Pergamon

PII: S0305-0548(96)00035-4

Computers Ops Res. Vol. 24, No. 2, pp. 193–198, 1997

0305-0548/97 $17.00+0.00

A FAST FINITE LOADING ALGORITHM FOR JOB ORIENTED SCHEDULING

Chung-Hsing Yeh†‡

Department of Business Systems, Monash University, Clayton, Victoria, 3168, Australia

(Received January 1996; in revised form April 1996)

Scope and Purpose—Industrial production practitioners face problems of detailed day-to-day operations scheduling. In line with practitioners' scheduling experience with the Gantt chart, job oriented scheduling has demonstrated its merits in solving practical job shop scheduling problems. It schedules all the operations of a job to finite capacity before the operations of a lower priority job are considered, whereby constructing a detailed, feasible shop floor schedule. To address the problem of efficient scheduling in a large-scale job shop environment, this paper develops a fast finite loading algorithm for scheduling operations, job by job, forward or backward onto capacity constrained parallel machines. The computational efficiency of the algorithm guarantees its successful implementation in actual industrial settings. The formulation of the algorithm presented in this article provides practical guidelines in the development of computer-based finite scheduling systems for industrial production.

Abstract—Job oriented scheduling (JOS) has been the most commonly used technique in actual job shop settings. It loads jobs one by one onto machines, in which all the operations of a job are scheduled in succession. This article presents a JOS model and formulates a fast finite loading procedure which is the key algorithm of computer-based JOS systems. The algorithm assigns feasible schedule start and finish times to the operations of a job by loading them forward or backward onto the capacity constrained parallel machines. A particular technique is the maintenance of blocks of consecutively scheduled operations on each machine, thereby reducing the search time for finding a feasible time slot for each operation on the corresponding machine. Experimental testing shows that the algorithm has significant computational performance for large size problems. Copyright © 1997 Elsevier Science Ltd

1. INTRODUCTION

Scheduling research has suggested that heuristics provide the only viable approach for job shop scheduling problems of practical size. The mainstream of existing heuristic scheduling research is conducted along the line of operation oriented heuristics [1–4]. Based on a dispatching rule, scheduling decisions at each machine are made by selecting the next operation from a set of operations awaiting processing. In general, operation oriented heuristics are single-pass procedures and are applied in a forward fashion [5], though recent research has proposed an iterative improvement procedure [6] and a backward approach [7]. To generate a detailed production schedule by dispatching rules, a simulation-based approach is usually pursued [8–10]. Simulation-based systems have the capability of dealing with the dynamic nature of the production environment, but some implementation problems such as computer time consumption, human modelling effort and users' acceptance of simulation results need to be resolved [9,10].

Instead of working with individual operations by dispatching rules, another class of heuristics, referred to as *job oriented scheduling* (JOS), schedules one job at a time. In JOS, all the operations of a job are scheduled before the next job is considered.

JOS has many variations, which can be classified by the way that the operation schedule times are determined. Examples include single-pass forward scheduling [11,12], single-pass backward scheduling [13], adjusting forward scheduling [2,14,15] and adjusting backward scheduling [15]. Although very little research on JOS has been reported in the literature, a number of research papers have explored the successful implementation of JOS in industry [11–13,15].

Without using a simulation model, JOS generates a detailed production schedule by loading all the

† email:chyeh@fcit-m1.fcit.monash.edu.au

‡ Chung-Hsing Yeh is a senior lecturer in the Department of Business Systems at Monash University, Australia. He holds a BSc and an MMgmtSc from National Cheng Kung University, Taiwan, and a Ph.D. in information systems from Monash University. His research in the development of manufacturing management software systems has yielded a number of key algorithms which incorporate Just-in-Time concepts to achieve practical production scheduling with low levels of work-in-progress. He has published operations management papers in *European Journal of Operational Research, Production and Inventory Management Journal, and International Journal of Operations and Production Management.* His current research interests include operations management systems and fuzzy logic applications.

operations, job by job, onto machines. This approach has a basic similarity to the manual loading method of the Gantt chart, which is used by most schedulers in actual industrial settings [16]. The classic scheduling book by Conway *et al.*[2] concludes the chapter on the general job shop problem in favour of the Gantt chart approach. The Gantt chart can be used to effectively monitor the production process. It is intelligible and controllable by production personnel. However, the Gantt chart does not provide any structured approach to schedule improvement [17]. In addition, it is too time-consuming and laborious to manually generate Gantt charts and keep them up-to-date for large size problems. A computer-based JOS system can enhance the merits of the Gantt chart approach by overcoming these weaknesses.

In this article, we first outline the general features of a JOS model for the general job shop problem and then present a fast finite loading procedure for computer-based JOS systems.

2. THE JOB ORIENTED SCHEDULING MODEL

2.1. *The problem*

In general, the shop has a number of work centers at which a number of jobs are to be processed. Each work center can have any number of parallel machines whose processing efficiency may vary. Work center capacities can be variably specified at different stages of the planning horizon in terms of the number of parallel machines and the hours of work. The shop may involve manufacturing multi-level items in which assembly job routings are created for making intermediate and end products. Therefore, jobs may have any number of immediate predecessors and successors which constitute job precedence constraints. Each job can have any number of operations which are to be performed at specific work centers in a specified sequence. Non-sequential precedence relationships such as parallel, free start, simultaneous start, and directly follows may exist within a job, and there may be transport time between operations. All job-related attributes and processing times are available when jobs arrive at the shop. Operations are competing for finite production capacity. The scheduling problem is to determine the feasible schedule start and finish times for each operation of each job on the corresponding machines. The scheduling objective for a given job is either to complete the job as early as possible or to complete the job on or close to the due date.

2.2. *The solution procedure*

The scheduling process begins with the determination of a job loading sequence based on some job-related criteria such as management priority, due date, release date, and job number. The job loading sequence needs to be adjusted to meet the job precedence constraints which arise when a set of assembled jobs created for making a multi-level item must be performed in a set sequence.

A feasible schedule is constructed by primarily applying a single-pass procedure through the operations, job by job, in order of the job loading sequence. As an individual job is scheduled, depending on its scheduling objective, a single-pass forward or backward scheduling is applied. *Forward scheduling* schedules all the operations of a job from the schedule start date, commencing with the first operation. The operations are assigned the earliest feasible schedule time on the corresponding machines, consistent with the operation precedence relationships. Forward scheduling aims to complete the job as early as possible. *Backward scheduling* schedules all the operations of a job from its due date, starting from the last operation. The operations are assigned the latest feasible schedule time on the corresponding machines, conforming to the due date requirement and operation precedence constraints. Backward scheduling attempts to complete the job on or close to its due date. An adjusting procedure may be needed for schedule feasibility and/or schedule improvement. The adjusting procedure applies forward and/or backward scheduling to individual jobs and between operations of a job [15].

As described above, the key algorithm of JOS involves loading individual operations forward or backward onto capacity constrained parallel machines. It assigns schedule start and finish times to an operation by searching for the earliest (called *forward loading*) or the latest (called *backward loading*) feasible time slot on the time scale of all the corresponding machines at which the operation can be performed. The time scale of a machine is drawn based on its working time (i.e., its production capacity) over the planning horizon. A time slot on a machine is said to be feasible for an operation to be loaded if (a) it has not been used by previously loaded operations, (b) it is greater than or equal to the operation processing time interval, and (c) it meets all the precedence constraints imposed on the operation. Thus, the computation time of the loading procedure depends on the number of search steps needed to find the feasible time slot. The more the number of previously loaded operations on a machine, the more the

number of search steps required for loading a new operation onto the machine.

To reduce the search time, an efficient algorithm, referred to *as the fast method*, is incorporated into the loading procedure. The fast method regards operations scheduled consecutively on a given machine with no time gaps between them as one *block*. Instead of checking through every previously loaded operation one by one (referred to as *the usual method*), the fast method need examine only each block which consists of previously loaded and consecutive operations. Clearly, the greater the number of operations scheduled consecutively (i.e., in a block), the less the search time needed by the fast method. This fast finite loading algorithm is formulated and computationally examined in subsequent sections.

3. THE ALGORITHM

In a general job shop, I jobs are to be scheduled. Each job i ($i=1, 2,..., I$) has J_i operations and each operation (i, j) ($j=1, 2,..., J_i$) of job i is to be loaded on the earliest or latest of feasible time slots on all L_k parallel machines (k, l) ($l=1, 2,..., L_k$) at its corresponding work center k. The relationship between i, j, and k is specified before scheduling, indicating that operation (i, j) requires processing at work center k. The relationship between i, j, k, and l is assigned by the loading procedure, determining that operation (i, j) is loaded on machine (k, l).

We first formulate the forward loading procedure. The procedure determines the schedule start and finish times of operation (i, j) loaded on machine (k, l) by working out the following time-phased variables step by step.

(1) Potential start time ($pst_{i,j,k,l}$), which is initially assigned based on the constraints imposed on operation (i, j), including (a) the earliest start time of the job if it is the first operation of the job, (b) the schedule finish time of the preceding operation if it is not the first operation of the job, (c) the precedence relationship and transport time considerations with the preceding operation, and (d) the working time of machine (k, l).

(2) Potential finish time ($pft_{i,j,k,l}$), which is calculated by the operation processing time on machine (k, l).

(3) Feasible start and finish times ($fst_{i,j,k,l}$ and $fft_{i,j,k,l}$) on each parallel machine (k, l) ($l=1, 2,..., L_k$), which are given by the feasible time interval $[pst_{i,j,k,l}, pft_{i,j,k,l}]$ that can be fitted in the earliest time slot on the machine.

(4) Schedule finish time ($SFT_{i,j,k,l}$), which is the earliest of all feasible finish times obtained above. The schedule start time ($SST_{i,j,k,l}$) is thus the corresponding feasible start time. This ensures that the operation is scheduled to finish as early as possible under situations where the working time and processing efficiency of parallel machines differ from each other. If two or more machines meet the condition, the lowest numbered one will be loaded.

The computational efficiency of the loading procedure depends on an iterative process for finding the feasible time interval $[pst_{i,j,k,l}, pft_{i,j,k,l}]$. To make the procedure efficient, two measures are taken.

(1) Operations scheduled consecutively on a machine are treated as one block by the use of the following two variables.

$dfo_{k,l}(i,j)$=operation which is scheduled to directly follow operation (i,j) on machine (k, l)

$dpo_{k,l}(i,j)$=operation which is scheduled to directly precede operation (i,j) on machine (k, l)

$$dfo_{k,l}(i,j)=\begin{cases}(g,h), \text{ if } SFT_{i,j,k,l}=SST_{g,h,k,l}\\(0,0), \text{ otherwise}\end{cases}$$

$$\text{If } dfo_{k,l}(i,j)=(g,h), \text{ then } dpo_{k,l}(g,h)=(i,j).$$

These two variables are associated with each operation and their values are given when the operation is scheduled. As a result, $pst_{i,j,k,l}$ can be adjusted by only examining the block as a whole without checking through those infeasible time intervals given by each operation within the block.

(2) If the adjusted pft_{i,j,k,l^*} on the currently considered machine (k, l^*) is later than any of the feasible

schedule finish times assigned on previously considered parallel machines, the loading procedure is terminated. This implies that operation (i,j) will not be loaded on the machine (k, l^*).

The fast finite forward loading algorithm for scheduling operation (i, j) of job i to be carried out at work center k is given below. Initially, $pst_{i,j,k,l}=pft_{i,j,k,l} = fst_{i,j,k,l} = fft_{i,j,k,l} = SST_{i,j,k,l} = SFT_{i,j,k,l} = 0$, and $dfo_{k,l}(i,j) = dpo_{k,l}(i,j)=(0,0)$.

F1. Consider each machine (k, l) $(l=1, 2,..., L_k)$ in numbered order.
F2. Assign initial $pst_{i,j,k,l}$ which meets the operation constraints.
F3. Find $fst_{i,j,k,l}$ and $fft_{i,j,k,l}$ by the following steps:
F3.1. Compute $pft_{i,j,k,l}$ by the operation processing time on machine (k, l).
F3.2. If $l>1$ and $pft_{i,j,k,l} \geq fft_{i,j,k,l^*} \neq 0$ $(l^* \in\{1, 2,...,(l-1)\})$, then return to Step F1 to consider the next machine $(k, l+1)$ until $l=L_k$. Otherwise proceed to Step F3.3.
F3.3. If the current time interval $[pst_{i,j,k,l}, pft_{i,j,k,l}]$ is feasible, i.e.,

$$\left.\begin{array}{l} pst_{i,j,k,l} \geq SFT_{g,h,k,l} \neq 0 \\ or\ pft_{i,j,k,l} \leq SST_{g,h,k,l} \neq 0 \end{array}\right\} \begin{array}{l} \text{for all previously loaded operations} \\ \text{(g,h) on machine } (k,l) \\ (g=1,2,...,i-1; i>1; h \in \{1,2,...,J_g\}) \end{array} \quad (1)$$

then proceed to Step F4.
Otherwise reset $pst_{i,j,k,l}$ and $pft_{i,j,k,l}$ by the following iterative steps until eqn (1) is satisfied.
F3.3.1. Find the operation (x, a) which interferes with the current time interval $[pst_{i,j,k,l}, pft_{i,j,k,l}]$ by: $pst_{i,j,k,l} < SFT_{x,a,k,l} \neq 0$ and $pft_{i,j,k,l} > SST_{x,a,k,l} \neq 0$
where $x \in \{1, 2,..., i - 1\}$; $i>1$; $a \in \{1, 2,..., J_x\}$.
F3.3.2. New $pst_{i,j,k,l}$ is given by:

$$pst_{i,j,k,l}=\begin{cases} SFT_{x,a,k,l}, \text{ if } dfo_{k,l}(x,a)=(0,0) \\ SFT_{y,b,k,l} \text{ otherwise} \end{cases}$$

where $SFT_{y,b,k,l}$ is the schedule finish time of operation (y, b).
Operation (y, b) is determined by the following steps:

s1. Let $(y^*, b^*)=(x, a)$.
s2. If $dfo_{k,l}(y^*,b^*)\neq(0,0)$,
then go to Step s3;
otherwise go to Step s4.
s3. Reset $(x, a)=dfo_{k,l}\neq(y^*, b^*)$ and
$dfo_{k,l}(y^*,b^*)$
return to Step s1.
s4. Set $(y, b)=(y^*, b^*)$.

F3.3.3. Return to Step F3.1.
F4. Let $fst_{i,j,k,l} = pst_{i,j,k,l}$ and $fft_{i,j,k,l} = pft_{i,j,k,l}$.
F5. Continue Step F1 to Step F4 until $l = L_k$.
F6. The schedule start and finish times of operation (i,j) are given by:
$SFT_{i,j} = SFT_{i,j,k,p} = \min \{fft_{i,j,k,l}\}$ for all machines $(k,l)(l=1,2,...,L_k)$; $fft_{i,j,k,l} \neq 0$
$SST_{i,j} = SST_{i,j,k,p} = fst_{i,j,k,p}$
where operation (i,j) is loaded on machine (k, p), $p \in \{1, 2,..., L_k\}$
F7. Set $dfo_{k,p}(g,h) = (i,j)$ and $dpo_{k,p}(i,j) = (g,h)$, if $SST_{i,j,k,p} = SFT_{g,h,k,p}$.
Set $dfo_{k,p}(i,j) = (g,h)$ and $dpo_{k,p}(g,h) = (i,j)$, if $SFT_{i,j,k,p} = SFT_{g,h,k,p}$.

The forward loading logic for determining time-phased variables also applies to backward loading, but is carried out in a reverse manner. For example, in backward loading $pft_{i,j,k,l}$ is initially assigned by (a) the due date of the job if it is the last operation of the job, (b) the schedule start time of the subsequent operation if it is not the last operation of the job, (c) the precedence relationship and transport time considerations with the subsequent operation, and (d) the working time of machine (k, l). $fst_{i,j,k,l}$ and $fft_{i,j,k,l}$ are given by the latest feasible time interval $[pst_{i,j,k,l}, pft_{i,j,k,l}]$ on each machine, in which $pft_{i,j,k,l}$ is adjusted by:

$$pft_{i,j,k,l}=\begin{cases} SST_{x,a,k,l}, \text{ if } dpo_{k,l}(x,a)=(0,0) \\ SST_{y,b,k,l}, \text{ otherwise} \end{cases}$$

where $SST_{y,b,k,l}$ is the schedule start time of operation (y, b). Operation (y, b) is determined by the following steps:

t1. Let $(y^*, b^*)=(x, a)$.
t2. If $dpo_{k,l}(y^*, b^*)\neq(0, 0)$,
then go to Step t3;
otherwise go to Step t4.
t3. Reset $(x, a)=dpo_{k,l}(y^*, b^*)$ and
return to Step t1.
t4. Set $(y, b)=(y^*, b^*)$.

The schedule start and finish times of operation (i,j) by backward loading are given by:

$$SFT_{i,j}=SFT_{i,j,k,p}=\max\{fft_{i,j,k,l}\} \text{ for all machines } (k,l)(l=1,2,...,L_k); fft_{i,j,k,l}\neq 0$$

$$SST_{i,j}=SST_{i,j,k,p}=fst_{i,j,k,p}$$

where operation (i,j) is loaded on machine (k, p), $p \in \{1, 2,..., L_k\}$.

4. COMPUTATIONAL EXPERIENCE

The computational efficiency of finite forward and backward loading by the fast and usual methods has been examined on one actual and 100 randomly generated problems. The scheduling program is coded in FORTRAN and run on a VAX 8700.

The actual scheduling problem, provided by a glass making factory, consists of 856 jobs (5421 operations) scheduled on 17 machines (14 work centers). The 100 randomly generated problems range in size from 50 jobs (256 operations) to 500 jobs (2538 operations) scheduled on 10 machines to 100 machines (10 work centers). The schedule start and finish times of each operation are calculated to the nearest minute and are represented by ordinary calendar dates and times.

It takes 5 min 18 s CPU time for the fast forward loading algorithm to schedule the actual problem (5421 operations). The CPU time required for scheduling 100 testing problems ranges from 2 s to 2 min. 5 s. The computational efficiency of the fast backward loading algorithm is similar to forward loading. The experimental results show that the fast method dramatically reduces the computer processing time required for the same scheduling problem, in comparison with the usual method. For problems of practical size, the fast method may spend only half the time required by the usual method. The larger the job size, the more the computational efficiency of the fast method. Figure 1 shows this.

For a given set of jobs, as the number of parallel machines at a work center increases, the computation time required for the loading procedure increases. However, this increase is not dramatic. For example, the CPU times for loading 500 jobs (2538 operations) on 10 and 100 machines are 1 min 15 s and 2 min 5 s respectively. This is because there is a trade-off between the number of machines to be considered

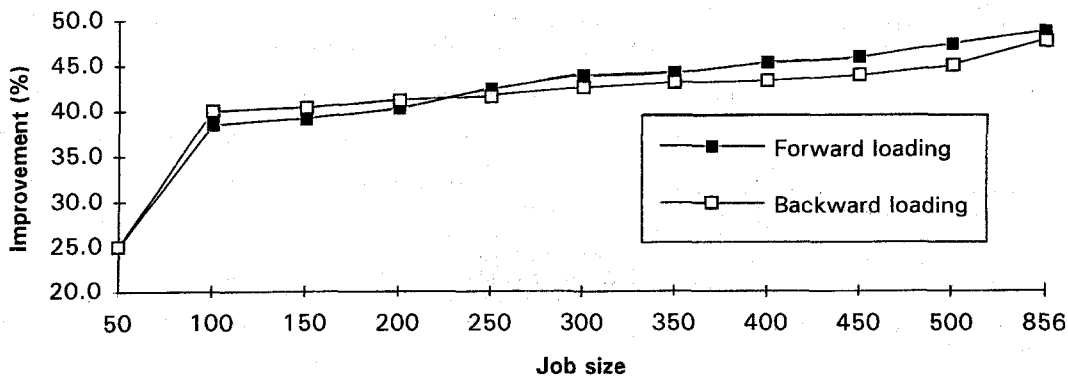

Fig. 1. Computational efficiency improvement of the fast method over the usual method on different job sizes.

for loading an operation and the number of previously loaded operations or blocks to be examined for finding the feasible time slot on each machine. Nevertheless, the increase in the number of parallel machines to be considered cannot be offset completely by the decrease in the number of previously loaded blocks on each machine, resulting in a slight reduction in the efficiency improvement of the fast method over the usual method. For example, the improvement rate decreases from 52% to 39% as the number of parallel machines increases from 10 to 100. Note that the number of parallel machines used for the 500-job problem in Fig. 1 is 20. This indicates that the fewer the number of parallel machines, the more the computational efficiency of the fast method.

When incorporated in a JOS system, the fast finite loading algorithm spends approximately 60–70% of total CPU time required by the whole scheduling process which also includes data editing and validation for all production data concerning scheduling, the determination of the job loading sequence, and the writing of scheduling results onto disk files. For example, it takes approximately 8 min CPU time to complete the whole scheduling process for the actual problem (5421 operations).

The above study demonstrates that the fast finite loading algorithm substantially improves the computational performance of JOS. Although the solution time required increases as the problem size increases, it is still well within practical limits for scheduling job shops of large size. This ensures that the algorithm can be implemented in a real-time scheduling environment where the production controller can experiment with various scheduling parameters for schedule improvement and/or carry out a prompt rescheduling process to keep the schedule up-to-date for all practical purposes.

5. CONCLUSION

The computational complexity of the job shop scheduling problem justifies the development of efficient methods for generating good schedules for industrial production. The problem size and the dynamic nature of the job shop environment make the use of computers necessary. JOS produces good quality of feasible schedules and is intelligible and controllable by industrial production practitioners. A computer-based JOS system can incorporate realistic features of industrial settings and rapidly generate realistic schedules in response to actual production conditions.

The key algorithm of JOS involves loading operations forward or backward to capacity constrained parallel machines. In this paper we have presented a fast finite loading algorithm which substantially reduces the computer processing time of JOS, especially for large size problems. The computational study indicates practical advantages when using this fast algorithm for detailed, day-to-day operations scheduling.

REFERENCES

1. Moore, J. M. and Wilson, R. C., A review of simulation research in job shop scheduling. *Prod. Inventory Mgmt.*, 1967, **8**, 1–10.
2. Conway, R. W., Maxwell, W. L. and Miller, L. W., *Theory of Scheduling*. Addison–Wesley, Reading, MA, 1967.
3. Panwalkar, S. S. and Iskander, W., A survey of scheduling rules. *Ops Res.*, 1977, **25**, 45–61.
4. Blackstone, J. H., Phillips, D. T. and Hogg, G. L., A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *Int. J. Prod. Res.*, 1982, **20**, 27–45.
5. Baker, K. R., *Introduction to Sequencing and Scheduling*. Wiley, New York, 1974.
6. Sun, D., Batta, R. and Lin, L., Effective job shop scheduling through active chain manipulation. *Computers Ops Res.*, 1995, **22**, 159–172.
7. Kim, Y.-D., A backward approach in list scheduling algorithms for multi-machine tardiness problems. *Computers Ops Res.*, 1995, **22**, 307–319.
8. Wyman, F. P., Common features of simulation based scheduling. *Proc.* 1991 *Winter Simulation Conf.*, 1991, pp. 341–347.
9. Rodammer, F. A. and White, K. P., A recent survey of production scheduling. *IEEE Trans. Systems Man Cybernet.*, 1988, **18**, 841–851.
10. Rogers, P. and Flanagan, M. T., On-line simulation for real-time scheduling of manufacturing systems. *Ind. Engng*, 1991, **23**, 37–40.
11. Hastings, N. A. J., Marshall, P. H. and Willis, R. J., Scheduled based M.R.P. : an integrated approach to production scheduling and material requirements planning. *J. Opl. Res. Soc.*, 1982, **33**, 1021–1029.
12. McCarthy, S. W. and Barber, K. D., Medium to short term finite capacity scheduling: a planning methodology for capacity constrained workshops. *Engng Costs Prod. Econ.*, 1990, **19**, 189–199.
13. White, C. and Hastings, N. A. J., Scheduling techniques for medium scale industry. *Aust. Soc. Ops Res. Bull.*, 1983, **3**, 1–4.
14. Magee, J. F. and Boodman, D. M., *Production Planning and Inventory Control*. McGraw–Hill, New York, 1967.
15. Hastings, N. A. J. and Yeh, C.- H., Job oriented production scheduling. *Eur. J. Opl. Res.*, 1990, **47**, 35–48.
16. Hax, A. C. and Candea, D., *Production and Inventory Management*. Prentice–Hall, N.J., 1984.
17. Johnson, L. A. and Montgomery, D. C., *Operations Research in Production Planning, Scheduling, and Inventory Control*. Wiley, New York, 1974.