Journal of the
Operations Research
Society of Japan

VOLUME 16 *June 1973* NUMBER 2

AN ALGORITHM FOR GENERAL SCHEDULING PROBLEM[1]

MASAAKI YAMAMOTO

Hōsei University

(Received February 7, 1972)

Abstract

For solving general scheduling problem (including job-shop problem), order-assignment model is proposed. In this model the scheduling problem results in the selection of resource orders which are assigned for the purpose of resolving resource constraints, and the sequence of this selection constructs a search tree. The proposed algorithm for searching an optimal schedule is based on a branch-and-bound method, in which the order of selection of resource orders is carefully controled.

1. Introduction

A set of operations which are necessary for the completion of a project or a group of jobs constitutes a network by the technological

[1] The contents of the paper was partly reported at Autumn Conference of ORSJ in 1970.

orders between some operations. By introducing resource orders to this network, we can get a feasible schedule which satisfies constraints of available resources. Therefore, the selection of optimal resource orders for a given objective function gives a solution of scheduling problem. Such a type of model was investigated by the author in reference [5], [6]. The purpose of this paper is to report on an improved method of solution for the same type of model. Job-shop problem is included by this model, for the sequences of operations concerning each job can construct a network by connecting their first operations with a starting node and their last operations with a terminal node. The model by integer linear programming [3] and the representation by disjunctive graph [1], [2] have essentially the same structure as the proposed model, and we intend to present a more efficient method of solution.

2. Description of Problem

We represent the network of operations by an oriented graph $G=(N, A)$, where N is the set of nodes, which are made up of node i $(i=1, 2, \cdots, n)$ corresponding to n given operations with a starting node s and a terminal node t. Each node has a given processing-time d_i, especially $d_s=d_t=0$.

By technological ordering constraints, the operation set N makes a partially ordered set by order relation "$\succ$", which means that one operation precedes another. For $x, y \in N$, if $x\succ y$ and if there is not z such that $x\succ z\succ y$, we say that x and y have a directly order relation "$\succ\succ$", that is $x\succ\succ y$. If a directed arc (i, j) corresponds to the directly order relation $i\succ\succ j$, we can get the set of arcs $A=\{(i_1, j_1), (i_2, j_2), \cdots\}$, which is called technological order set.

If $i\succ i$ $(i\in N)$, we say that there is a loop in graph G. We call graph $G=(N, A)$ schedule network S^0 if there is no loop in G. From the directly order relation in S^0, predecessor set PJ_j and successor set SJ_j are defined, respectively:

$$(1)\qquad \begin{cases} PJ_j=\{i|i\succ\succ j\} \\ SJ_j=\{k|j\succ\succ k\} \end{cases}$$

In the schedule network we can compute schedule time for each operation by the same method as PERT technique [4]. Earliest schedule time is given by

$$(2)\qquad \begin{cases} ES_s=0 \\ ES_j=\max\limits_{i\in PJ_j} EF_i & (j=1,\ 2,\ \cdots,\ n,\ t) \\ EF_j=ES_j+d_j\,. & (j=s,\ 1,\ 2,\ \cdots,\ n,\ t) \end{cases}$$

We define critical path length λ by

$$(3)\qquad \lambda=EF_t\ (=ES_t)$$

and especially λ^0 for $S^0=(N,\ A)$.

Conversely, latest schedule time is

$$(4)\qquad \begin{cases} LF_t=\lambda^0 \\ LF_j=\min\limits_{k\in SJ_j} LS_k & (j=s,\ 1,\ 2,\ \cdots,\ n) \\ SL_j=LF_j-d_j\,. & (j=s,\ 1,\ 2,\ \cdots\ n,\ t) \end{cases}$$

The assumptions of resources necessary for performing each operation are:

1) We can use only one resource of each type in the project.
2) Each resource can be used by at most one operation at a time.
3) Once an operation is started on some resources it must be completed before another operation can begin on the same resources.
4) Each operation cannot start before all necessary resources are ready for this operation.

A subset of N containing the operations which require a common type of resource is called common resource set R_k ($k=1,\ 2,\ \cdots,\ m$). For any $i,\ j\in R_k$, either $i\succ\succ j$ or $j\succ\succ i$ must be held for the assurance of feasibility for resource constraints, in other word, either arc $(i,\ j)$ or $(j,\ i)$ must be introduced to graph G, which is called resource orders. A set of all possible node pair i-j for a resource $k(i,\ j\in R_k)$

is called P_k, and $P=\bigcup_{k=1}^{m} P_k$ for all resources. $|P|$, the number of elements of P, is $\sum_{k=1}^{m} {}_{|R_k|}C_2$. For each element of P, i-j, either arc (i, j) or (j, i) is assigned, and the resulting arc set is called resource order set B. We use notation B^h to emphasize that a set B is constructed by a special selection h. The possible number of selections is $2^{|P|}$. If $G=(N, A\cup B^h)$ has no loop, this selection gives a schedule network S^h. Then we can define:

Problem: Given $S^0=(N, A)$, R_k, and d_i, select $S=(N, A\cup B^h)$ minimizing λ.

3. Properties of the Schedule Network

Consider graph $G=(N, A\cup Z)$, where Z is the set of arcs which are already assigned at intermediate stage of the selection, that is $Z\subseteq B$. We can prove the following properties for G.

Proposition 1: When a resource order $I\succ\succ J$ $(I, J\in R_k)$ is introduced into $S=(N, A\cup Z)$, let $\Delta\lambda_{IJ}$ be the increment of the longest path length containing arc (I, J) in comparison with λ^0. Then

(5) $\Delta\lambda_{IJ}=EF_I-LS_J\,.$

Proof: By the definition of schedule time (2), (4), the longest path length from s to I is EF_I and from J to t is (λ^0-LS_J). Then

$$\begin{aligned}\Delta\lambda_{IJ}&=\{EF_I+(\lambda^0-LS_J)\}-\lambda^0\\&=EF_I-LS_J\end{aligned}$$

Proposition 2: By introducing resource orders $i_1\succ\succ j_1$, $i_2\succ\succ j_2$, $\cdots$, $i_l\succ\succ j_l$ into $S^0=(N, A)$ one after another we have $S=(N, A\cup Z)$, where $Z=\{(i_1, j_1), (i_2, j_2), \cdots, (i_l, j_l)\}$. Then the critical path length of S is:

(6) $\lambda=\lambda^0+\max(0,\ \Delta\lambda_{i_1j_1},\ \Delta\lambda_{i_2j_2},\ \cdots,\ \Delta\lambda_{i_lj_l})\,.$

We denote the second term of light hand side of (6) by $\Delta\lambda_{\max}$.

Proof: Starting from $S^0=(N, A)$, we introduce an arc (i, j) into S one by one. From Prop. 1, critical path length at each stage is, iteratively,

$$\lambda^1=\lambda^0+\max(0,\ \Delta\lambda_{i_1j_1})$$

$$\begin{aligned}\lambda^2&=\max\{(\lambda^0+\max(0,\ \Delta\lambda_{i_2j_2})),\ \lambda^1\}\\&=\lambda^0+\max(0,\ \Delta\lambda_{i_1j_1},\ \Delta\lambda_{i_2j_2})\\&\ \vdots\\\lambda&=\lambda^0+\max(0,\ \Delta\lambda_{i_1j_1},\ \Delta\lambda_{i_2j_2},\ \cdots,\ \Delta\lambda_{i_lj_l})\,.\end{aligned}$$

Let U be the set of node pairs which are not assigned yet at that stage. Then, for all elements of U, we can compute $\Delta\lambda_{ij}$ and $\Delta\lambda_{ji}$ by using (5) at each stage and use them as the criterion of selection for new assignment. Especially, $L_{ij}=|\Delta\lambda_{ij}-\Delta\lambda_{ji}|$ is called reversal allowance. For recomputing $\Delta\lambda_{ij}$ and $\Delta\lambda_{ji}$ at each stage, we can use the value of the previous stage under the following conditions.

Proposition 3: Let I_{ij} be the increase of $\Delta\lambda_{ij}$ by introducing $I\succ\succ J$ into S. Then,

1) if $EF_I\leqq ES_J$ and $LF_I\leqq LS_J$,

$$I_{ij}=0 \qquad \text{for all } i,\ j\in N,$$

2) if $EF_I>ES_J$,

$$\begin{aligned}I_{ij}&=(EF_I-ES_J) && \text{for } i=J\\0\leqq I_{ij}&\leqq(EF_I-ES_J) && \text{for } i=\{k|J\succ k\}\\I_{ij}&=0 && \text{for other } i,\end{aligned}$$

3) if $LF_I>LS_J$,

$$\begin{aligned}I_{ij}&=(LF_I-LS_J) && \text{for } j=I\\0\leqq I_{ij}&\leqq(LF_I-LS_J) && \text{for } j=\{k|k\succ I\}\\I_{ij}&=0 && \text{for other } j.\end{aligned}$$

Proof: If $I\succ\succ J$ is introduced into S, I is added to PJ_J newly. If $EF_I\leqq ES_J$, ES_J is kept unchanged by $I\succ\succ J$ by (2). If $EF_I>ES_J$, however, ES_J increases by (EF_I-ES_J) and EF_J, too. From iterative properties of (2), EF_k of the node k for which $J\succ k$ holds may possibly increase with maximum limit of (EF_I-ES_J). Then $\Delta\lambda_{ij}(=EF_i-LS_j)$ increases by the same amount. For the latest schedule time LS_j it is obvious that the similar argument holds.

Proposition 4: If $\Delta\lambda_{IJ}\leqq\Delta\lambda_{JI}$, graph $G=(N,\ A\cup Z\cup(I,J))$ has no loop.

Proof: By Prop. 1 and the assumption of this proposition,

$$EF_I - LS_J \leqq EF_J - LS_I$$

then $$(EF_I - EF_J) + (LS_I - LS_J) \leqq 0.$$

Suppose the occurence of loop in the graph G obtained by introducing $I \succ\succ J$ into S. Then, as the order relation $J \succ I$ holds already in S before the assignment,

$$EF_I - EF_J > 0 \quad \text{and} \quad LS_I - LS_J > 0$$

must hold. This result contradicts the above inequality.

Proposition 5: Let $M = \max_{i\text{-}j \in U} \min(\varDelta\lambda_{ij}, \varDelta\lambda_{ji})$. Then, M is the lower bound of increment $\varDelta\lambda'' = \lambda'' - \lambda^0$, where λ'' is the critical path length of a final S^h which is attainable from the present schedule network $S = (N, A \cup Z)$ by some assignments for all elements of U.

Proof: Let $\varDelta\lambda'$ be the increment of λ for S, in which l resource orders are already assigned. Suppose that g resource orders (i_{l+1}, j_{l+1}), (i_{l+2}, j_{l+2}), $\cdots$, (i_{l+g}, j_{l+g}) are newly assigned for the elements of $U (l+g = |P|)$. By (6) in Prop. 2, λ'', the critical path length of S^h is:

$$\begin{aligned}\lambda'' &= \lambda^0 + \max(0, \varDelta\lambda_{i_1 j_1}, \cdots, \varDelta\lambda_{i_l j_l}, \varDelta\lambda_{i_{l+1} j_{l+1}}, \cdots, \varDelta\lambda_{i_{l+g} j_{l+g}}) \\ &= \lambda^0 + \max(0, \varDelta\lambda', \varDelta\lambda_{i_{l+1} j_{l+1}}, \cdots, \varDelta\lambda_{i_{l+g} j_{l+g}})\end{aligned}$$

According to Prop. 3, $\varDelta\lambda_{ij}$ is unchanged or increasing by assigning a new resource order. Therefore,

$$\begin{aligned}\lambda'' &\geqq \lambda^0 + \max\{0, \varDelta\lambda', \min(\varDelta\lambda'_{i_{l+1} j_{l+1}}, \varDelta\lambda'_{j_{l+1} i_{l+1}}), \cdots, \\ &\qquad \min(\varDelta\lambda'_{i_{l+g} j_{l+g}}, \varDelta\lambda'_{j_{l+g} i_{l+g}})\} \\ &= \lambda^0 + \max(0, \varDelta\lambda', M),\end{aligned}$$

where the dash simbol of $\varDelta\lambda'_{ij}$ represents the present value of the corresponding $\varDelta\lambda_{ij}$.

4. The Algorithm

Starting from $S^0 = (N, A)$, we introduce arc (i, j) into S step by step until all elements of P are assigned. Since either (i, j) or (j, i) are assigned in each node pair i-j, a sequence of the assignment process constructs a binary tree, the nodes of which correspond to node pair i-j $(i, j \in R_k)$ and the arcs of which correspond to two ways of assignment — (i, j) or (j, i). Then our algorithm is reduced to the search procedure of this binary tree.

We use a kind of branch-and-bound method; at each stage of searching the tree $\Delta\lambda_{\max}$ is used as the lower bound of objective. The order of taking up node pair for the assignment affects the amount of computation for the search procedure drastically. Therefore we take two rules of the assignment order:

a) the order of descending reversal allowance L_{ij},

b) the order of descending $\min(\Delta\lambda_{ij}, \Delta\lambda_{ji})$.

The search procedure consists of two phases. At Phase 1, a node pair $i\text{-}j \in U$ is selected in the order of rule a), and introduce $I \succ\succ J$ into S if $\Delta\lambda_{IJ} \leqq \Delta\lambda_{JI}$. Each assignment of resource order corresponds to a node of the tree, and the tree constructed at this phase becomes a straight line and has no branch. Every time a new resource order is introduced into S, $\Delta\lambda_{ij}\,(i\text{-}j \in U)$ are recomputed, if necessary, according to Prop. 3. At Phase 1 no loop occurs from Prop. 4. If all elements of P are assigned, we have a resource order set B^h. This is the first S^h and $\Delta\lambda_{\max}$ at that stage is reserved as the upper bound of $\Delta\lambda$, that is $\Delta\lambda^* = \Delta\lambda_{\max}$.

At Phase 2, the search procedure proceeds by backtracking from the first S^h through the tree obtained by Phase 1. Whenever a node for which $\Delta\lambda_{JI} \leqq \Delta\lambda^*$ is attained, another branch from this node is searched forwardly unless $J \succ\succ I$ introduces loop into S. The forward search procedure in Phase 2 is the same as Phase 1 except node pair $i\text{-}j$ are selected in the order of rule b). This forward search is stopped by two cases:

1) if $\Delta\lambda^* \leqq \max(M, \Delta\lambda_{\max})$, stop the procedure and backtrack from the present node again.

2) if all elements of P are assigned, set $\Delta\lambda^* = \Delta\lambda_{\max}$ and backtrack from that S^h again.

If the procedure comes back to the root of tree, stop all procedure.

The S^h which gave $\Delta\lambda^*$ is an optimal schedule.

The algorithm described above may consist of the following steps.

Step 1 (Initializing step)

$Z=\phi$, $\Delta\lambda^*=\infty$, $\Delta\lambda_{\max}=0$. Compute ES_i, EF_i, LS_i, LF_i $(i \in N)$ for $S^0 =(N, A)$ and $\Delta\lambda_{ij}$, $\Delta\lambda_{ji}$, L_{ij} for all elements of $P(\equiv U)$.

Step 2 (Assignment step)

Select a node pair I-J from U by rule a) (at Phase 1) or rule b) (at Phase 2). If $\Delta\lambda_{IJ} \leqq \Delta\lambda_{JI}$, introduce $I \succ\succ J$ into S.

$$Z=Z\cup(I, J)$$

$$\Delta\lambda_{\max}=\max(\Delta\lambda_{\max}, \Delta\lambda_{IJ})$$

Generate a new node and link it into the tree.

If B^h is completed ($U=\phi$), let $\Delta\lambda^*=\Delta\lambda_{\max}$ and go to step 4.

Step 3 (Recomputing step)

If $EF_I \leqq ES_J$ and $LF_I \leqq LS_J$, go to step 2. Otherwise recompute, if necessary, ES_i, EF_i, LS_i, LF_i, $\Delta\lambda_{ij}$, $\Delta\lambda_{ji}$, L_{ij}, M. If $M \geqq \Delta\lambda^*$, then go to step 4, otherwise go to step 2.

Step 4 (Backtracking step)

(Once passing this step, Phase 2 begins.) Backtrack to the predecessor node in the tree. If there is no predecessor node, stop algorithm. If $\max(\Delta\lambda_{JI}, M) \geqq \Delta\lambda^*$, repeat backtracking, otherwise go to next step.

Step 5 (Reverse assignment step)

Introduce $J \succ\succ I$, if loop do not occur.

$$Z=Z\cup(J, I)$$

$$\Delta\lambda_{\max}=\Delta\lambda_{JI}$$

go to step 3. If loops occur, go to step 4.

Proposition 6: The algorithm consisting of step 1, 2, 3, 4, and 5 above finds a S^h with minimum λ in a finite number of steps.

The proof of this proposition is obvious from the propositions already given and the search procedure of the algorithm.

5. Numerical Illustration

We now illustrate the algorithm by solving a 2-machine, 3-job flow-shop problem presented in Table 1.[2)]

Table 1.

Machine \ Job	1	2	3
A	7	10	9
B	11	5	8

The schedule network of $S^0=(N, A)$ is shown in Fig. 1. Starting from it, we assign resource order one by one according to the proposed algorithm. This step is shown in Table 2.

The situation of schedule network at stage 6 in Phase 1, which is the first S^h and later verified to be an optimal schedule, is shown in Fig. 2. Fig. 3 is the tree representation of the search procedure of Table 2.

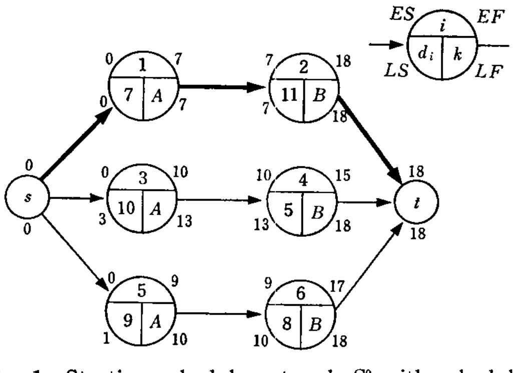

Fig. 1 Starting schedule network S^0 with schedule time.

2) This sample problem was given by E. Balas in reference [1].

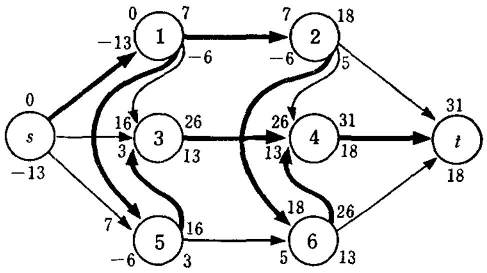

Fig. 2. Schedule network S^h at Stage 6 (optimal schedule).

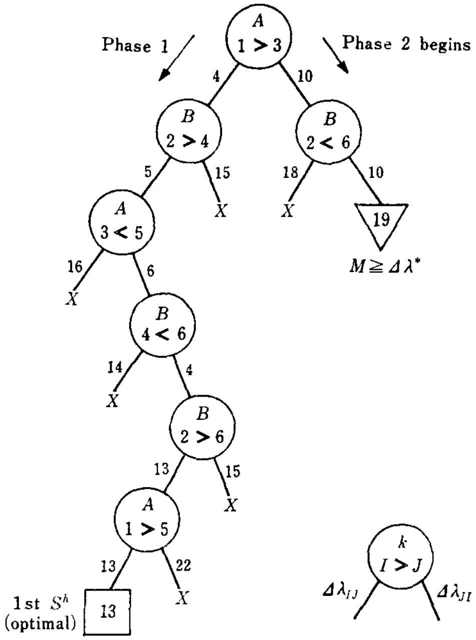

Fig. 3. The search tree of problem 1.

Table 2.

Machine		A			B					
Node-pair		1-3	1-5	3-5	2-4	2-6	4-6	$\Delta\lambda_{\max}$	M	$\Delta\lambda^*$
Phase	Stage									
1	1	④ 10 6	6 9 3	9 6 3	5 8 3	8 10 2	5 4 1	4	8	∞
	2	—	6 13 7	16 6 10	⑤ 15 10	8 10 2	12 4 8	5	8	∞
	3	—	6 14 8	16 ⑥ 10	—	8 15 7	13 4 9	6	8	∞
	4	—	13 14 1	—	—	8 15 7	14 ④ 10	6	13	∞
	5	—	13 14 1	—	—	⑬ 15 2	—	13	13	∞
	6	—	⑬ 22 9	—	—	—	—	13	13	13
2	7	4 ⑩ 6	6 9 3	9 6 3	5 8 3	8 10 2	5 4 1	10	8	13
	8	—	16 9 7	9 19 10	15 8 7	18 ⑩ 8	5 4 1	10	10	13
	9	—	27 9 18	20 19 1	15 8 7	—	16 4 12		19	13

Note: 1. For node-pair i-j, each entry represents $\Delta\lambda_{ij}$, $\Delta\lambda_{ji}$ in the upper side, L_{ij} in the lower side, respectively.
2. The circled figures present $\Delta\lambda_{IJ}$, where (I, J) is the assigned arc at that stage.

6. Conclusion

The search algorithm described above can solve a class of scheduling problems, including job-shop problems. In comparison with several methods already reported, the proposed method have the following features;

1) When the computation of schedule time is required in algorithm, we need not use a graph involving a complete set of resource orders at each stage and therefore we can decrease the amount of computation. Moreover, we need not always recompute schedule time at each stage by using the conditions of Prop. 3.
2) In Phase 1, it is not necessary to try loop-test when a new arc is introduced into S. In Phase 2, loop-test is required only at step 5 and not required while the forward search is going.
3) The algorithm indicates not only the lower bound of the objective at each node but also the order of selecting a node-pair from U. By the rationality of the order of selection, the necessary number of nodes in the search tree becomes smaller. Although we have to consider that the increase of computation time to decide one node at each assignment offsets this effect, we can expect to reduce computation time.

In randomly routed job-shop problem in which the numbers of jobs and machines are as small as hand-computation is possible, the search tree becomes a straight line in most trials. Generally speaking, the algorithm for solving job-shop problem is rather inefficient for flow-shop type problem, because all operations in R_k, especially in R_1, conflict together from starting period. However, in the proposed method the number of nodes in search tree is relatively small. This is mainly due to the effectiveness of the ordering method of selection at Phase 1.

These conclusions are based on the inference from a few cases by hand-computation and their verification by computer experiments will be our next task.

References

[1] Balas, E., "Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm," *O.R.*, 17-6 (1969).

[2] Charlton, J.M. and C.C. Death, "A Generalized Machine-Scheduling Algorithm," *Ope. Res. Quart.*, **21**-1 (1970).

[3] Greenberg, H.H., "A Branch-Bound Solution to the General Scheduling Problem," *O.R.*, **16**-2 (1968).

[4] Levy, F.K., G.L. Thompson and J.D. Wiest, "Introduction to the Critical-Path Method," *Industrial Scheduling*, Chap. 20, Prentice-Hall, 1963.

[5] Yamamoto, M., "Resource Allocation on a Project Network (Part 1)" (in Japanese), *Keiei-Kagaku*, **10**-3 (1967).

[6] Yamamoto, M., "Resource Allocation on a Project Network (Part 2)" (in Japanese), *Keiei-Kagaku*, **11**-2 (1968).