Journal of the Operations Research Society of Japan
2004, Vol. 47, No. 2, 112-122

AN $O(n^2 \log^2 n)$ ALGORITHM FOR INPUT-OR-OUTPUT TEST IN DISJUNCTIVE SCHEDULING

Yuichiro Miyamoto
Sophia University

Takeaki Uno
National Institute of Informatics

Mikio Kubo
Tokyo University of Marine Science and Technology

(Received April 4, 2001; Revised February 24, 2004)

Abstract This paper is concerned with the input-or-output test that is a kind of interval capacity consistency tests. And an $O(n^2 \log^2 n)$ algorithm dealing with the test is proposed, where n denotes the number of jobs. In literature, an $O(n^4)$ algorithm has been known. The tests can be effectively used to reduce the search space of time- and resource-constrained scheduling problems. Computational results show that our new algorithm is about 3 times faster for instances of 30 jobs than the existing algorithm.

Keywords: Scheduling, interval consistency test, data structure

1. Introduction

Interval capacity consistency tests [5] consider the resource capacities available and required within certain time intervals. The goal of the tests is to draw conclusions that allow to rule out inadmissible job start times or sequences. The tests can be effectively used to reduce the search space of time- and resource-constrained scheduling problems. They have successfully been applied in algorithms for solving idealized problems such as the classical job shop scheduling problem (JSP) [2]. The tests can be applied in scheduling algorithms such as list scheduling or branch and bound procedures, or in constraint propagation based scheduling systems. The benefit of the tests is that they can reduce the search space and direct an algorithm towards good solutions. Some of interval capacity consistency tests are also known under the names of immediate selection, edge finding and energetic reasoning. The tests can also serve to derive tight lower bounds for make-span minimization problems. Here, we are only interested in the tests themselves and do not address scheduling algorithms in which they can be embedded. Since the tests only eliminate solutions incompatible with the capacity constraints, they are independent of the overall objective function to be optimized.

In this paper, we are concerned with disjunctive scheduling. An instance of the disjunctive scheduling problem consists of a set J of n jobs to be performed on one disjunctive resource, which can process only one job at a time. Jobs cannot be interrupted. The planning horizon is infinite, and the disjunctive resource is always available through the planning horizon. A release time r_j, a deadline d_j, and a processing time p_j characterize each job j of J. The relation $d_j \geq r_j + p_j$ holds $^{\forall} j = 1, \ldots, n$. The scheduling problem in this study is seen as a constraint satisfaction problem (CSP). The question is to find an assignment of the start time t_i of each job i within $[r_i,\ d_i - p_i]$ that satisfies the set of constraints of the problem.

In the recent years, constrained-based approaches have turned out to be an efficient way

to represent and solve $\mathcal{NP}$-hard scheduling problems such as JSP [3]. The input-or-output test is one of the interval capacity consistency tests which are based on resource constraints.

In branch and bound procedures that branch over disjunctive edges, the test may be employed to immediately select the orientation of edges, this process often called immediate selection, as first suggested by Carlier and Pinson [2], or edge finding, this term introduced by Applegate and Cook [1]. Using these tests, Carlier and Pinson [2] were for the first time able to optimally solve a notoriously difficult 10×10 JSP instance (Fisher and Thompson [6]) that — despite many attempts — had defied solution for over 25 years.

Domains of application of such rules concern an early detection of inconsistencies in time-resource constrained scheduling problems, and a support for the generation of solutions by pruning the search space without loss of solutions. Recent works using constraint propagation and especially the shaving technique to prune large parts of neighborhood report excellent results [4]. The use of constraint propagation in local search methods and particularly the shaving techniques employed are promising fields of research.

A basic idea of our algorithm in this paper was given in our previous work [7]. We modify it and report computational results.

The paper is organized as follows. In Section 2, the input-or-output test and other interval consistency tests for disjunctive scheduling problems are more precisely stated. In Section 3, the algorithm and its time complexity are presented. In Section 4, computational results are reported.

2. Input/Output Test and Input-or-Output Test

Figure 1 shows an example with a set $J = \{i, j, k\}$ of three jobs to be processed by the same resource. In the figure, a horizontal line represents an interval of time between r_j and d_j of j, a digit in a box represents p_j of j. We can deduce that i must be scheduled first in

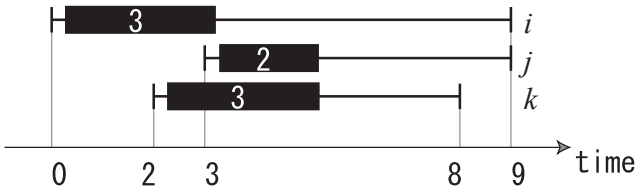

Figure 1: An example for the input test

the following way. Suppose i does not start first. Then all three jobs must be processed in the time interval $[2, 9)$. This means that a total processing time of $8 = 3 + 2 + 3$ must be scheduled in $7 = 9 - 2$ available time units, which is a contradiction. Thus we can conclude that i must start first. This observation leads to the following Lemma [5].

Lemma 1 (Input/output) *Let $i \in J' \subseteq J$. If*

$$\max_{j\in J'} d_j - \min_{j\in J'\setminus\{i\}} r_j < \sum_{j\in J'} p_j,$$

then i must be scheduled first in J' (input condition). Likewise, if

$$\max_{j\in J'\setminus\{i\}} d_j - \min_{j\in J'} r_j < \sum_{j\in J'} p_j,$$

then i must be scheduled last in J' (output condition). ∎

Figure 2 shows an example with a set $J = \{i, j, k, l\}$ of four jobs to be processed by the same resource. We can deduce that i must precede j. Suppose i is not scheduled first and j

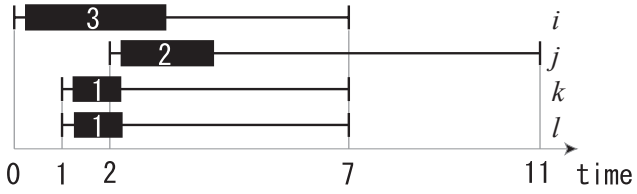

Figure 2: An example for the input-or-output test

is not scheduled last. Then all four jobs with a total processing time of $7 = 3+2+1+1$ must be scheduled within the interval $[1, 7]$, which is a contradiction. Hence we can conclude that it is impossible that at the same time i is not first and j is not last. If either i must be first or j must be last, then i must precede j. This observation leads to the following Lemma [5].

Lemma 2 (Input-or-output) *Let $i, j \in J' \subseteq J$. If*

$$\max_{k \in J' \setminus \{j\}} d_k - \min_{k \in J' \setminus \{i\}} r_k < \sum_{k \in J'} p_k, \tag{1}$$

then i must be scheduled first or j must be scheduled last in J'. If $i \neq j$ then i must precede j (input-or-output condition). ∎

In the context of the branch-and-bound method, our objective is to find all job pairs (i, j) that satisfy the equation (1). Here it is sufficient to find at least one subset J' for a pair (i, j) that satisfies the condition.

The develop of algorithms with lower polynomial time complexity for testing the input-or-output conditions is an open issue. The number of a subset J' is 2^n. It is unpractical to enumerate all subsets and to examine the condition (1).

Let $J' \subseteq J$ be a subset of jobs. Let $i, j \in J$ suffice $r_i < \min\{r_k \mid k \in J'\}$, $d_j > \max\{d_k \mid k \in J'\}$. And assume that $\max_{k \in J'} d_k - \min_{k \in J'} r_k < \sum_{k \in J' \cup \{i,j\}} p_k$. Then for all J'' $(J' \subseteq J'' \subset J)$, $\max\{d_k \mid k \in J'\} = \max\{d_k \mid k \in J''\}$, $\min\{r_k \mid k \in J'\} = \min\{r_k \mid k \in J''\}$ suffice $\max_{k \in J''} d_k - \min_{k \in J''} r_k < \sum_{k \in J'' \cup \{i,j\}} p_k$. From this observation, we obtain the following Corollary.

Corollary 1 (Input-or-output with an interval) *Let $r \in \{r_k \mid k \in J\}$ and $d \in \{d_k \mid k \in J\}$. Let $i \in J$ be an index that satisfies $d_i \leq d$. Let $j \in J$ be an index that satisfies $r_j \geq r$, $j \neq i$. Let $J' = \{\{k \in J \mid r_k \geq r,\ d_k \leq d\} \cup \{i, j\}\}$. If*

$$d - r < \sum_{k \in J'} p_k, \tag{2}$$

then i must be scheduled first or j must be scheduled last in J', and i must precede j. ∎

Our problem is to find all job pairs (i, j) that satisfy Lemma 2 with some job subset J'. That means to find all job pairs (i, j) that satisfy Corollary 1 with some interval $[r, d]$. The number of time intervals $[r, d]$ is $O(n^2)$. For a time interval, there are $O(n^2)$ pairs of jobs i and j to be examined. We can check the condition (2) in $O(n)$ time for a pair of jobs and a job interval, and hence there is an obvious $O(n^5)$ algorithm. By changing the order of examination, the total time complexity can be reduced from $O(n^5)$ to $O(n^4)$ [5]. And we have designed an $O(n^2 \log^2 n)$ algorithm.

3. Faster Algorithm for Input-or-Output Test

Though the input-or-output test could be applied to all pairs of jobs and all time intervals, our algorithm is applied to a pair of jobs i, j that satisfy

$$r_i < r_j \leq d_i < d_j, \tag{3}$$

and time intervals $[r, d]$ that satisfy

$$r_i < r \leq r_j,\ d_i \leq d < d_j. \tag{4}$$

In the other cases, the test is induced to an input test or an output test. Let

$$P_{[r,d]} = \sum_{i \in J,\ r_i \geq r,\ d_i \leq d} p_i,$$

and

$$f(r, d, i, j) = d - r - (P_{[r,d]} + p_i + p_j).$$

The condition $f(r, d, i, j) < 0$ means the input-or-output condition of a pair of jobs (i, j) for a time interval $[r, d]$ holds. Let

$$f_{interval}(r, d) = d - r - P_{[r,d]},$$
$$f_{pair}(i, j) = -p_i - p_j,$$

then

$$f(r, d, i, j) = f_{interval}(r, d) + f_{pair}(i, j).$$

The value $f_{interval}(r, d)$ depends on the time interval $[r, d]$, and the value $f_{pair}(i, j)$ depends on the pair of jobs (i, j). By calculating $f_{interval}(r, d)$ and $f_{pair}(i, j)$ separately, we design faster algorithms for the input-or-output test.

In this section, for a sake of clarity, we assume that

$$^{\forall} i \neq j \in J, r_i \neq r_j, d_i \neq d_j. \tag{5}$$

In the case of $^{\exists} i, j \in J,\ d_i = d_j$, our algorithm works correctly by applying lexicographical ordering.

3.1. $O(n^3 \log n)$ Algorithm

In this subsection, we show an $O(n^3 \log n)$ algorithm. The algorithm is rather simple, and plays a role in preparations for an $O(n^2 \log^2 n)$ algorithm. In this section, we assume that $d_1 < d_2 < \cdots < d_n$.

Let us consider a set of time intervals $\{[r, d_1], [r, d_2], \ldots, [r, d_n]\}$, and a pair of jobs (i, j) that satisfy the condition (3). Since there are $O(n)$ time intervals, it takes $O(n)$ time to calculate $f(r, d_1, i, j), f(r, d_2, i, j), \ldots, f(r, d_n, i, j)$. The time complexity of $O(n^2)$ pairs of jobs for $O(n)$ time intervals is $O(n^3)$. By using a balanced binary tree, we can reduce the time complexity to $O(n^2 \log n)$. For each $r = r_1, \ldots, r_n$, we calculate f in this way, thus the total time complexity is $O(n^3 \log n)$

We let BT be a balanced binary tree that has n leaves, i.e. the height of the tree is $O(\log n)$. In the tree, we let

- V be the set of vertices of BT,
- $LEAF = \{l_1, l_2, \ldots, l_n\} \subset V$ be the set of leaves of BT,

- $dleaf(v)$ be the set of descendant leaves of a vertex v in BT, $^{\forall}l \in LEAF, dleaf(l) = \{l\}$,
- $parent(v)$ be the parent of a vertex v.

In BT, each leaf l_i corresponds to a time interval $[r, d_i]$. We define a label of $l_i \in LEAF$ as

$$L_{interval}(l_i) = f_{interval}(r, d_i).$$

We define a label of $v \in V$ as

$$L_{interval}(v) = \min\{L_{interval}(l) \mid l \in dleaf(v)\}.$$

Let an interval $L(i, j) = \{l_i, l_{i+1}, \ldots, l_j\} \subseteq L$. To reduce the memory size representing the interval, we define $rep(i, j)$ as

$$rep(s, t) = \{v \in V \mid dleaf(v) \text{ is included in } L(s, t), \text{ but } dleaf(parent(v)) \text{ isn't}\}.$$

The size of $rep(s, t)$ is $O(\log n)$ for any s and t, and we take $O(\log n)$ time to obtain $rep(s, t)$, since all vertices of $rep(s, t)$ are children of vertices of the path connecting l_s and l_t. Each set $rep(s, t)$ has one-to-one correspondence to each interval $\{l_s, \ldots, l_t\}$.

We show an example of representative vertices in Figure 3.

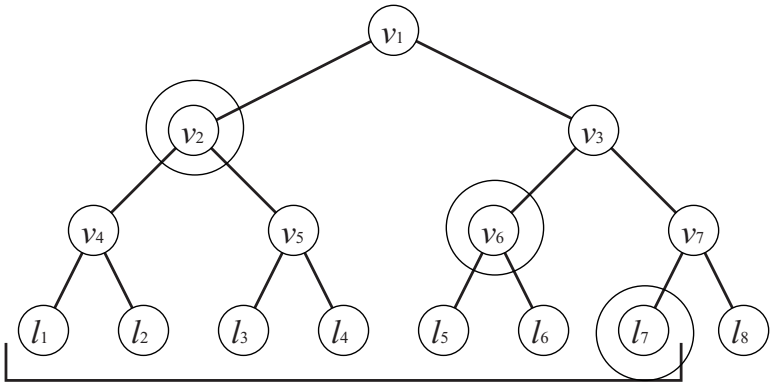

Figure 3: Representative vertices

In the example, $L = \{l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8\}$ and $rep(1, 7) = \{v_2, v_6, l_7\}$

From the definition of $L_{interval}$, $L_{interval}(v) + f_{pair}(i, j) < 0$ means

$$^{\exists}l \in dleaf(v),\ L_{interval}(l) + f_{pair}(i, j) < 0.$$

Hence it is sufficient to test the input-or-output condition for all vertices in $rep(s, t)$ instead of testing the input-or-output condition for all leaves $\{l_s, \ldots, l_t\}$. This is the key idea of the $O(n^3 \log n)$ algorithm.

The $O(n^3 \log n)$ algorithm consists of three procedures, initial procedure, update procedure and test procedure. In the initial procedure, we construct a binary tree BT and attach labels to vertices in BT. In the update procedure, we update labels of BT storing the sum of processing times in the time intervals and storing the sum of processing times of pairs of jobs. After the update procedure, we search pairs of jobs that satisfy the input-or-output condition. If we find such pairs, we output the pair. An abstract of the algorithm is below.

Abstract of $O(n^3 \log n)$ algorithm

Step1(initial procedure) Initialize a binary tree BT.
Step2(update procedure) If possible, update BT. Else, exit the algorithm.
Step3(test procedure) Test candidate pairs of jobs. Go to Step2.

Let $L_{pair}(v)$ be a set of values $f_{pair}(i,j)$ of v. The initial procedure is below.

Initial procedure

Step1 Put indices to jobs so that $d_1 < d_2 < \cdots < d_n$.
Step2 Construct a binary tree BT that has n leaves $\{l_1, l_2, \ldots, l_n\}$.
Let $head = \min\{r_i \mid i \in J\}$. Each leaf l_i corresponds to a time interval $[head, d_i]$.
Step3 Set $L_{interval}(l_i) = f_{interval}(head, d_i)$, $^{\forall} i \in J$.
Step4 Set $L_{interval}(v) = \min\{L_{interval}(l) \mid l \in dleaf(v)\}$, $^{\forall} v \in V$.
Step5 Set $L_{pair}(v) = \emptyset$ for all $v \in V$.

The time complexity of the initial procedure is $O(n)$ except for the sorting. The required memory space is $O(n)$, since each vertex $v \in V$ has just one label $L_{interval}(v)$.

Figure 4 shows an instance of the disjunctive scheduling problem.

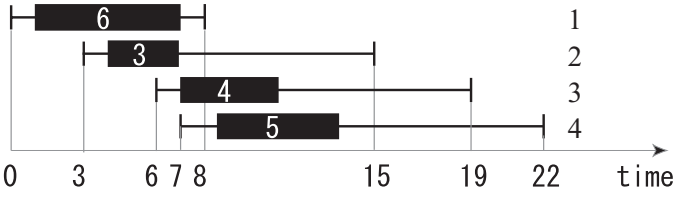

Figure 4: An instance of the disjunctive scheduling problem

In Figure 5, we show BT of the instance. In BT, $head = 0$. Leaves of BT correspond to time intervals $[0,8], [0,15], [0,19], [0,22]$, respectively. In the figure, the value in a box attached to the vertex v represents $L_{interval}(v)$.

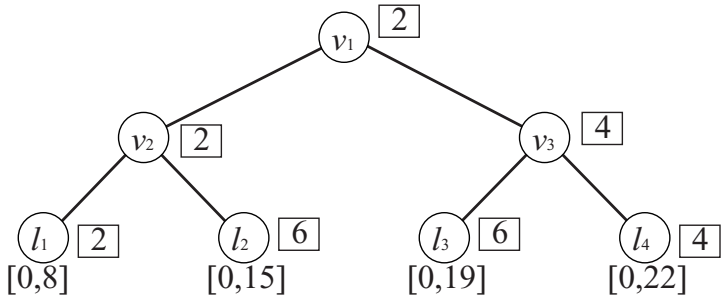

Figure 5: BT of the instance

Next we show the update procedure below.

Update procedure

Step1 Let i_* be a job that satisfies $head = r_{i_*}$.

Step2 If $\{r_i \mid r_i > r_{i_*},\ i \in J\} = \emptyset$, then stop.
else $head = \min\{r_i \mid r_i > r_{i_*},\ i \in J\}$.

Step3 For all $i \in J$, if $d_i < head$, then delete l_i from BT.
For all $v \in V$, if $dleaf(v) = \emptyset$, then delete v from BT.

Step4 For all $v \in V$, update $L_{interval}(v)$.

Step5 Let $pairs(i_*)$ be the set $\{i \in J \mid r_{i_*} < r_i < d_{i_*} < d_i\}$.

Step6 For all $i \in pairs(i_*)$, insert $f_{pair}(i_*, i)$ to $L_{pair}(v),\ v \in rep(i_*, i-1)$.

In Step1, we find a job i_* that is one of the pair of jobs examined in the following steps. In Step2, we update the time interval of the input-or-output condition in the followings. In Step3, we remove unnecessary leaves and vertices. If unnecessary leaves or vertices remain, an interval $[r, d],\ r > d$ might be considered in our algorithm. In Step5, the pair of jobs $(i_*, i),\ i \in pairs(i_*)$ are candidates of the test. In Step6, if $f_{pair}(i_*, i)$ is inserted to $L_{pair}(v),\ v \in rep(i_*, i)$, then the right hand of the equation (2) is over estimated.

The time complexity of Step3 is $O(n)$ in the bottom-up manner. In Step6, the time complexity of the insertions for all vertices in $rep(i_*, i)$ is $O(\log n)$. The time complexity of Step6 is $O(n \log n)$. The total time complexity of the update procedure is $O(n \log n)$. The required memory space increments is $O(n \log n)$

Let us see the instance. First, $head = 0$. Then we obtain $i_* = 1$, and $head$ is updated to 3. In Figure 6, labels $L_{interval}(v)$ are updated. Since $pairs(1) = \{2, 3, 4\},\ rep(1, 1) = \{l_1\},\ rep(1, 2) = \{v_2\},\ rep(1, 3) = \{v_2, l_3\},\ f_{pair}(1, 2) = -9,\ f_{pair}(1, 3) = -10,\ f_{pair}(1, 4) = -11$. In Figure 7, all values f_{pair} are attached to corresponding vertices of BT.

The test procedure is below.

Test procedure

Step1 For all $v \in V$,

Step1-1 For all $f_{pair}(i, j) \in L_{pair}(v)$, if $f_{pair}(i, j) + L_{interval}(v) < 0$, then
output the pair (i, j) since the pair (i, j) satisfies the input-or-output condition.

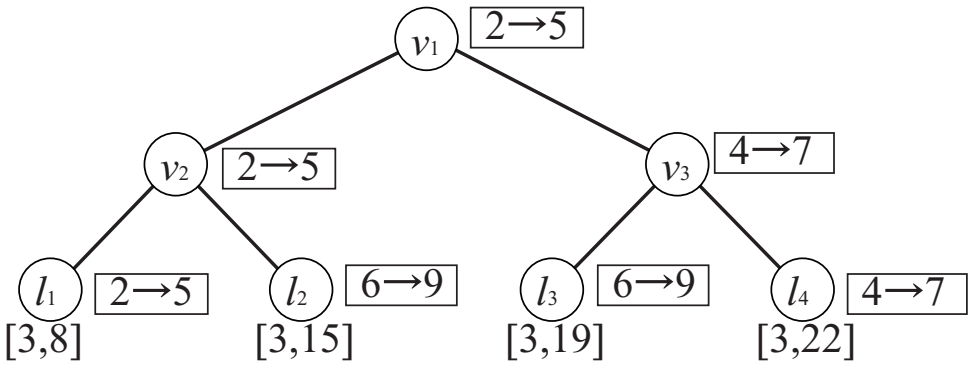

Figure 6: The update of $L_{interval}(v)$

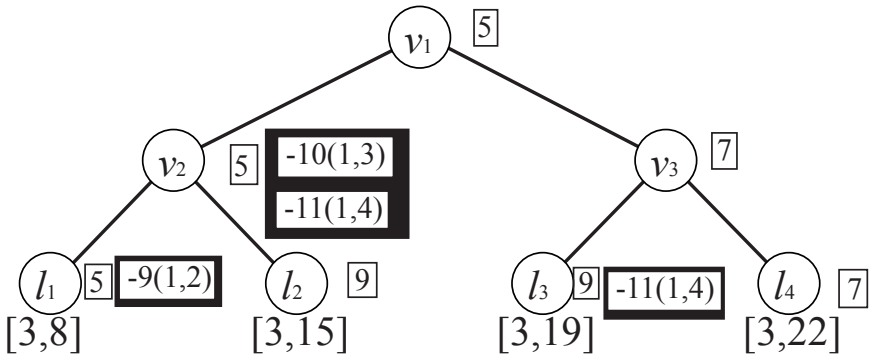

Figure 7: The update of BT

The total time complexity of the test procedure is $O(n^2 \log n)$, since at most $O(n^2)$ pairs of jobs can be examined and it occurs $O(\log n)$ test operation for a pair of jobs.

In the case of the instance, pairs $(1,2),\ (1,3),\ (1,4)$ satisfy the input-or-output condition.

3.2. $O(n^2 \log^2 n)$ Algorithm

In this subsection, we improve the $O(n^3 \log n)$ algorithm in previous subsection and obtain an $O(n^2 \log^2 n)$ algorithm. In the test procedure described in the above, we test all pair of jobs wastefully. In a new test procedure, we directly find pairs of jobs that satisfy the input-or-output condition. To realize it, we keep a set of labels $L_{pair}(v)$ in the non-increasing order by the heap structure.

Our $O(n^2 \log^2 n)$ algorithm consists of three procedures, initial procedure, update procedure and test procedure. The initial procedure of the $O(n^2 \log^2 n)$ algorithm is just the same with the initial procedure of the $O(n^3 \log n)$ algorithm. In the update procedure and the test procedure, we reduce the number of the check of the input-or-output condition by storing the sum of processing times of each pair of jobs in the non-increasing order. An abstract of the algorithm is the same with the abstract of the $O(n^3 \log n)$ algorithm.

Next we show the update procedure of the $O(n^2 \log^2 n)$ algorithm.

Update procedure

Step1–Step5 are same as $O(n^3 \log n)$ algorithm.

Step6 For all $i \in pairs(i_*)$,

Step6-1 For all $v \in rep(i_*, i)$, insert $f_{pair}(i_*, i)$ to $L_{pair}(v)$ so that $L_{pair}(v)$ is kept in

non-decreasing order by applying the heap structure.

In Step6-1, the time complexity of the insertions for all vertices in $rep(i_*, i)$ is $O(\log^2 n)$. The time complexity of Step6 is $O(n \log^2 n)$. The total time complexity of the update procedure is $O(n \log^2 n)$. The required memory space increments is $O(n \log n)$

After the update procedure, we search pairs of jobs that satisfy the input-or-output condition. If we find such pairs, we output the pair. The test procedure is below.

Test procedure

Step1 For all $v \in V$, repeat below steps.

Step1-1 If $L_{interval}(v) + \min\{L_{pair}(v)\} < 0$,
then output a pair of jobs (i, j) satisfying $\min\{L_{pair}(v)\} = f_{pair}(i, j)$.
Since the pair (i, j) satisfies the input-or-output condition.

Step1-2 Remove $f_{pair}(i, j)$ from the heap of $rep(i, j)$.

If each pair of jobs (i, j) has pointers to all $f_{pair}(i, j)$, then the time complexity of the deletions in Step1-2 is $O(\log^2 n)$ and the required memory space to the pointers is $O(\log n)$. Please note that $^{\forall} v \in V$, $L_{interval} + \min\{L_{pair}(v)\}$ cannot decrease because $L_{pair}(v)$ is sorted in non-decreasing order. The total time complexity of the test procedure is $O(n^2 \log^2 n)$. Because at most $O(n^2)$ pairs of jobs can be examined in the whole algorithm and it requires $O(\log^2 n)$ time to remove a pair of jobs from the heap of $L_{pair}(v)$.

In this section, we assume the condition (5). Our algorithm uses a binary tree whose each leaf corresponds to a time interval. Using lexicographical order of r_j, d_j in implementation, there is no problem whether the condition (5) is satisfied or not.

Theorem 1 *The algorithm find all pairs of jobs that satisfy the input-or-output condition in* $O(n^2 \log^2 n)$ *time and* $O(n^2 \log n)$ *memory space.*

Proof. It takes $O(\log^2 n)$ time to remove a pair of jobs from the heap and a job is never removed again after the job had been removed once. At most $O(n^2)$ deletions occur, hence the total time complexity of the test procedure is $O(n^2 \log^2 n)$. The initial procedure takes $O(n)$ time. Each update procedure takes $O(n \log^2 n)$ time, and the update procedure is called at most n time in our algorithm. Hence the total time complexity of our algorithm is $O(n^2 \log^2 n)$.

We also refer to the required size of memory space for computing. The number of all pairs of jobs are $O(n^2)$. Each pair pushes a value to $O(\log n)$ heaps and has $O(\log n)$ pointers for each value. Hence, the algorithm requires $O(n^2 \log n)$ memory space. ∎

4. Computational Results

Our goal in this section is to clarify the number of jobs in which our algorithm is faster than the obvious one. We have implemented this algorithm in C++ on a personal computer IBM 2662-34J*, and tested it. We generate JSP instances randomly, and use sub problems (disjunctive scheduling) that arise in the branch and bound procedure for the JSP instances. These instances are tested with new $O(n^2 \log^2 n)$ algorithm and the existing $O(n^4)$ algorithm. The number of jobs of instances are $10, 20, \ldots, 150$.

*OS: Windows 98, CPU: Pentium III 500MHz, RAM: 128MByte.

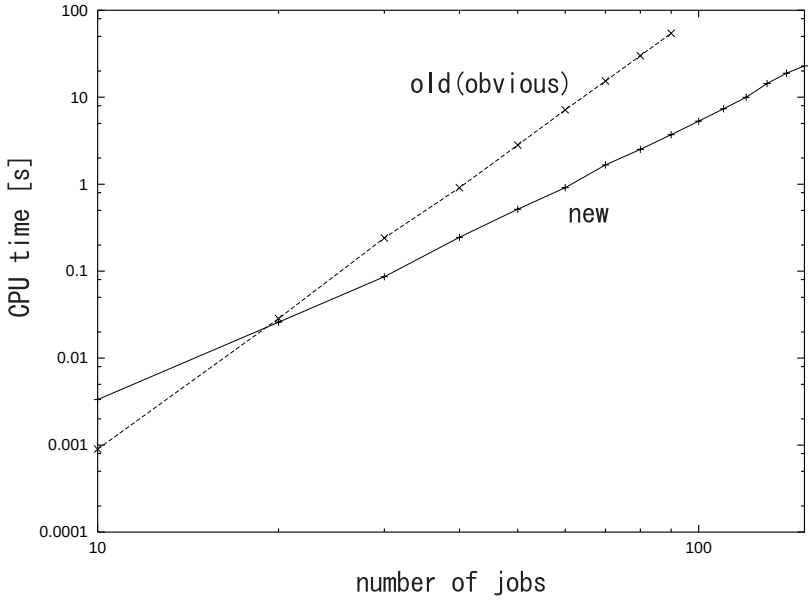

Figure 8: CPU time of the new algorithm and the previous algorithm

In Figure 8, we compare new $O(n^2 \log^2 n)$ algorithm with the existing $O(n^4)$ algorithm. In the figure, a point represent an average of CPU time of 100 instances of same size(number of jobs). Our new algorithm have a great advantage in terms of CPU time. In researches of exact algorithm of JSP, we want to solve instances with about 30 jobs. For these instances, our new algorithm is about 3 times faster than the obvious algorithm. For larger instances, we apply the meta-heuristics to get good solutions. Our new algorithm is about 20 times faster than the obvious algorithm, for the instances with about 100 jobs that are the target of the meta-heuristics.

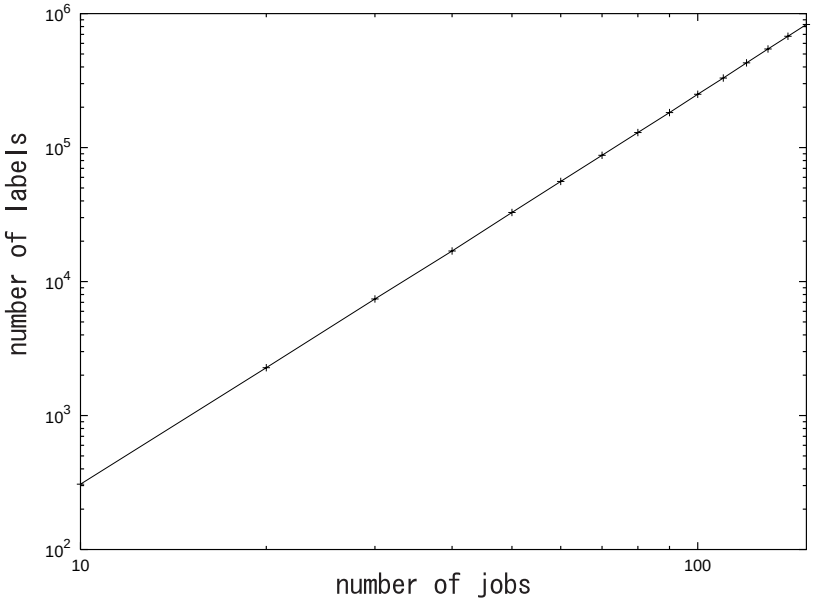

Figure 9: The number of labels required by the proposed $O(n^2 \log^2 n)$ algorithm

In Figure 9, we present the maximum size of $\cup_v L_{pair}(v)$ that is required by the new

algorithm. The theoretical maximum size of $\cup_v L_{pair}(v)$ in the algorithm is $O(n^2 \log n)$. In the figure, a point represent an average of the maximum size of $\cup_v L_{pair}(v)$ of 100 instances.

5. Conclusion

In this paper we have considered the input-or-output test for disjunctive scheduling. More precisely, we propose an $O(n^2 \log^2 n)$ algorithm for the input-or-output test. For the algorithm, data structures are most important; a binary tree and heaps attached to the tree make possible lower polynomial time complexity. This result could deserve to be delved, in particular as an eventual support for local search to obtain good upper bounds.

Finally, we claim that the results presented in this paper are not restricted to disjunctive scheduling problems. Indeed they can contribute to derive strong deductions also for the resource constrained project scheduling problem.

Acknowledgements

The authors are grateful to the anonymous referees for their helpful comments.

References

[1] D. Applegate and W. Cook: A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, **3** (1991) 149–156.

[2] J. Carlier and E. Pinson: An algorithm for solving the job-shop problem. *Management Science*, **35** (1989) 164–176.

[3] J. Carlier and E. Pinson: Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, **78** (1994) 146–161.

[4] Y. Caseau and F. Laburthe: Effective forget-and-extend heuristics for scheduling problems. *Proceedings of the Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems*, (Ferrara, Italy, 1999).

[5] U. Dorndorf, T. P. Huy, and E. Pesch: A survey of interval capacity consistency tests for time and resource-constrained scheduling. J. Weglarz (ed.): *Project scheduling—recent models, algorithms and application*, (Kluwer, LD, 1999), 213–238.

[6] H. Fisher and G. Thompson: Probabilistic learning combinations of local job-shop scheduling rules. J. Muth and G. Thompson (eds.): *Industrial Scheduling*, (Prentice-Hall, Englewood Cliffs, NF, 1963), 225–251.

[7] Y. Miyamoto, T. Uno and M. Kubo: Speeding up immediate selections on job shop scheduling. *Proceedings of the Twelfth RAMP Symposium* (2000), 43–52 (in Japanese).

Yuichiro Miyamoto
Department of Mechanical Engineering
Faculty of Science and Technology
Sophia University
7-1 Kioi-cho, Chiyoda-ku, Tokyo 102-8554 Japan.
E-mail: y-miyamo@sophia.ac.jp