Journal of the Operations Research
Society of Japan
Vol. 24, No. 2, June 1981

ALGORITHM FOR ONE MACHINE JOB SEQUENCING WITH PRECEDENCE CONSTRAINTS

Tetsuo Ichimori
Osaka University

Hiroaki Ishii
Osaka University

Toshio Nishida
Osaka University

(Received July 28, 1980; Revised November 4, 1980)

Abstract Consider the problem of sequencing n jobs with general precedence constraints on a single machine to minimize total weighted completion time, which is NP-complete. However some class of problems can be solved in polynomial time, i.e., the problems whose precedence constraints are trees or series parallel. This paper proposes an $O(n^4)$ algorithm for implementing a decomposition procedure due to Sidney. It also shows that if each decomposed subgraph is series parallel, then the problem is solved in $O(n^5)$ running time, even if the whole graph is not series parallel.

1. Introduction

Consider the following scheduling problem. A set of jobs $J_1, J_2, \ldots, J_n$, with each J_i associated a positive processing time p_i and a weight w_i, are to be processed without preemption on one machine. Furthermore precedence constraints $<$ among jobs are given. $J_i < J_j$ means J_i must precede J_j. Given a feasible schedule, the completion time C_i of job J_i is well determined. The objective is to find the optimum sequence minimizing the weighted sum of the completion times, $\sum_{i=1}^{n} w_i C_i$, under precedence constraints.

Smith [9] proved that, with no precedence constraints, the optimum sequence could be obtained by ordering the jobs according to nonincreasing order of w_i/p_i ratio. For the case in which precedence constraints are trees (by the transitive reduction), Horn [3] and Sidney [8] gave $O(n \log n)$ algorithms.

Furthermore Lawler [5] gave an $O(n \log n)$ algorithm for the case in which precedence constraints are series parallel. However, when precedence constraints are general, Lawler [5] proved that the problem is NP-complete. For the problem, Sidney [8] gave the following decomposition algorithm, which can produce the optimum sequence.

Sidney Decomposition Algorithm

Step 1. Identify the optimum initial set I^* maximizing $(\sum_{i\in I} w_i)/(\sum_{i\in I} p_i)$, where I is an *initial set*, i.e., some of jobs plus all predecessors of them.

Step 2. Schedule the members of I^* and remove I^* from the whole of remaining jobs and go back to Step 1.

For implementing Step 1 of Sidney algorithm Lawler [5] gave an algorithm with running time bounded by a polynomial in the length of the input, which means Step 2 is NP-complete. However the computation time of Lawler algorithm (in Step 1 of Sidney algorithm) depends on the ranges of p_i and w_i (it is $O(n^3 \log p^* w^*)$ where $p^* = \max p_i$ and $w^* = \max w_i$), moreover p_i and w_i are restricted to integers. In order to improve these drawbacks an alternative algorithm is proposed which runs in $O(n^4)$ time. Namely, the alternative algorithm is superior to Lawler's in that it does not require the integrality of p_i or w_i, nor does the running time of it depend on the ranges of p_i or w_i. Hence it can be shown that if precedence constraints with respect to each I^* are series parallel, then the optimum sequence is obtained in $O(n^5)$ running time.

The organization of this paper is as follows. First we briefly review *maximum closure* and *fractional programming* in Section 2. Then we propose the alternative algorithm of finding the optimum initial set I^* in time $O(n^4)$ in Section 3. Then we discuss the complexity of it in Section 4. Finally a simple example is worked out in Section 5.

2. Maximum Closure and Fractional Programming

The object of this paper is to maximize $(\sum_{i\in I} w_i)/(\sum_{i\in I} p_i)$ over all the initial sets I. If the objective function is not fractional but linear, then the problem becomes that of finding the maximum closure of a graph (see Picard [7] for details). On the other hand, if the above maximization is taken over all non-empty subsets of jobs, disregarding precedence constraints among them, then it is reduced to the ordinary fractional programming problem (see Dinkelbach [1] for details). Consequently we review maximum closure and fractional programming.

Let $G=(N,A)$ be an acyclic directed graph where N is the set of nodes (1), $(2),\ldots,(n)$ and A the set of arcs connecting these nodes. A closure of G is defined as a subset of nodes Y such that if a node belongs to Y, then all its successors also belong to Y. If with each node (i) is associated a real

number m_i, then the maximum closure Y^* of G is defined as a closure of maximum value, where the value of a closure Y is defined by $\sum_{i \in Y} m_i$.

Picard [7] demonstrates that the problem of finding a maximum closure of a graph G is equivalent to solving the maximum flow problem in a network formed by the graph G with infinite capacities on its arcs, a source linked to each node (i) such that $m_i > 0$ by an arc of capacity $(+m_i)$ and a sink linked from each node (i) such that $m_i < 0$ by an arc of capacity $(-m_i)$. Consequently, the maximum closure problem can be reduced to that of finding a maximum flow in the network formed as above.

Now we turn to the following fractional problem:

$$\text{maximize } (\sum_{i \in X} a_i)/(\sum_{i \in X} b_i)$$

where X is a non-empty subset of n objects $(1),(2),\ldots,(n)$, positive a_i and positive b_i are assigned to object (i). Dinkelbach [1] gave the following algorithm of solving the fractional problem posed above (Step 1 in Algorithm 1 below is modified slightly in order to start with a good initial value).

Algorithm 1

Step 1. Set $q_1 = \min_{i=1,2,\ldots,n} a_i/b_i$ and proceed to Step 2 with $k=1$.

Step 2. Solve the following problem:

$$F(q_k) = \max \sum_{i \in X} (a_i - q_k b_i),$$

and denote the solution subset of objects by X_k.

Step 3. If $F(q_k) = 0$, then X_k is optimum, otherwise compute $q_{k+1} = (\sum_{i \in X_k} a_i)/(\sum_{i \in X_k} b_i)$ and return to Step 2 after setting k to $k+1$.

Lemma 1. Let $q^* = \max (\sum_{i \in X} a_i)/(\sum_{i \in X} b_i)$, then the following holds:

(1) If $q_k < q^*$, then $F(q_k) > 0$.

(2) If $q_k = q^*$, then $F(q_k) = 0$.

(3) If $q_k > q^*$, then $F(q_k) < 0$.

(4) $F(q_1) \geq 0$, (since $\min_{i=1,2,\ldots,n} a_i/b_i \leq q^*$).

(5) $q_1 < q_2 < \ldots < q_k < \ldots \leq q^*$.

(6) $\sum_{i \in X_k} (a_i - q_{k+1} b_i) = 0$.

Proof: See Dinkelback [1]. □

Remark 1. For the latter discussion, if $X_k = \phi$ (empty set) for q_k, which is forbidden here, we define $F(q_k) = 0$.

3. Alternative Procedure for I*

Let $G=(N,A)$ show job J_i as node (i) and a precedence constraint $J_i < J_j$ as arc (j,i) in order to correspond "predecessors" in the scheduling problem to "successors" in the maximum closure problem, i.e., in order to identify an initial set I with a closure Y. (Note, in Algorithm 2 described later, the relation $J_i < J_j$ is shown by arc (i,j) as usual.)

In order to find I^* we modify Algorithm 1 as follows. Replace a_i and b_i by w_i and p_i, respectively. Let X be a closure (possibly empty) of G.

The above modification derives the maximum closure problem where a real number $m_i = w_i - q_k p_i$ is associated with each node (i). As mentioned above, this problem can be reduced to that of finding a maximum flow in the corresponding network. As soon as a maximum flow in the network with respect to q_k is found, the solution subset S_k (corresponding to X_k at Step 2), which is a maximum closure with respect to q_k, is at hand, for the minimum cut with respect to source s and sink t is $(\{s\} \cup S_k, \{t\} \cup T_k)$ where $T_k = N - S_k$ [7]. Though at Step 2 an empty set is excluded as a solution, the solution subset S_k is now allowed to be empty. Hence for q_k such that $F(q_k) < 0$ if an empty set is excluded, we have $F(q_k) = 0$ if it is allowed. (See Remark 1 of Algorithm 1.) In other words if $q_k > q^*$, then we have $F(q_k) = 0$ instead of $F(q_k) < 0$. For q_k such that $q_k < q^*$ there must be non-empty solution subset S_k since $F(q_k) > 0$ from Lemma 1-(1), however it is possible that S_k becomes empty from the nature of *labeling method* due to Ford and Fulkerson [2], which finds the minimum cut "nearest" to source s [2, Theorem 5.5]. In order to circumvent such difficulty, it is sufficient to reverse directions of all arcs and to interchange source and sink. Now we delineate the algorithm of scheduling all jobs.

Algorithm 2

Step 0. Construct graph G which consists of n nodes $\{(i)\}$ corresponding to n jobs $\{J_i\}$, source node s, sink node t and arcs $\{(i,j)\}$ with infinite capacity corresponding to precedence constraints $\{J_i < J_j\}$.

Step 1. Set q_1 to max w_i/p_i where the maximization is taken over nodes without predecessors. Proceed to Step 2 with $k=1$.

Step 2. Compute $m_i = w_i - q_k p_i$ for each (i).

Step 3. For each node (i), augment the graph G by arc (s,i) with capacity $(-m_i)$ if $m_i < 0$ and arc (i,t) with capacity $(+m_i)$ if $m_i > 0$.

Step 4. Find a maximum flow of the augmented graph and its corresponding minimum cut $(\{s\} \cup S_k, \{t\} \cup T_k)$.

Step 5. If $F(q_k) = \sum_{i \in T_k} (w_i - q_k p_i) = 0$, then go to Step 6 setting I^* to T_k, otherwise compute $q_{k+1} = (\sum_{i \in T_k} w_i)/(\sum_{i \in T_k} p_i)$ and go to Step 2

after setting k to $k+1$.

Step 6. Schedule the jobs in I^* and remove I^* from G. If the resulting graph G is empty, halt. Else go to Step 1.

Lemma 2. For q_1 determined at Step 1,

$$q_1 \leq (\sum_{i \in I^*} w_i)/(\sum_{i \in I^*} p_i).$$

Proof: Since I^* is a non-empty subset and the node determining q_1 is an initial set, we have the lemma. □

Remark 2. Step 6 is easily implemented by the $O(n \log n)$ time algorithm due to Lawler if the subgraph corresponding to initial set T_k is series parallel. Otherwise, a heuristic due to Morton [6] can be used.

4. Number of Operations

In this section we establish an upper bound on the worst-case running time for Algorithm 2 in the case where precedence constraints with respect to each I^* are series parallel.

Theorem 1. Let $(\{s\} \cup S_k, \{t\} \cup T_k)$ and $(\{s\} \cup S_{k+1}, \{t\} \cup T_{k+1})$ be the minimum cuts at the kth and $(k+1)$st iterations of Algorithm 2, respectively. Then if $q_{k+1} \neq q^*$, T_{k+1} is a proper subset of T_k, or $|T_k| - 1 \geq |T_{k+1}|$, where $|X|$ denotes the cardinality of the set X.

Proof: Let N partition into four subsets: $A = S_k \cap S_{k+1}$, $B = S_k \cap T_k$, $C = T_k \cap S_{k+1}$, and $D = T_k \cap T_{k+1}$, see Fig. 1. We first show by contradiction that $B \neq \phi$.

Since now T_k, but not S_k, corresponds to X_k of Step 2 of Algorithm 1 (note the directions of arcs), $T_k = C \cup D$ maximizes $\sum(w_i - q_k p_i)$. Hence it follows that

$$\sum_{i \in C \cup D \cup B} (w_i - q_k p_i) \leq \sum_{i \in C \cup D} (w_i - q_k p_i),$$

which results in

$$\sum_{i \in B} (w_i - q_k p_i) \leq 0. \tag{1}$$

Similarly, since $T_{k+1} = B \cup D$ maximizes $\sum(w_i - q_{k+1} p_i)$, it follows that

$$\sum_{i \in B \cup D} (w_i - q_{k+1} p_i) \geq \sum_{i \in D} (w_i - q_{k+1} p_i),$$

which results in

$$\sum_{i \in B} (w_i - q_{k+1} p_i) \geq 0. \tag{2}$$

Since $q^* > q_{k+1} > q_k$ ((5) of Lemma 1) and $p_i > 0$, we have

$$\sum_{i \in B} (w_i - q_k p_i) > \sum_{i \in B} (w_i - q_{k+1} p_i). \tag{3}$$

From (1), (2) and (3) we have a contradiction. Hence $B = \phi$ is obtained. This implies $T_k \supseteq T_{k+1}$. From (1), (6) of Lemma 1 and $q_{k+1} \neq q^*$, however, we have

$$\sum_{i \in T_{k+1}} (w_i - q_{k+1} p_i) > 0$$

and

$$\sum_{i \in T_k} (w_i - q_{k+1} p_i) = 0.$$

Hence $T_k = T_{k+1}$ is impossible. Therefore $T_k \supset T_{k+1}$, or $|T_k| - 1 \geq |T_{k+1}|$. □

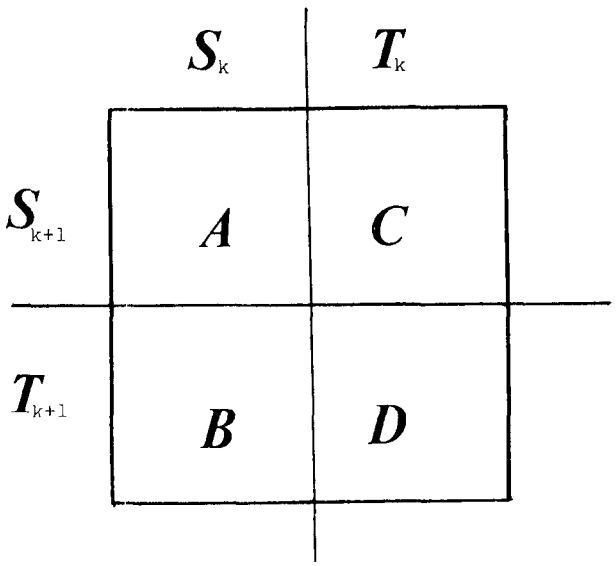

Fig. 1.

Since $|T_1| \leq n$, the preceding Theorem 1 implies that the number of iterations of Algorithm 2 for obtaining each I^* is at most n. The computationally most expensive step among Steps 1 though 5 is Step 4 requiring $O(n^3)$ per iteration. (Note, algorithms of finding a maximum flow can run in $O(n^3)$ time. For example see [4].)

From what has been mentioned above each I^* is obtained in time $O(n^4)$. In additon since $|I^*| \geq 1$ and Steps 1 and 6 are negligible as compared with Step 4, the optimum sequence of n jobs is obtained in $O(n^5)$ time if precedence constraints with respect to each I^* are series parallel.

5. Example

Consider the following problem [8]:

$p_1 = 5$, $p_2 = 8$, $p_3 = 3$, $p_4 = 5$, $p_5 = 3$, $p_6 = 7$, $p_7 = 6$, $w_i = 1$ for $i = 1,2,\ldots,7$,

$J_1 < J_3$, $J_1 < J_4$, $J_2 < J_4$, $J_2 < J_5$, $J_3 < J_7$, $J_4 < J_6$, $J_5 < J_6$, $J_6 < J_7$.

Algorithm 2 proceeds as follows:

Step 0. G is shown in Fig. 2.

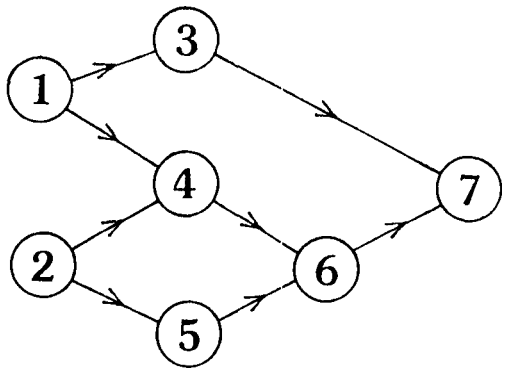

Fig. 2. Graph G.

($k = 1$)

Step 1. $q_1 = \max\{w_1/p_1, w_2/p_2\} = \max\{1/5, 1/8\} = 1/5$.

Step 2. $m_1 = 0$, $m_2 = -3/5$, $m_3 = 2/5$, $m_4 = 0$, $m_5 = 2/5$, $m_6 = -2/5$, $m_7 = -1/5$.

Step 3. Flow network is shown in Fig. 3.

Step 4. Minimum cut is $(\{s\} \cup S_1, \{t\} \cup T_1)$ where $T_1 = \{(1),(3)\}$.

Step 5. $F(q_1 = 1/5) = \sum_{i \in T_1} (w_i - q_1 p_i) = m_1 + m_3 = 2/5$. $q_2 = (w_1 + w_3)/(p_1 + p_3) = 1/4$.

($k = 2$)

Step 2. $m_1 = -1/4$, $m_2 = -1$, $m_3 = 1/4$, $m_4 = -1/4$, $m_5 = 1/4$, $m_6 = -3/4$, $m_7 = -1/2$.

Step 3. See Fig. 4.

Step 4. We have $(\{s\} \cup S_2, \{t\} \cup T_2)$, where $T_2 = \{(1),(3)\}$.

Step 5. $F(q_2 = 1/4) = m_1 + m_3 = 0$. $I^* = \{(1), (3)\}$.

Step 6. The optimum sequence of I^* is (1)-(3). A new graph removed by I^* is shown in Fig. 5.

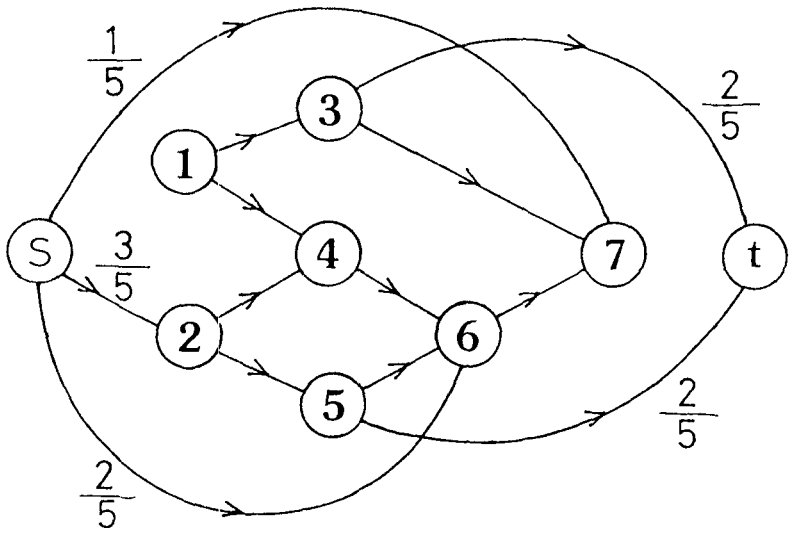

Fig. 3. Capacities are shown on arcs. Absence of them means infinite capacity.

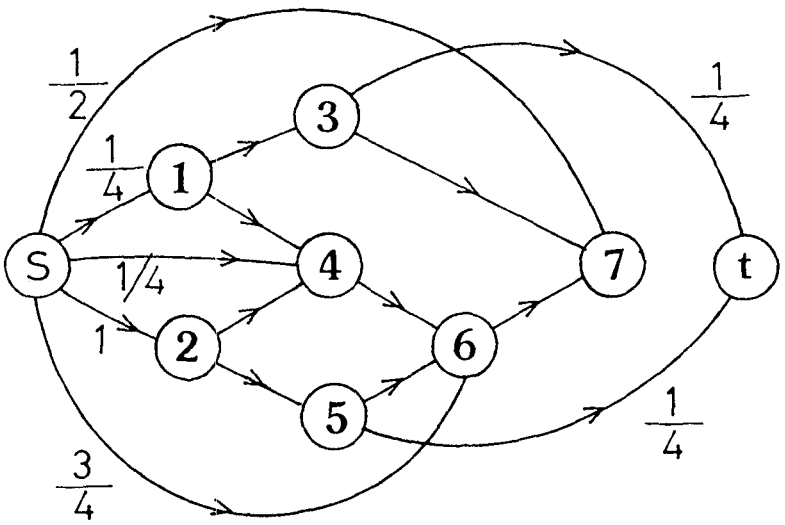

Fig. 4.

Following the same line as above, the next optimum initial set is $I^* = \{(2),(4),(5)\}$ and its optimum sequence is (2)-(5)-(4). The next new graph is shown in Fig. 6. Therefore the sequence (1)-(3)-(2)-(5)-(4)-(6)-(7) is optimum.

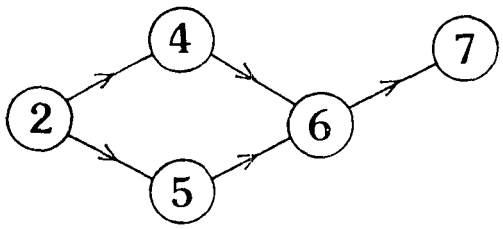

Fig. 5. New graph.

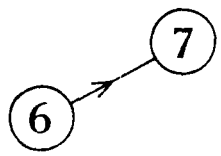

Fig. 6. Next new graph.

References

[1] Dinkelbach, W.: On Nonlinear Fractional Programming. *Management Science*, Vol.13, No.7 (1967), 492-498.

[2] Ford, L. R. and Fulkerson, D. R.: *Flows in Networks*, Princeton University Press, Princeton, 1962.

[3] Horn, W. A.: Single-Machine Job Sequencing with Treelike Precedence Ordering and Linear Delay Penalties. *SIAM Journal on Applied Mathematics*, Vol. 23, No.2 (1972), 189-206.

[4] Karzanov, A. V.: Determining the Maximum Flow in a Network by the Method

of Preflows. *Soviet Mathematics Doklady*, Vol.15, No.2 (1974), 434-437.

[5] Lawler, E. L.: Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints. *Annals of Discrete Mathematics*, Vol. 2 (1978), 75-90.

[6] Morton, T. E. and Dharan, B. G.: Algoristics for Single-Machine Sequencing with Precedence Constraints. *Management Science*, Vol.24, No.10 (1978), 1011-1020.

[7] Picard, J. C.: Maximal Closure of a Graph and Application to Combinatorial Problems. *Management Science*, Vol.22, No.11 (1976), 1268-1272.

[8] Sidney, J. B.: Decomposition Algorithms for Single-Machine Sequencing with Precedence Relations and Deferral Costs. *Operations Research*, Vol.23, No. 2 (1975), 283-298.

[9] Smith, W. E.: Various Optimizers for Single-Stage Production. *Naval Research Logistics Quarterly*, Vol.3 (1956), 59-66.

Tetsuo ICHIMORI: Department of Applied Physics, Faculty of Engineering, Osaka University, Yamada-Oka, Suita, Osaka 565, Japan.