery time/ inserted idle time). In section 3 we investigate the worst-case performance of MDT/IIT algorithm. In section 4 we present the results of testing the algorithm. Summary of this paper is in section 5.

II. Approximation algorithm MDT/IIT

Algorithm MDT/IIT generates the schedule, in which a processor can be idle at the time when it could begin performing a job.

Let $r_{\min} = \min\{r(i) \mid i \in U\}$ and $q_{\min} = \min\{q(i) \mid i \in U\}$.

First we calculate the lower bound LB of the optimal makespan [7]:
$LB = \max\{r_{\min} + \sum_{i=1}^{n} t(i)/m + q_{\min}, \max\{r(i) + t(i) + q(i) \mid i \in U\}\}$.

Let $t_{\max} = \max\{t(i) \mid i \in U\}$.

Let a partial schedule S_k have been constructed, where k is the number of scheduling jobs. Let $C_{\max}(S_k))$ be the makespan of S_k.

Let $time_k[i]$ be the time of the termination of the processor i after completion all its jobs.

Procedure $SET(i, j, k, C_{\max}(S_k))$ sets a job j on processor i at step k and include the job j in S_k.

$SET(i, j, k, C_{\max}(S_k))$.

1) $\tau(j) := \max\{time_k[i], r(j)\}$.
2) $k := k + 1$.
3) $time_k[i] := \tau(j) + t(j)$.
4) $C_{\max}(S_k) := \max\{C_{\max}(S_{k-1}), \tau(j) + t(j) = q(j)$.

The approximation schedule S is constructed by MDT/IIT algorithm as follows:

1) Determine the processor l_0 such that
$$t_{\min}(l_0) = \min\{time_k[i] | i \in 1..m\}.$$
2) If there is no job u_i, such that $r(u_i) \le t_{\min}(l_0)$ then $t_{\min}(l_0) := \min\{r(u_i) \; |u_i \notin S_k\}$.
3) Select a job u with the largest delivery time $q(u) = \max\{q(u_i) \mid r(u_i) \le t_{\min}(l_0)\}$.
4) If $t_{\min}(l_0) > t_{\max}$ then $SET(l_0, u, k, C_{\max}(S_k))$; go to 11.
5) Select a job u^* such that $q(u^*) = \max\{q(u_i) \mid t_{\min}(l_0) < r(u_i) < t_{\min}(l_0) + t(u)\}$.
6) If there is no such job u^* or one of inequality is hold $q(u) \ge q(u^*)$ or $q(u^*) \le LB/3$, or $r(u^*) \ge t_{\max}$ then $SET(l_0, u, k, C_{\max}(S_k))$. Go to 11.
7) Calculate the idle time of the processor l_0 before the start of job u^* $idproc(l_0) = r(u^*) - t_{\min}(l_0)$. If $q(u^*) - q(u) < idproc(l_0)$, then $SET(l_0, u, k, C_{\max}(S_k))$. Go to 11.
8) Select a job u_1 which can be executed during the time interval $[t_{\min}(l_0), r(u^*)]$, namely such that $q(u_1) = \max\{q(u_i) \mid t_{\min}(l_0) \ge r(u_i) \; \& \; t(u_i) \le idle(u^*)\}$. If job u_1 exists, then $SET(l_0, u1, k, C_{\max}(S_k))$. Go to 11.
9) Select the ready job u_2 such that $q(u_2) = \max\{q(u_i) \mid t_{\min}(l_0) < r(u_i) \; \& \; r(u_i) + t(u_i) \le r(u^*)\}$. If we find u_2, then $SET(l_0, u2, k, C_{\max}(S_k))$. Go to 11.
10) $SET(l_0, u^*, k, C_{\max}(S_k))$.
11) If $k < n$, then go to 1.
12) If $k = n$, we construct the approximation schedule $S = S_n$ and we have the objective function $C_{\max}(S) = C_{\max}(S_n)$.

The algorithm sets on the processor l_0 the job u^* with the largest delivery time $q(u^*)$. If job u^* is not ready, then the processor l_0 does not work in the interval $[t_1, t_2]$, where $t_1 = t_{\min}(l_0)$, $t_2 = r(u^*)$.

In order to avoid too much idle of the processor the inequality $q(u^*) - q(u) \ge idproc(l_0)$ is verified on step 7 and if it is hold, we select job u^*. In order to use the idle time of the processor l_0 we look for job u_1 or u_2 to perform in this interval (see steps 8 and 9). Job u^* starts at $\tau(u^*) = r(u^*)$.

The MDT/IIT algorithm generates the schedule in $O(mn^2)$ times. It generates the schedule by n iterations, the processor selection requires $O(m)$ times and the job selection requires $O(n)$ time on each iteration.

III. Property of MDT/IIT algorithm

Let algorithm generate a schedule S, and for each job j we have the start time $\tau(j)$. The makespan is $C_{\max}(S) = \max\{\tau(j) + t(j) + q(j) \mid j \in U\}$.

Definition 3.1:

Critical job j_c is the first processed job such that $C_{\max}(S) = \tau(j_c) + t(j_c) + q(j_c)$.

Let C_{opt} be the length of an optimal schedule.

Theorem 3.2: $C_{\max}(S) - C_{opt} < t_{\max}(2m-1)/m$, and this bound is tight.

Proof:

Let c be the critical job then $C_{\max}(S) = \tau(c) + t(c) + q(c)$. If the processors do not idle in the time interval $[0, \tau(c)]$, then we set $\tau^* = 0$, else let

$$\tau^* = \max\{t \mid 0 < t < \tau(c)\},$$

where t is the time, when the number of processors working from time $t-1$ to t is smaller then m.

Let $J = \{v_i \in U | \tau^* \le \tau(v_i) < \tau(c)\}$ be the set of jobs, which begin in interval $[\tau^*, \tau(c))$.

Let $\tau(j_0) = \max\{\tau(v_i) | \tau(j_0) < \tau(c) \ \& \ q(v_i) < q(c)\}$. The job j_0 is the last scheduling job with $q(j_0) < q(c)$ and $\tau(j_0) < \tau(c)$.

If there is no such work j_0, then we set $\tau(j_0) = 0$.

We consider four cases.

Case 1. There is not any idle time of processors before $\tau(c)$ and then $\tau^* = 0$.

Let $\tau(j_0) = 0$, then all jobs, which start time $\tau(v_i) < \tau(c)$, have delivery time $q(v_i) \ge q(c)$. The jobs from J must start in interval $[0, \tau_c)$, then

$$\sum_{v_i \in J} t(v_i) \ge m\tau(c)$$

and

$$C_{opt} \ge \sum_{v_i \in J} t(v_i)/m + t(c)/m + q(c) \ge \tau(c) + t(c)/m + q(c).$$

Then

$$C_{\max}(S) - C_{opt} \le t(c) - t(c)/m < t_{\max}$$

.

Case 2. Let $0 \le \tau(j_0) < \tau^* < t_{\max}$.

Then $q(v_i) \ge q(c), \forall v_i \in J$.

We can consider three sets of jobs:

$A_1 = \{v_i \in J | r(v_i) \ge \tau^*\}$, the jobs can start in interval $[\tau^*, \tau_c)$,

$A_2 = \{v_i \in J | r(v_i) < \tau^*\}$, the jobs can start before τ^*,

$A_3 = \{v_i \in U | \tau(v_i) \le \tau^* - 1 \ \& \ \tau(v_i) + t(v_i) \ge \tau^*\}$. A_3 contains not more $m-1$ jobs and this jobs process in the interval $[\tau^* - 1, \tau^*]$. There are no any idle time of processors in the interval $[\tau^*, \tau(c)]$, then

$$T_A = \sum_{v_i \in A_3} (t(v_i) - 1) + \sum_{v_i \in A_1} t(v_i) + + \sum_{v_i \in A_2} t(v_i) \ge m(\tau(c) - \tau^*).$$

The jobs from set A_1 can process only after the time τ^*, but the jobs from sets A_2 and A_3 can process before τ^*. The job c can process before τ^*, if $r(c) < \tau^*$.

$$C_{opt} \ge (T_A + t(c))/m + q(c) \ge \tau(c) - \tau^* + t(c)/m + q(c).$$

Hence

$$C_{\max}(S) - C_{opt} \le \tau^* + t(c) - t(c)/m < t_{\max}(2 - 1/m),$$

because $\tau^* < t_{max}$ (see step 3 of MDT/IIT algorithm).

Case 3. Let $t_{\max} \le \tau^*$ and $\tau(j_0) < \tau^*$.

If $t_{\max} \le \tau^*$ then $A_2 = \emptyset$ and the job c can process only after τ^*. Then

$$\sum_{v_i \in A_3} (t(v_i) - 1) + \sum_{v_i \in A_1} t(v_i) \ge m(\tau(c) - \tau^*).$$

$$C_{opt} \ge \tau^* + \sum_{v_i \in A_1} t(v_i)/m + t(c)/m + q(c) \ge$$

$$\ge \tau(c) - \sum_{v_i \in A_3} (t(v_i) - 1)/m + t(c)/m + q(c)$$

A_3 contains not more $m-1$ jobs, hence

$$C_{\max}(S) - C_{opt} \le t(c) - t(c)/m + 1/m \sum_{v_i \in A_3} (t(v_i) - 1) \le$$

$$\le t(c) - t(c)/m + (m-1)/m(t_{\max} - 1) \le$$

$$\le (2t_{\max} - 1)(m-1)/m < t_{\max}(2 - 1/m).$$

Case 4. Consider the case $0 \le \tau^* \le \tau(j_0)$.

Let $J = \{v_i \in U | \tau(j_0) < \tau(v_i) < \tau(c)\}$.

For all $v_i \in J$ it is true, that $r(v_i) > \tau(j_0)$, otherwise the processor must process job v_i instead of j_0. $q(v_i) \ge q(c)$.

Then

$C_{opt} \ge \tau(j_0) + 1 + \sum_{v_i \in J} t(v_i)/m + t(c)/m + q(c)$.

We can see the set of jobs:

$A_3 = \{v_i \in U | \tau(v_i) \le \tau(j_0) \ \& \ \tau(v_i) + t(v_i) \ge \tau(j_0) + 1\}$, the jobs must process in interval $[\tau(j_0), \tau(j_0)+1]$. A_3 contains m jobs. Then

$$\sum_{v_i \in A_3} (t(v_i) - 1) + \sum_{v_i \in J} t(v_i) \ge m(\tau(c) - \tau(j_0) - 1).$$

$$C_{opt} \ge \tau(j_0) + 1 + \tau(c) - \tau(j_0) - 1 - 1/m \sum_{v_i \in A_3} (t(v_i) - 1) +$$

$$+ t(c)/m + q(c) =$$

$$= \tau(c) + t(c)/m + q(c) - 1/m \sum_{v_i \in A_3} (t(v_i) - 1).$$

A_3 contains m jobs, hence

$$C_{\max}(S) - C_{opt} \le 1/m \sum_{v_i \in A_3} (t(v_i) - 1) + t(c)(m-1)/m \le$$

$$\le t_{max} - 1 + t_{\max}(m-1)/m$$

TABLE I
MDT SCHEDULE $C_{\max}(MDT) = 5m - 2$

t	$m-1$	$m-1$		m	m	m
$P1$	idle	u_1	u_4	a	v_3	$v6$
$P2$	idle	u_2	u_5	v_1	v_4	idle
$P3$	idle	u_3	u_6	v_2	v_5	idle

TABLE II
OPTIMAL SCHEDULE $C_{\max} = 3m$

t	m	m			m
$P1$	v_3	a			v_6
$P2$	v_1	u_1	u_4	u_3	v_4
$P3$	v_2	u_2	u_5	u_6	v_5

$$C_{\max}(S) - C_{opt} \leq t_{\max}(2m-1)/m - 1.$$

Now, we show that this bound is tight.

Example 3.3: Consider the m^2+m+1 jobs and m machine instance. There are $2m$ jobs $v_i : r(v_i) = 0; t(v_i) = m; q(v_i) = 0$. There are $m(m-1)$ jobs $u_i : r(u_i) = m-1$; $t(u_i) = 1$; $q(u_i) = m$ and job $a : r(a) = m-1$; $t(a) = m$; $q(a) = m$.

The makespan of MDT/IIT schedule is $C_{\max}(MDT) = 5m - 2$. The makespan of the Jackson's schedule is $C_{\max}(JR) = 4m - 1$. The optimal makespan is equal $3m$.

Table 1 shows the schedule posted by algorithm MDT/IIT, and Table 2 shows the optimal schedule, for the case $m = 3$. The first row of the table shows the time of the assignments. The next three lines indicate the tasks performed on the processors $P1, P2, P3$, respectively.

We can see that $C_{\max}(MDT) - C_{opt}$ is equal $2m-2$, that is $2t_{max} - 2$. ■

We compare schedules constructed by MDT/IIT algorithm with schedules constructed by nondelay Jackson's algorithm. Consider next example.

Example 3.4: Consider the $m^2 + 1$ jobs and m machine instance.

There are m jobs $v_i : r(v_i) = 0; t(v_i) = m; q(v_i) = 0$. There are $m(m-1)$ jobs $u_i : r(u_i) = 1$; $t(u_i) = 1$; $q(u_i) = m$ and there is job $a : r(a) = 1$; $t(a) = m$; $q(a) = m$.

The makespan of the Jackson's schedule is $C_{\max}(JR) = 4m - 1$. The makespan of MDT/IIT schedule is $C_{\max}(MDT) = 3m$. The makespan of an optimal schedule is $C_{opt} = 2m + 1$.

Table 3 shows the schedule posted by algorithms MDT/IIT, Table 4 shows the Jackson's schedule schedule and Table 5 shows the optimal schedule for the case $m = 3$.

The algorithms JR and MDT are in a certain sense opposites: if the algorithm JR generates a schedule with a large error, the algorithm MDT/IIT works well and vice versa. Examples 3.3 and 3.4 illustrate this property of the algorithms. We propose the combined algorithm that builds two schedules: one by the algorithm JR, the other by the algorithm MDT and selects the best.

TABLE III
MDT SCHEDULE $C_{\max}(MDT) = 3m$

t	1	$m-1$		m	m
$P1$	idle	u_1	u_4	a	v_3
$P2$	idle	u_2	u_5	v_1	idle
$P3$	idle	u_3	u_6	v_2	idle

TABLE IV
THE JACKSON'S SCHEDULE $C_{\max}(JR) = 4m - 1$.

t	m	$m-1$		m
$P1$	v_1	u_1	u_4	a
$P2$	v_2	u_2	u_5	idle
$P3$	v_3	u_3	u_6	idle

TABLE V
OPTIMAL SCHEDULE $C_{\max} = 2m + 1$

t	1	m			m
$P1$	idle	a			v_6
$P2$	idle	u_1	u_4	u_3	v_4
$P3$	idle	u_2	u_5	u_6	v_5

IV. COMPUTATION RESULT

In this section we present the results of testing the proposed algorithm on several types of tests. The quality of the schedules we estimated the average relative gap produced by each algorithm, where the gap is equal to $RT = (C_{\max} - LB)/LB$. We compared algorithms JR, MDT/IIT and the combined algorithm CA, that builds two schedules (one schedule by the algorithm JR, the other by the algorithm MDT) and selects the best solution.

The experiment considered several types of examples. The number of jobs n changed from 100 to 500.

In examples type A job processing time, release and delivery times are generated with discrete uniform distributions between 1 and n. Groups for $m = 20$ and $n = 100, 200, 300, 400, 500$ were tested. For each n we generate 30 instances. 150 instances of type A are tested. The results are given in Table 6. The first column of this table contains the number of jobs n.The columns $N_{opt}(MDT)$, $N_{opt}(JR)$ and $N_{opt}(CA)$ shows the cases (in percents) where optimal schedules were obtained by MDT method, JR method and combined method.

We can see that the problem becomes easier as n increases, because the average number of jobs per processor tends to increase. The average relative gap ranges from 4 % to 21 % for CA algorithm. The combined algorithm allows to improve RT in all cases.

TABLE VI
TYPE A. VARIATION OF n.

n	$RT(MDT)$	$RT(JR)$	$N_{opt}(CA)$	$RT(CA)$
100	0.219	0.228	0	0.216
200	0.147	0.159	0	0.141
300	0.061	0.066	0	0.058
400	0.053	0.051	0	0.052
500	0.047	0.042	0	0.039

In the next experiment we fix the number of jobs $n = 500$ and change the number of processors m from 3 to 170. For each m we generate 30 instances and a total of 240 instances are tested. The results of the experiments are shown in Table 7. The first column of this table contains the number of processors m. Table 7 shows the performance of JR, MDT and CA algorithms.

Table 7 shows that average relative gap increases when m changes from 3 to 100 and reaches a maximum at $m = 100$. Then it decreases and when $m = 170$ algorithm MDT generates 98 % optimal solutions, algorithm JR 96 % and algorithm CA generates optimal solutions for all instances. Algorithms JR and MDT give very close solutions and only with $m = 3, 20, 30, 130, 170$ the algorithm MDT has an advantage. The combined algorithm allows to improve RT in all cases.

We can see from tables 6 and 7 that the most difficult examples occur when the average number of jobs per processor is equal 5.

In the next series of tests, we restricted our instances to those types that found hard. The number of jobs n is equal to 100 and the number of processors m is equal to 20 (5 jobs on average per processor). In instances of type C we change $t_{\max}$. Type C, that were randomly generated as follows: the job processing time is generated with discrete uniform distributions between 1 and $t_{\max}$, where $t_{\ max}$ changes from 20 to 500. For each $t_{\max}$ we generate 30 instances.240 instances of type C are tested. Release and delivery times are generated with discrete uniform distributions on [1,100]. The results of the work are given in Table 8.

We can see that the problem becomes more difficult with increasing $t_{\max}$, the average relative gap increases and remains large at a $t_{\max}$ from 100 to 500. The maximum deviation is reached at $t_{\max} = 200$. The combined algorithm allows to increase (at $t_{\max} = 50$) the number of optimal solutions by 9% and to improve RT in all cases.

In the series of tests considered, the average deviation was slightly different for the algorithms JR and MDT. The combined algorithm allowed us to slightly improve the value of the objective function.

In order to get a better picture of the actual effectiveness of MDT/IIT we consider other types of instances.

In next series we consider instances in which jobs have the same processing time.

Type EJ (Equal job): The heads are drawn from the discrete uniform distribution on [1, 10] and tails from [1, 60], $n = 100$. All processing times $t_i = 60$. We can see the computational results in Table 9, where the last column F contains the difference $RT(JR) - RT(MDT)$.

We can see that for examples of Type EJ the average relative gap is less for algorithm MDT/IIT for all values of m. For $m = 50$, the average relative gap for the JR algorithm is equal 0.40, but for the MDT/IIT algorithm it is only 0.17.

Type SG (small-great) : The heads are generated from the discrete uniform distribution on [1, 10] and tails on [1, 80], $n = 100$. The processing times are drawn from the discrete uniform distribution on $[40, 60]$.

Table 10 shows the results of examples of type SG. For cases $m = 40$ and $m = 50$, there is a significant difference between the results obtained by different algorithms. The average relative gap for MDT algorithm and JR algorithm is equal 0.14 and 0.24, respectively, for $m = 40$. Algorithm MDT/IIT generated 100% of optimal solutions, whereas algorithm JR only 25% for $m = 50$. We observe from Tables 9 and 10 that MDT/IIT exhibits a good performance with instances of types EJ and SL.

Type GS(great-small): The $r(u)$ are drawn from the discrete uniform distribution on [1, 100] and $q(u)$ on [1, 20], $n = 100$. Table 11 shows the results of examples of type LS. The processing times are drawn from the discrete uniform distribution on $[1, n]$.

For examples of the type LS, the greatest deviation is observed at $m = 20$ and $m = 30$. The optimal solutions were obtained only at $m = 50$. The combined algorithm works better than each of the algorithms separately in all types of examples.

V. Conclusion

We propose an approximation IIT algorithm named MDT/IIT (maximum delivery time/ inserted idle time) for $P|r_j, q_j|C_{max}$ problem. We proved that $C_{\max}(S) - C_{opt} < t_{\max}(2m - 1)/m$, and this bound is tight, where $C_{\max}$ is the objective function of MDT/IIT schedule, and C_{opt} is the makespan of an optimal schedule. We observe that MDT/IIT algorithm exhibits a good performance with instances in which delivery times are large compared with processing times and release times.

We propose the combined algorithm that builds two schedules (one by the algorithm JR, the other by the algorithm MDT) and selects the best solution.The algorithms JR and MDT are in a certain sense opposites: if the algorithm JR generates a schedule with a large error, the algorithm MDT works well and vice versa. Computational experiments have shown that the combined algorithm works better than each of the algorithms separately.

References

[1] C. Artigues, D. Feillet, "A branch and bound method for the job-shop problem with sequence-dependent setup times",*Annals of Operations Research*, vol. 159,2008, pp.135—159.

[2] K.R. Baker,*Introduction to Sequencing and Scheduling.* John Wiley & Son, New York, 1974.

[3] P. Brucker, *Scheduling Algorithms. fifth ed.* Springer,Berlin, 2007.

[4] J. Carlier, E. Néron, "An exact algorithm for solving the multiprocessor flowshop," *RAIRO Operations Research*, vol. 34, 2000, pp. 1—25.

[5] J. Carlier, "The one machine sequencing problem." *European Journal of Operational Research*,vol.11,1982, pp. 42–47.

[6] J. Carlier, E. Pinson, " Jackson's pseudo preemptive schedule for the $Pm|rj, qj|Cmax$ scheduling problem," *Annals of Operations Research*, vol. 83, 1998, pp.41–58.

[7] J. Carlier, "Scheduling jobs with release dates and tails on identical machines to minimize the makespan."*European Journal of Operational Research*, vol. 29, 1987,pp.298—306.

TABLE VII
TYPE A. VARIATION OF m.

m	$N_{opt}(MDT)$	$RT(MDT)$	$N_{opt}(JR)$	RT(JR)	$N_{opt}(CA)$	$RT(CA)$
3	0	0.003	0	0.004	0	0.002
10	0	0.019	0	0.016	0	0.014
20	0	0.042	0	0.046	0	0.041
30	0	0.068	0	0.075	0	0.066
50	0	0.146	0	0.135	0	0.132
100	0	0.201	0	0.195	0	0.191
130	0	0.021	0	0.025	0	0.019
170	98	0.001	97	0.001	**100**	**0.000**

TABLE VIII
TYPE C. VARIATION OF $t_{\max}$.

$t_{\max}$	$N_{opt}(MDT)$	$RT(MDT)$	$N_{opt}(JR)$	RT(JR)	$N_{opt}(CA)$	$RT(CA)$
20	100	0.000	99	0.000	100	0.000
50	51	0.004	52	0.005	61	0.003
70	0	0.016	0	0.014	0	0.013
100	0	0.212	0	0.219	0	0.210
200	0	0.223	0	0.221	0	0.220
300	0	0.209	0	0.218	0	0.207
400	0	0.207	0	0.213	0	0.206
500	0	0.203	0	0.206	0	0.202

TABLE IX
TYPE EJ. VARIATION OF m.

m	$RT(MDT)$	$RT(JR)$	$RT(CA)$	F
20	0.03	0.04	0.03	0.01
30	0. 22	0.24	0.22	0.02
40	**0.26**	0.32	0.26	0.06
50	**0.17**	0.40	**0.17**	0.23

TABLE X
TYPE SG. VARIATION OF m

m	$RT(MDT)$	RT(JR)	$RT(CA)$	F
20	0.10	0.11	0.10	0.01
30	0.19	0.23	0.19	0.04
40	**0.14**	0.24	**0.14**	0.10
50	**0.000**	0.23	**0.000**	0.23

TABLE XI
TYPE GS. VARIATION OF m.

m	$RT(MDT)$	$RT(JR)$	$RT(CA)$
3	0.015	0.014	0.014
10	0.100	0.110	0.092
20	0. 239	0.236	0.227
30	0.205	0.198	0.192
50	**0.006**	0.008	**0.000**

[8] C. Chandra, Z.Liu, J. He, J,T. Ruohonen, "A binary branch and bound algorithm to minimize maximum scheduling cost," *Omega*, vol. 42, 2014,,pp.9–15.
[9] A.,Gharbi, M.,Haouari, "Minimizing makespan on parallel machines subject to release dates and delivery times," *Journal of Scheduling*,vol. 5, 2002, pp.329—355.
[10] A. Gharbi, M. Haouari, "Optimal parallel machines scheduling with availability constraints,"*Discrete Applied Mathematics* vol. 148, 2005, pp.63—87.
[11] A. Gharbi,M. Haouari, "An approximate decomposition algorithm for scheduling on parallel machines with heads and tails," *Computers & Operations Research*, vol. 34, 2007, pp.868 —883.
[12] R.L. Graham, E.L. Lawner, A.H.G. Rinnoy Kan, "Optimization and approximation in deterministic sequencing and scheduling," A survey.*Ann. of Disc. Math.*, vol. 5 (10),1979, pp. 287–326.
[13] N.S.Grigoreva, "Branch and bound method for scheduling precedence constrained tasks on parallel identical processors", Lecture Notes in Engineering and Computer Science. *In proc. of The World Congress on Engineering 2014, WCE 2014 London, U.K.* 2014, pp.832–836.
[14] N.Grigoreva, "Single Machine Inserted Idle Time Scheduling with Release times and Due Dates," *Proc.DOOR2016. Vladivostoc,Russia. Sep.19-23.2016. CEUR-WS.*2016, vol.1623, pp. 336—343.
[15] D.Gusfield, "Bounds for naive multiple machine scheduling with release times and deadlines,"*Journal of Algorithms* vol.5,1984, pp.1—6.
[16] L.A.Hall, D.B. Shmoys, "Jackson's rule for single-machine scheduling: making a good heuristic better," *Mathematics of Operations Research.* vol.17 (1),1992, pp.22–35.
[17] L.A.Hall,D.B. Shmoys, "Approximation schemes for constrained scheduling problems", *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science,* 1989, pp. 134 —139.
[18] M. Haouari, A. Gharbi, "Lower bounds for scheduling on identical parallel machines with heads and tails,"*Annals of Operations Research* vol. 129, 2004, pp.187—204.
[19] J. Kanet, V. Sridharan, "Scheduling with inserted idle time:problem taxonomy and literature review," *Oper.Res.* vol.48 (1),2000, pp. 99–110.
[20] J.A Lenstra, A.H.G. Rinnooy Kan, P. Brucker, "Complexity of machine scheduling problems,"*Ann. of Disc. Math.*, vol. 1,1977, pp.343–362.
[21] M. Mastrolilli, "Efficient approximation schemes for scheduling prob-

lems with release dates and delivery times,"*Journal of Scheduling*, vol. 6, 2003, pp.521—531.

[22] E. Nowicki, C. Smutnicki, "An approximation algorithm for a single-machine scheduling problem with release times and delivery times," *Discrete Applied Mathematics* ,vol. 48, 1994, pp.69–79.

[23] J. Omer, A. Mucherino, "Referenced Vertex Ordering Problem",*Theory, Applications and Solution Methods HAL open archives, hal-02509522, version 1,* March, 2020.

[24] K. Sourirajan, R. Uzsoy, "Hybrid decomposition heuristics for solving large-scale scheduling problems in semiconductor wafer fabrication," *Journal of Scheduling,* vol. 10, 2007, pp.41–65.

[25] Y. Pan, L. Shi, "Branch and bound algorithm for solving hard instances of the one-machine sequencing problem,"*European Journal of Operational Research,* 168, 2006, pp. 1030—1039.

[26] C.N. Potts, "Analysis of a heuristic for one machine sequencing with release dates and delivery times," *Operational Research.* vol. 28 (6), 1980, pp. 445–462.

[27] J. Ullman, "NP-complete scheduling problems," *J. Comp. Sys. Sci.* vol. 171, 1975,pp. 394—394.

[28] Y. Zinder, D. Roper, "An iterative algorithm for scheduling unit-time operations with precedence constraints to minimize the maximum lateness,"*Annals of Operations Research,* 81, 1998, pp.321–340.

[29] Y. Zinder, " An iterative algorithm for scheduling UET tasks with due dates and release times," *European Journal of Operational Research,* vol.149, 2003, pp.404–416.