IEICE TRANS. FUNDAMENTALS, VOL.E88–A, NO.5 MAY 2005

PAPER Special Section on Discrete Mathematics and Its Applications

A Heuristic Algorithm for One-Machine Just-In-Time Scheduling Problem with Periodic Time Slots

Eishi CHIBA[†], *Student Member and* **Kunihiko HIRAISHI**[†a)], *Member*

SUMMARY Just-in-time scheduling problem is the problem of finding an optimal schedule such that each job finishes exactly at its due date. We study the problem under a realistic assumption called periodic time slots. In this paper, we prove that this problem cannot be approximated, assuming $P \neq NP$. Next, we present a heuristic algorithm, assuming that the number of machines is one. The key idea is a reduction of the problem to a network flow problem. The heuristic algorithm is fast because its main part consists of computation of the minimum cost flow that dominates the total time. Our algorithm is $O(n^3)$ in the worst case, where n is the number of jobs. Next, we show some simulation results. Finally, we show cases in which our algorithm returns an optimal schedule and is a factor 1.5 approximation algorithm, respectively, and also give an approximation ratio depending on the upper bound of set-up times.

key words: *scheduling, just-in-time, set-up times, heuristic algorithm, minimum cost flow*

1. Introduction

For many years, research on scheduling has focused on single performance measures, referred to as *regular* measures that are nondecreasing in job completion times. Most of the literature deals with such regular measures as mean flow-time, mean latency, percentage of tardy jobs, and mean tardiness (see [1] for the definition of terms). With the growing interest in Just-In-Time (JIT) production, the demand for research into problems with *irregular* performance measures has considerably increased (see [2]). In JIT production, an ideal schedule is one in which all jobs finish exactly on their assigned due dates. There are only a few good studies that deal with such ideal schedule; see Cormen et al. [3] for such a classical and famous scheduling problem, sometimes called activity-selection problem.

Recently, a quadratic time algorithm which solves generalization of activity-selection problem was presented in [4]. A polynomial time algorithm which solves a more general class than above was presented in [5]. Čepek and Sung [4] suggests that the main contribution of Hiraishi et al. [5] lies in the results for the identical parallel machines with due dates. It was shown in [5], that the problem of activity selection is solvable in polynomial time even if a nonnegative weight is assigned to each job, a nonnegative set-up time is assigned to each ordered pair of jobs, and the objective is to maximize the weighted number of just-in-time jobs, where a job is called *just-in-time* if it is completed exactly on its due date. Considering real-life situations, jobs which are not processed in some period will be scheduled on the next chance, e.g. tomorrow, next week, next month, etc. We can formalize such situations as periodic time slots.

The reason that all scheduled jobs must be just-in-time in our problem simply comes from the fact that inventory costs must be reduced in production. In manufacturing, if the time instance of manufacturing a product is earlier than the scheduled shipping time, inventory costs increases proportionally. The concept of periodic time slots comes from the real situation of production. For example, we can have an assumption that the shipping time is fixed on a span (daily, weekly, monthly, etc.) basis. Our problem was first studied by Hiraishi [6] in the environment of identical parallel machine. There are many practical problems motivating our study, e.g. automotive manufacturing, observation from satellites etc.

In this paper, we study the problem of Just-In-Time scheduling with periodic time slots. A processing time and periodically repeating due dates are assigned to each job, where the period is same for all jobs. A set-up time is assigned to each ordered pair of jobs. The objective is to minimize the maximum number of periodic time slots required for each machine that are sufficient for scheduling all given jobs, in which each job is completed exactly at one of its due dates. We first formulate our problem as an optimization problem. Next, we prove the inapproximability result for the problem assuming $P \neq NP$. Next, we present a heuristic algorithm using network flows assuming a single machine. Then, we show some simulation results. Finally, we show cases for which our algorithm returns an optimal schedule and is a factor 1.5 approximation algorithm, respectively, and also give an approximation ratio depending on the upper bound of set-up times.

2. Notation and Problem Formulation

The scheduling problem considered here has multiple time slots and in each time slot, due dates of a job j are $d_j, L + d_j, 2L + d_j, \ldots$, where L is the length of a time slot. Each job is processed by parallel identical machines, and should be finished exactly at its due date in one of time slots. Then the problem is to find a nonpreemptive schedule that minimizes the maximum number of time slots required for each machine when all jobs are processed.

Manuscript received August 16, 2004.
Manuscript revised November 19, 2004.
Final manuscript received January 5, 2005.

[†]The authors are with the School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa-ken, 923-1292 Japan.
a) E-mail: hira@jaist.ac.jp

DOI: 10.1093/ietfec/e88–a.5.1192

We describe a formal definition of the problem. The following notations will be used:

- $\mathbb{N}$ is the set of natural numbers.
- $M_1, M_2, \ldots, M_m$: m parallel identical machines.
- $J = \{1, 2, \ldots, n\}$: the set of n jobs to be processed.
- p_j (> 0): processing times of job j on a machine.
- d_j ($\geq p_j$): due dates of job j.
- s_{jk} (≥ 0): set-up times for each ordered pair of jobs j and k, i.e., the time difference between the completion of j and the start of k must be at least s_{jk} if they are scheduled consecutively on the same machine.
- L ($\geq \max_{j\in J}\{d_j\}$): the length of a time slot.

The assumption of $d_j \geq p_j$ is practical because if a job is scheduled just-in-time, it completely fits within a single time slot.

A *schedule* is a mapping $S : j \mapsto (M^S_{[j]}, C^S_j)$, where $M^S_{[j]}$ is the machine on which job j is processed and C^S_j is the time instant when job j is finished on machine $M^S_{[j]}$. A schedule S is called *feasible* if

(i) for each job j, there exists a nonnegative integer r^S_j such that $C^S_j = r^S_j \cdot L + d_j$, and
(ii) for every ordered pair of jobs j and k, if $M^S_{[j]} = M^S_{[k]}$, then $C^S_j + s_{jk} + p_k \leq C^S_k$ or $C^S_k + s_{kj} + p_j \leq C^S_j$.

Let $r(S) := \max_{j\in J}\{r^S_j\}$. Then our problem, i.e. *Just-In-Time Scheduling Problem with Periodic Time Slots* (JIT-SP) is stated as follows: Given processing time p_j, due dates d_j, set-up times s_{jk}, and the length of a time slot L; minimize the value $r(S)$ called the *objective function* over the set of all feasible nonpreemptive schedules. Intuitively, the problem is to schedule n jobs so that the maximum number of time slots required for each machine is minimized. When the number of machines is equal to one, i.e., $m = 1$, the problem is to schedule n jobs so that the number of time slots required for the machine is minimized.

An *optimal schedule* for an instance of the problem is a feasible schedule that achieves the smallest objective function value, which is called the *optimal cost*. The optimal cost plus one means the number of time slots in the optimal schedule. If the number of machines is no less than the number of jobs, then there is a feasible schedule in which no machines executes more than one job. Consequently, the problem is trivial. From now on, we assume that $m < n$.

Now, for every ordered pair of jobs j and k, let g_{jk} be a nonnegative integer such that

$$(g_{jk} - 1) \cdot L + d_k < d_j + s_{jk} + p_k \leq g_{jk} \cdot L + d_k. \qquad (1)$$

This inequality means that a machine can start processing job k in g_{jk} time slots after job j is finished on the same machine such that both jobs are finished exactly at their due dates. Given an instance of the problem, such g_{jk} uniquely exists for any ordered pair of jobs j and k. Let $g := \max_{j,k\in J, j\neq k}\{g_{jk}\}$.

Known results are [7]:

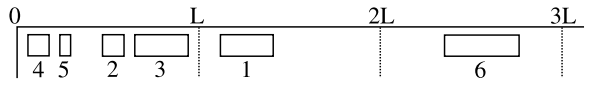


Fig. 1 A schedule by the greedy method.

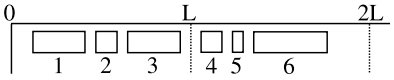


Fig. 2 An optimal schedule.

- JIT-SP becomes *NP*-hard (in the strong sense) even for the single machine case (i.e. for $m = 1$).
- If set-up times are not considered (i.e. $s_{jk} = 0$), JIT-SP is solvable in polynomial time for an arbitrary number of identical parallel machines.

Example 1: We consider the following instance of the one-machine six-job problem, i.e. $m = 1$, $n = 6$: $s_{jk} = 1$ for every ordered pair of jobs j and k, $L = 17$, and the following processing times and due dates.

j	p_j	d_j
1	5	7
2	2	10
3	5	16
4	2	3
5	1	5
6	7	13

Using the algorithm in [5], we can obtain a feasible solution in a greedy way, i.e., we schedule maximum possible number of jobs from the first time slot sequentially. Then, three time slots are required for the machine; see Fig. 1. However, two time slots are sufficient for the optimal schedule; see Fig. 2.

3. Inapproximability Result for JIT-SP

In this section we prove that JIT-SP cannot be approximated, assuming *P≠NP*.

Theorem 1: For any polynomial time computable function $\alpha(n)$, JIT-SP cannot be approximated within a factor of $\alpha(n)$ even for the single machine case ($m = 1$), unless *P=NP*.

Proof: Assume, for a contradiction, that there is a factor $\alpha(n)$ polynomial time approximation algorithm, $\mathcal{A}$, for the general JIT-SP. We will show that $\mathcal{A}$ can be used for deciding the well-known strongly *NP*-complete Hamiltonian path problem (see e.g. [8]) in polynomial time, thus implying *P=NP*. The reduction is given below.

Hamiltonian path (HP)
Instance: An undirected graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ for some $n \in \mathbb{N} - \{0\}$.
Question: Is there a Hamiltonian path in G, i.e. a permutation of vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_n}$ such that $(v_{i_k}, v_{i_{k+1}}) \in E$ for every $1 \leq k \leq n - 1$?

Now we shall construct an instance of our JIT-SP as follows.

JIT-SP

Instance: $J = \{1, 2, \ldots, n\}$ (we identify jobs with vertices of the input graph), $m = 1$, $L = 1$, $p_j = d_j = 1$ for all $j \in J$, and

$$s_{jk} = \begin{cases} 0 & \text{if } (v_j, v_k) \in E, \\ \alpha(n) \cdot n & \text{otherwise.} \end{cases}$$

This reduction transforms an input of HP to an input of JIT-SP such that

- if G has a Hamiltonian path, then the number of time slots in an optimal schedule of the input of JIT-SP is n, and
- if G does not have a Hamiltonian path, then an optimal schedule of the input of JIT-SP is of number of time slots $> \alpha(n) \cdot n$.

Observe that, on the above input of JIT-SP, algorithm $\mathcal{A}$ must return a solution of number of time slots $\leq \alpha(n) \cdot n$, exactly n in fact, in the first case, and a solution of number of time slots $> \alpha(n) \cdot n$ in the second case. Thus, it can be used for deciding whether G contains a Hamiltonian path. □

Notice that, to obtain such a strong nonapproximability result, we had to assign set-up time that violates $s_{jk} \leq h \cdot L$ $(j, k \in J,\ j \neq k,\ h \in \mathbb{N} \setminus \{0\})$, where h is a constant. If we restrict ourselves to inputs of JIT-SP in which the number of machines, $m = 1$ and satisfies $s_{jk} \leq h \cdot L$ $(j, k \in J,\ j \neq k,\ h \in \mathbb{N} \setminus \{0\})$, the problem remains *NP*-hard (in the strong sense), but is no longer hard to approximate. We describe our solution to that in Sect. 6.

4. A Heuristic Algorithm

In this section, we present a heuristic algorithm for one-machine JIT-SP using network flows. This algorithm also gives a lower bound on the number of time slots for general JIT-SP.

From this section, the following well-used notations are used. Let $G = (V, A)$ be a digraph with vertex set V and arc set A. For each arc $e \in A$, let $lcap(e)$ and $ucap(e)$ be lower and upper bounds for the flow across e and let $w(e)$ be the weight of shipping one unit of flow across e, and for each node $v \in V$ let $supply(v)$ be the supply or demand at node v. We talk about a supply if $supply(v) > 0$ and we talk about a demand if $supply(v) < 0$. We assume that the supplies and demands balance, i.e., $\sum_{v \in V} supply(v) = 0$. A flow f is a function on the arcs satisfying the capacity constraints and the mass balance conditions, i.e., $lcap(e) \leq f(e) \leq ucap(e)$ for every arc $e \in A$ and $supply(v) = \sum_{e; source(e)=v} f(e) - \sum_{e; target(e)=v} f(e)$ for every node $v \in V$.

For every arc $e \in A$, $w(e)$ is the weight of sending one unit of flow across the arc. The total weight of a flow f is therefore given by $w(f) = \sum_{e \in A} f(e) \cdot w(e)$.

Now, given an instance of general JIT-SP, we construct a simple connected digraph $G = (V, A)$ as follows:

- The set V consists of $2n+2$ nodes, i.e., $s, a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n, t$, where s is the source and t is the sink; and each pair of vertices a_j, b_j represents job j and are called *transshipment vertices*.
- The set A consists of the following $n^2 + 2n$ arcs:
  - (s, a_j) with $w(s, a_j) = 0$ for all $j \in J$;
  - (a_j, b_j) with $w(a_j, b_j) = -w_{\text{job}}$ for all $j \in J$;
  - (b_j, t) with $w(b_j, t) = 0$ for all $j \in J$;
  - (b_j, a_k) with $w(b_j, a_k) = g_{jk}$ for every ordered pair of jobs j and k
- $lcap(e) = 0$ and $ucap(e) = 1$ for all $e \in A$.

where w_{job} is any positive integer such that $g < w_{\text{job}}$. This inequality means that the absolute value of the weight of each arc (a_j, b_j) which represents a job j is greater than the weight of other arcs.

Example 2: We consider the following instance of the one-machine four-job problem, i.e., $m = 1, n = 4$: $s_{jk} = 1$ for every ordered pair of jobs j and k, $L = 8$, and the following processing times and due dates.

j	p_j	d_j
1	2	2
2	2	6
3	3	4
4	2	8

Given above instance, we can compute g_{jk} by Eq. (1) as follows: $g_{12} = 0$, $g_{13} = 1$, $g_{14} = 0$, $g_{21} = 1$, $g_{23} = 1$, $g_{24} = 1$, $g_{31} = 1$, $g_{32} = 1$, $g_{34} = 0$, $g_{41} = 2$, $g_{42} = 1$, $g_{43} = 1$. Then, we construct G consisting of 10 nodes and 24 arcs, as shown in Fig. 3.

Next, for above-constructed G, we solve the minimum cost flow problem that can be stated as follows:

Minimize $w(f)$ subject to

$$supply(v) = \begin{cases} m & (v = s), \\ 0 & (v \in V \setminus \{s, t\}), \\ -m & (v = t), \end{cases}$$

$0 \leq f(e) \leq 1$ for all $e \in A$.

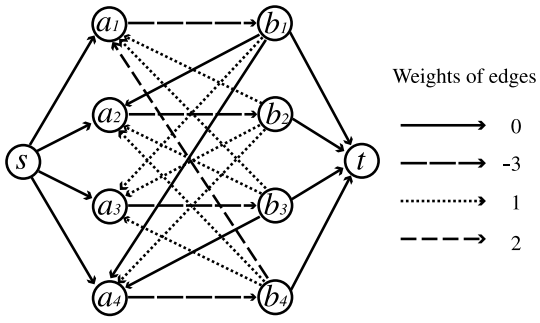


Fig. 3 A constructed graph G from Example 2.

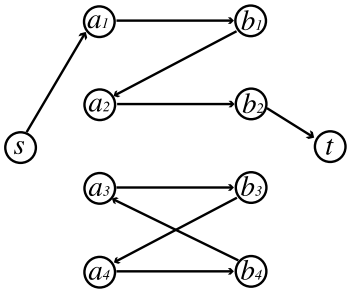


Fig. 4 An example of G' obtained from Example 2.

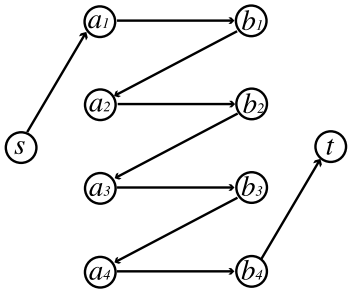


Fig. 5 An example of G' obtained from Example 2.

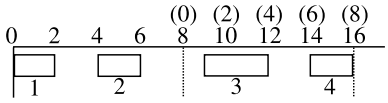


Fig. 6 An optimal schedule for Example 2.

Now, it is easy to find a feasible flow in G because $m < n$. For example, the flow defined by

$$f(u,v) = \begin{cases} 1 & \text{if } u = s \text{ and } v = a_j \ (1 \le j \le m), \\ 1 & \text{if } u = a_j \text{ and } v = b_j \ (1 \le j \le n), \\ 1 & \text{if } u = b_j \ (1 \le j \le m-1,\ j = n) \text{ and } v = t, \\ 1 & \text{if } u = b_j \text{ and } v = a_{j+1} \ (m \le j \le n-1), \\ 0 & \text{otherwise} \end{cases}$$

is one of them.

A minimum cost flow maps each arc to a zero or one. Each arc (a_j, b_j), which is a pair of transshipment vertices, are always mapped to one. It is because we define the function w on each arc (a_j, b_j) by $-w_{\text{job}}$ such that $g < w_{\text{job}}$.

We here introduce G' in order to represent a flow in G. The vertex set of G' is equal to the vertex set of G. The arc set of G' consists of arcs such that $f(e) = 1$ for all $e \in A$. Therefore, when G' is depicted in a figure, we omit arcs such that $f(e) = 0$, and depict arcs such that $f(e) = 1$ for all $e \in A$. Then, there exist m paths from s to t in G' obtained by computing a minimum cost flow for G. Note that, if $w(e)$ for all $e \in A$ are positive, there exist no paths outside of m paths from s to t in G'. But, there may exist some directed cycles apart from the m paths in G' because each $w(a_j, b_j)$ is negative. Such example is shown in Fig. 4. This is the G' obtained by computing a minimum cost flow for G shown in Fig. 3. There is only one path from s to t because of $m = 1$ and one directed cycle, i.e., $a_3 \to b_3 \to a_4 \to b_4 \to a_3$, in this G'.

If G' has no directed cycles, G' corresponds to a feasible schedule as follows. We can assume that there are exactly m paths from s to t and no arcs apart from the m paths in G'. We can describe m paths as m flows. More precisely, the flow that G' represents can be decomposed into exactly m flows $f_1, f_2, \ldots, f_m$ such that $f = \sum_{i=1}^{m} f_i$. The arcs at which f_i has value one determine a path from s to t. We can associate each flow f_i with a machine and schedule the jobs represented by arcs (a_j, b_j) belonging to the same path from s to t on the associated machine. There are $m!$ such mapping from $\{f_1, f_2, \ldots, f_m\}$ to $\{M_1, M_2, \ldots, M_m\}$ but we are free to select arbitrarily one of them because the machines are identical. If a path from s to t in G' goes through an arc with a positive weight, then the jobs following this arc are processed in the next or later time slots. More precisely, each path π from s to t in G' is decomposed into paths $\pi_1, \pi_2, \ldots, \pi_l$ by removing every arc with a positive weight. We assign each path π_i to a time slot as follows.

(i) Assign path π_1 to the first time slot.
(ii) Suppose that path π_i is assigned to the k-th time slot. If the arc between π_i and π_{i+1} has positive weight k', then assign path π_{i+1} to the $(k + k')$-th time slot.

Therefore, we can associate G' with a feasible schedule if G' has no directed cycles.

Example 3: We consider the same instance as Example 2. We compute a minimum cost flow for G shown in Fig. 3. Note that, the flow is underspecified. There generally exist some flows that have the same total weight. Such example is shown in Fig. 5. The arc from b_2 to a_3 has weight 1. Both arcs from b_1 to a_2 and b_3 to a_4 have weight 0. This is also obtained by computing a minimum cost flow for G shown in Fig. 3. This G' has no directed cycles. Therefore, this corresponds to a feasible schedule. The obtained schedule is shown in Fig. 6. In this case, the schedule is optimal.

When G' has no directed cycles, the *total number of time slots* in the feasible schedule to which G' corresponds is as follows.

$$\sum_{e \in A, w(e) > 0} (f(e) \cdot w(e)) + m.$$

Remark 1: When G' has no directed cycles, the feasible schedule to which G' corresponds minimizes the total number of time slots. But, we want to find the schedule which minimizes the maximum number of time slots required for each machine. If the number of machines is one, i.e., $m = 1$, both are equal. Thus, when $m = 1$ and G' has no directed cycles, a schedule which G' corresponds to, is optimal. Therefore, the obtained schedule is optimal in the case of Example 3.

Remark 2: Generally, G' may have some directed cycles in addition to some paths from s to t. Therefore, a lower bound on the total number of time slots is

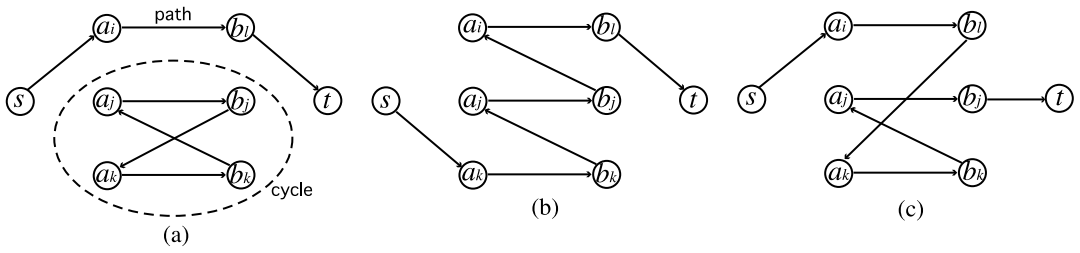


Fig. 7 The updation of G' at Step 4.

$\sum_{e\in A,w(e)>0}(f(e)\cdot w(e))+m$. Thus, the following expression gives a lower bound on the number of time slots for general JIT-SP.

$$\left\lceil \frac{1}{m}\cdot \sum_{e\in A,w(e)>0}(f(e)\cdot w(e)) \right\rceil + 1.$$

We assume $m = 1$ in what follows. When G' has some directed cycles, G' does not directly correspond to a feasible schedule by the above-mentioned way. In this paper, we obtain a feasible schedule as follows even if G' has some directed cycles.

Heuristic Algorithm (the case of one machine):

Step 1. Construct G.

Step 2. Compute a minimum cost flow for G. (If G' has no directed cycles, go to Step 5; otherwise go to Step 3.)

Step 3. Find an arc (b_j, a_k) on a directed cycle with $\min_{j\neq k,j,k\in J}\{w(b_j,a_i) - w(b_j,a_k), w(b_l,a_k) - w(b_j,a_k)\}$, where a_i is the first node and b_l is the last node on the path from s to t in G'.

Step 4. For the arc (b_j, a_k) obtained at Step 3,
IF $w(b_j,a_i) - w(b_j,a_k) < w(b_l,a_k) - w(b_j,a_k)$
$f(b_j,a_k) = 0, f(s,a_i) = 0,$
$f(b_j,a_i) = 1, f(s,a_k) = 1.$
ELSE
$f(b_j,a_k) = 0, f(b_l,t) = 0,$
$f(b_l,a_k) = 1, f(b_j,t) = 1.$
(If G' that represents updated flow has no directed cycles, go to Step 5; otherwise go to Step 3.)

Step 5. Obtain a feasible schedule from G'.

At Step 2, G' has exactly one path from s to t because of $m = 1$. If G' has no directed cycles, we obtain an optimal schedule. Otherwise, we do not necessarily obtain an optimal schedule. At Step 3, we find an arc which is deleted in G'. The number of operations in Step 3 is proportional to the number of arcs in the directed cycles. Step 4 is to update current flow. The updation of the flow is shown in Fig. 7. Figure 7(a) corresponds to original G'. If the conditional expression of if statement at Step 4 is true, the G' which represents updated flow corresponds to Fig. 7(b). Otherwise, the G' which represents updated flow corresponds to Fig. 7(c). The role of Step 4 is to reduce the number of directed cycles in G' by exactly one. Therefore, the total number of Step 4 operations over the heauristic algorithm is equal to the number of directed cycles in G' at Step 2. At Step 2, G' has at most $\lfloor (n-1)/2 \rfloor$ directed cycles. When each directed cycle contains exactly two (a_j, b_j)-type arcs, G' has $\lfloor (n-1)/2 \rfloor$ directed cycles. We can assign G' obtained at Step 5 to a feasible schedule because G' always has no directed cycles. At that time, the number of time slots in the feasible schedule is as follows:

$$\sum_{e\in A,w(e)>0}(f(e)\cdot w(e)) + 1. \tag{2}$$

In fact, the arc reversal transformation (cf. [9]) is used to remove arcs with negative weights before computing a minimum cost flow at Step 2. Then, the flow value is set to $n + 1$ because 1 is the original flow value and n is the increase in the flow value by the arc reversal transformation. A minimum cost flow is computed in time $O(n^3)$ using the successive shortest path algorithm as presented by [9]. Computation of a minimum cost flow obviously dominates the total computation time for our heuristic algorithm. Therefore, our algorithm returns a feasible schedule in time $O(n^3)$.

Remark 3: We presented a method based on network flow in this section, but it can also be solved based on Traveling Salesman Problem (TSP) and cycle cover. Concretely speaking, we reduce one-machine JIT-SP to TSP by constructing G, and then use a solution of the minimum cost cycle cover problem (i.e., the problem of finding a min cost set of directed cycles that cover all the vertices in a given directed graph). The minimum cost cycle cover problem is a relaxation of TSP, and it is solvable in time $O(n^3)$ by a well-known reduction to the assignment problem [10]. If an optimal solution for this problem consists of a single cycle, it must be an optimal TSP tour as well; however, an optimal solution consists of multiple cycles in general. We connect such cycles.

5. Computational Experiment

We have implemented our algorithm in C++ on a Dell Precision 650† PC. For our experiment, the number of machines, $m = 1$ and length of a time slot, $L = 20$. We generate our problem instances randomly as follows:

- $0 <$ due dates $d_j \leq L$ $(j \in J)$

†OS: Microsoft Windows 2003 Server, CPU: Intel Xeon 3.06 GHz × 2, RAM: 4 GByte.

Table 1 The rate of obtaining G' which has no directed cycles at Step 2.

n	5	10	20	40	80	160	320
Rate(%)	35	21	5	4	5	2	0

Table 2 Number of directed cycles in G'.

n	Maximum	Average	Variance	#
100	8	3.56	2.4264	49
200	10	4.08	3.1136	99
400	12	4.57	4.8051	199
800	12	5.14	4.3804	399
1600	11	5.8	4.46	799

- $0 <$ processing times $p_j \leq d_j$ $(j \in J)$
- $0 \leq$ set-up times $s_{jk} \leq L$ $(j, k \in J, j \neq k)$

The inequality constraints on due dates and processing times described above come from our problem definition. The inequality constraint on set-up times described above is different from our problem definition. But, here we have L as an upper bound on set-up times in order to simulate our algorithm. Both the algorithms based on the minimum cost flow and pseudorandom generator are implemented using the library functions in LEDA [11].

Now, if G' has no directed cycles at Step 2, we directly go to Step 5, i.e., an obtained schedule is always optimal. We checked how many such cases appear by repeating 100 trials. The experimental result is shown in Table 1. The first row and second row denote the number of jobs n and the rate of obtaining G' that has no directed cycles at Step 2, respectively. It turns out from Table 1 that the rate is relatively high when $n = 5$, but G' has some directed cycles in 100% when $n = 320$. We predictably confirmed the result that the overall rate tends to decrease with increasing number of jobs.

Next, we checked how many directed cycles G' has at Step 2 by repeating 100 trials. The experimental result is shown in Table 2. Each column denotes number of jobs n, maximum number of directed cycles, average number of directed cycles, their variance, and maximum number of directed cycles # in the theoretical sense, i.e., $\lfloor (n-1)/2 \rfloor$, respectively. The maximum number of directed cycles is considerably less than the theoretical maximum for each job. The average is also low, around 3–6 cycles throughout the whole experiment. Therefore, G' actually does not tend to have much directed cycles at Step 2. Additionally, both the maximum and average do not increase so much with increasing number of jobs.

Next, we checked approximation ratio on the number of time slots by repeating 10,000 trials. The approximation ratio is the ratio between the number of time slots computed by our algorithm (see Eq. (2)), and the minimum number of time slots obtained in an optimal schedule by checking all feasible schedules. The experimental result is shown in Table 3. Each column denotes number of jobs n, maximum, average, and variance, respectively of the approximation ratios. The maximum is at most 1.5, but the average is very low, i.e., almost one, throughout the whole experiment. The variance is nearly equal to zero. These results mean that our algorithm has pretty good performance on the average for approximation ratio.

Table 3 Approximation ratio.

n	Maximum	Average	Variance
3	1	1	0
4	1.5	1.00978	0.00300402
5	1.33333	1.01522	0.00366418
6	1.5	1.01932	0.00393428
7	1.4	1.0224	0.00393347
8	1.4	1.02461	0.00370951
9	1.4	1.0237	0.00326279
10	1.4	1.02431	0.00301326

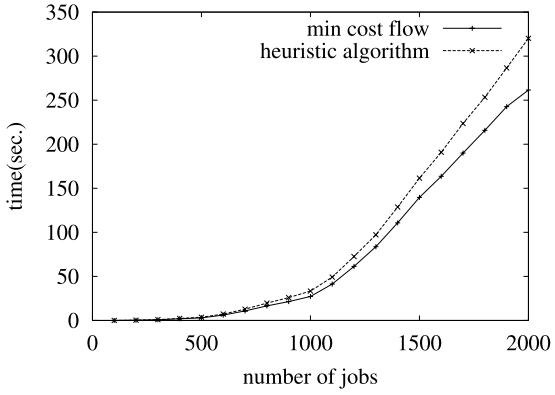


Fig. 8 CPU time of our algorithm.

Finally, we checked the CPU time used by our algorithm by repeating 100 trials. The experimental result is shown in Fig. 8. In the Fig. 8, a point represents the average of CPU time for each $n = 100, 200, \ldots, 2000$. The solid line represents the time of computing a minimum cost flow. The broken line represents the time of including updations of the flow over and above computation of the minimum cost flow. Our algorithm has a great advantage in terms of CPU time because the number of updations of the flow is very low and computation of the minimum cost flow dominates the total time of our algorithm. The experiments whose results were discussed above show that it is much faster in practice.

6. Approximation Ratio

In this section, we show some results on approximation ratio under a constraint.

Lemma 1: If number of jobs is two, i.e., $n = 2$, our heuristic algorithm returns an optimal schedule.

Proof: If $n = 2$, Step 2 can compute one or the other of two different minimum cost flows. G's which represents such flows are shown in Figs. 9(a) and (b). Both have no directed cycles. Therefore, the obtained schedule is optimal. □

Next, we introduce the following variables for G constructed at Step 1.

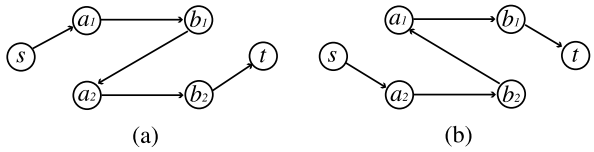


Fig. 9 G' obtained at Step 2 when $n = 2$.

$$\delta_{j*} := \max_{j\neq k,k\in J}\{w(b_j,a_k)\} - \min_{j\neq k,k\in J}\{w(b_j,a_k)\},$$
$$\delta_{*k} := \max_{j\neq k,j\in J}\{w(b_j,a_k)\} - \min_{j\neq k,j\in J}\{w(b_j,a_k)\},$$
$$\delta := \max_{j\neq k,j,k\in J}\min\{\delta_{j*},\delta_{*k}\}.$$

δ denotes an upper bound on the weight increased by updating the flow at Step 4. Additionally, the following notations are used:

- γ: number of time slots in any schedule by our algorithm.
- γ^*: number of time slots in an optimal schedule.
- γ_f^*: number of time slots obtained by a minimum cost flow $f_{\min}$ at Step 2, which is defined by

$$\sum_{e\in A, w(e)>0}(f_{\min}(e)\cdot w(e)) + 1.$$

Note that γ_f^* is a lower bound of γ^*. Therefore, $\gamma_f^* \leq \gamma^*$.

Lemma 2: Our algorithm gives a schedule with at most $\gamma^* + \delta \cdot \lfloor (n-1)/2 \rfloor$ time slots.

Proof: At Step 2, G' has at most $\lfloor (n-1)/2 \rfloor$ directed cycles. Therefore, the number of updations of the flow is also at most $\lfloor (n-1)/2 \rfloor$. Because the increase in weight is at most δ every time we update the flow, the total number of time slots is finally at most $\gamma_f^* + \delta \cdot \lfloor (n-1)/2 \rfloor$. Since $\gamma_f^* \leq \gamma^*$, this lemma is proved. □

Theorem 2: Our algorithm is a factor 1.5 approximation algorithm for the assumption that G constructed at Step 1 has arcs (b_j, a_k) with $w(b_j, a_k) \in \{1, 2\}$ for every ordered pair of jobs j and k.

Proof: Since $w(b_j, a_k) \in \{1, 2\}$, n is a lower bound of γ^*, i.e., $n \leq \gamma^*$. δ is at most one. Then, by Lemma 2

$$\gamma \leq \gamma^* + \delta \cdot \left\lfloor \frac{n-1}{2} \right\rfloor \leq \gamma^* + \frac{n-1}{2}.$$

Thus,

$$\frac{\gamma}{\gamma^*} \leq 1 + \frac{n-1}{2\gamma^*} \leq 1 + \frac{n-1}{2n} = 1.5 - \frac{1}{2n} \xrightarrow[(n\to\infty)]{} 1.5.$$

□

It's easy to find a tight example, and is omitted.

Next, we introduce the following variables in order to derive an approximation ratio for general case:

$$w_j^* = \min_{k\in J}\left\{w(b_j,a_k)\right\}, w^* = \max_{j\in J}\{w_j^*\}, c = \frac{(\sum_{j=1}^n w_j^*) - w^*}{n-1}.$$

Then, $c(n-1)+1$ is a lower bound of γ_f^*, and we have by Lemma 2

$$\gamma \leq \gamma^* + (\Delta - 1)\cdot\left\lfloor \frac{n-1}{2} \right\rfloor, \tag{3}$$

thus,

$$\frac{\gamma}{\gamma^*} \leq 1 + \frac{(n-1)(\Delta-1)/2}{\gamma^*} \leq 1 + \frac{(n-1)(\Delta-1)/2}{c(n-1)+1} \leq 1 + \frac{(n-1)(\Delta-1)/2}{c(n-1)} = 1 + \frac{\Delta-1}{2c}, \tag{4}$$

where $\Delta := \max_{j\neq k,j,k\in J}\{w(b_j,a_k)\}$. At Step 4, the increase in weight by changing the flow is at most $\Delta - 1$, because:

- at Step 4, the increase in weight by selecting an arc with cost 0 is at most Δ.
- every cycle has an arc with positive weight and the increase in weight by selecting this arc at Step 4 is less than Δ.
- Since an arc (b_k, a_j) with minimum $\{w(b_k,a_i) - w(b_k,a_j), w(b_l,a_j) - w(b_k,a_j)\}$ is selected at Step 3, the weight increased by changing the flow at Step 4 is at most $\Delta - 1$.

We have the following result, which is directly derived by Eq. (4).

Corollary 1: If $w(b_j, a_k) \in \{0, 1\}$ for every ordered pair of jobs j and k, then our algorithm gives an optimal schedule.

Finally, we derive an approximatin ratio under a constraint on the set-up time.

Lemma 3: At Step 2, G' has at most $\gamma^* - 1$ directed cycles.

Proof: We proceed to prove this lemma by contradiction as follows. Suppose, to the contrary that at Step 2 the number of directed cycles in G' is greater than or equal to γ^*. Every cycle has at least one arc with positive weight. Therefore, a cycle contributes at least one time slot. Thus, the total number of time slots in an optimal schedule is at least $\gamma^* + 1$. This is a contradiction. □

Theorem 3: Our algorithm is a factor $h + 1$ approximation algorithm under the constraint that set-up time $s_{jk} \leq h \cdot L$ $(j, k \in J, j \neq k, h \in \mathbb{N} \setminus \{0\})$.

Proof: By Eq. (3) and Lemma 3, we have

$$\gamma \leq \gamma^* + (\Delta - 1)\cdot(\gamma^* - 1).$$

Since $0 \leq s_{jk} \leq h \cdot L$, we have $\Delta \leq h + 1$. Thus,

$$\gamma \leq \gamma^* + h\cdot(\gamma^* - 1).$$

Thus,

$$\frac{\gamma}{\gamma^*} \leq 1 + h\cdot\left(1 - \frac{1}{\gamma^*}\right) \leq 1 + h.$$

□

We directly have the following result when $h = 1$.

Corollary 2: Our algorithm is a factor 2 approximation algorithm under the constraint that set-up time $s_{jk} \leq L$ for every ordered pair of jobs j and k.

Alternatively, Corollary 2 suggests that if $g_{jk} \in \{0, 1, 2\}$ for every ordered pair of jobs j and k, our algorithm is a factor 2 approximation algorithm. On the other hand, Theorem 2 means that if $g_{jk} \in \{1, 2\}$ for every ordered pair of jobs j and k, our algorithm is a factor 1.5 approximation algorithm. The assumption of $h = 1$ (i.e. $0 \leq s_{jk} \leq L$) is considered reasonable and proper when we try to design an approximation algorithm. Such problem formulation is considered in [7].

7. Conclusion

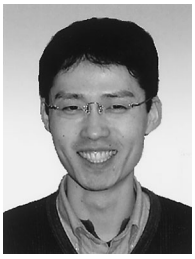
We have presented a heuristic algorithm for one-machine JIT-SP. Our algorithm is fast and has good performance on approximation ratio experimentally. We have also shown some results on approximation ratio under a constraint. Extending our algorithm to the case of m-machines is a future problem.

Acknowledgments

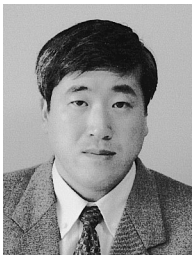
The authors would like to thank Shao Chin Sung, Arijit Bishnu (JAIST), and two anonymous referees for comments and suggestions.

References

[1] J. Błażewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz, Scheduling Computer and Manufacturing Processes, Second Edition, Section 3.1, Springer-Verlag, Berlin, 2001.
[2] K.R. Baker and G.D. Scudder, "Sequencing with earliness and tardiness penalties: A review," Oper. Res., vol.38, no.1, pp.22–36, 1990.
[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, Second Edition, Section 16.1, MIT Press, 2001.
[4] O. Čepek and S.C. Sung, "A quadratic time algorithm to maximize the number of just-in-time jobs on identical parallel machines," research report IS-RR-2004-003, School of Information Science, the Japan Advanced Institute of Science and Technology, 2004.
[5] K. Hiraishi, E. Levner, and M. Vlach, "Scheduling of parallel identical machines to maximize the weighted number of just-in-time jobs," Computers & Operations Research, vol.29, pp.841–848, 2002.
[6] K. Hiraishi, "Scheduling of parallel identical machines with multiple time slots," Proc. 4th Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty, Jindrichuv Hradec, pp.33–39, Sept. 2001.
[7] O. Čepek and S.C. Sung, "Just in time scheduling with periodic time slots," Proc. 5th Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty, pp.27–29, Osaka, Japan, Sept. 2002.
[8] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, San Francisco, 1979.
[9] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, Network Flows — Theory, Algorithms, and Applications, Prentice-Hall, Englewood Cliffs, New Jersey, 1993.
[10] C.H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice Hall, NJ, 1982.
[11] LEDA HP: http://www.algorithmic-solutions.com/enleda.htm



Eishi Chiba received the B.E. degree from Tohoku University in 2001, and the M.S. degree from the Japan Advanced Institute of Science and Technology (JAIST) in 2003. He is currently a doctoral student at JAIST. His main research interests include scheduling, discrete algorithms, combinatorial optimization and their applications.



Kunihiko Hiraishi received from the Tokyo Institute of Technology the B.E. degree in 1983, the M.E. degree in 1985, and D.E. degree in 1990. In 1985 he joined the IIAS-SIS, Fujitsu Limited. Since 1993 he has been with Japan Advanced Institute of Science and Technology, and is currently a Professor of School of Information Science. His current interests include theory and algorithm for concurrent systems. He is a member of the IEEE, IPSJ, and SICE.