Journal of the Operations Research Society of Japan

VOLUME 16 *September 1973* NUMBER 3

HEURISTIC ALGORITHMS FOR SCHEDULING n JOBS IN A FLOWSHOP*

JATINDER N. D. GUPTA
U.S. Postal Service
Washington, D.C.

ALBERT R. MAYKUT
U. S. Army Missile Command
Huntsville, Alabama

(Received February 14, 1972)

Abstract

The classical problem of scheduling n jobs on M machines in a flowshop to minimize the throughput time of all jobs is examined under the assumption that all jobs are processed on all machines in the same order. Based on the mathematical formulation of the job and machine slacks, a schedule evaluation algorithm is presented which consists of annotating the process time array. This proposed schedule evaluation algorithm identifies those jobs that are critical to the completion time of any job—thus yielding the critical path(s) and illus-

* This is a modified version of a paper presented at the 39th National Meeting of Operations Research Society of America, Dallas, Texas, May 5-7, 1971.

trates the manner in which job and machine slacks propagate through the flowshop. By defining and exploiting the concept of a synthetic job to represent a partial sequence, the basic ideas of the schedule evaluation algorithm are generalized to the point where they can serve as heuristic approaches to flowshop scheduling problem. One such algorithm, based on Gupta's idle time rule, is described and its performance (regarding its computational efficiency and solution effectiveness) discussed.

1. Introduction

Considerable research effort has been directed towards the solution of the combinatorial optimization problem associated with the scheduling of n jobs M machines in a shop where the flow of jobs to machines is unidirectional (flowshop). This problem was first formulated by Johnson [11] as an n-job 2-machine scheduling problem where the objective function is that of minimizing the total time to complete all jobs (called the makespan). Subsequent research publications have shown that the solution to the general flowshop scheduling problem can be obtained without exhaustive enumeration. The computational requirements associated with the factorial nature of the possible schedules, however, prohibits a practical solution of the moderately large-sized problems [8], [16]. The recent reviews by Bakshi and Arora [1], Day and Hottenstien [3], Elmaghraby [4], and Gupta [6]–[8], discuss several aspects of the flowshop scheduling problem which seem to have been somewhat neglected in the current research efforts to find a practical solution procedure. Because of these difficulties, it appears that the only suitable means of solving the flowshop scheduling problem of any size are the heuristic algorithms, which, while not guarantying optimal solutions, do reliably produce satisfactory solutions with a reasonably small amount of computational effort. The purpose of this paper is to explore one such

area of the development of suitable heuristic algorithms for the solution of the flowshop scheduling problem.

2. Mathematical Analysis of Flowshop Schedules

In order to develop suitable heuristic algorithms, this section describes the behavior of the flowshop schedule in mathematical terms. Here the flow of jobs to machines is viewed as a two directional arrangement of numbers, the functional transformations of which result in a makespan. Thus, considering a given schedule, it is possible to generate sufficient information to provide the background for the development of the heuristic algorithms.

The Gantt chart representation of a flowshop schedule in Fig. 1 illustrates two situations that constantly arise in a flowshop, viz. machine idleness and/or job waiting. In the first situation the second machine is idle for a unit of time during which the second job com-

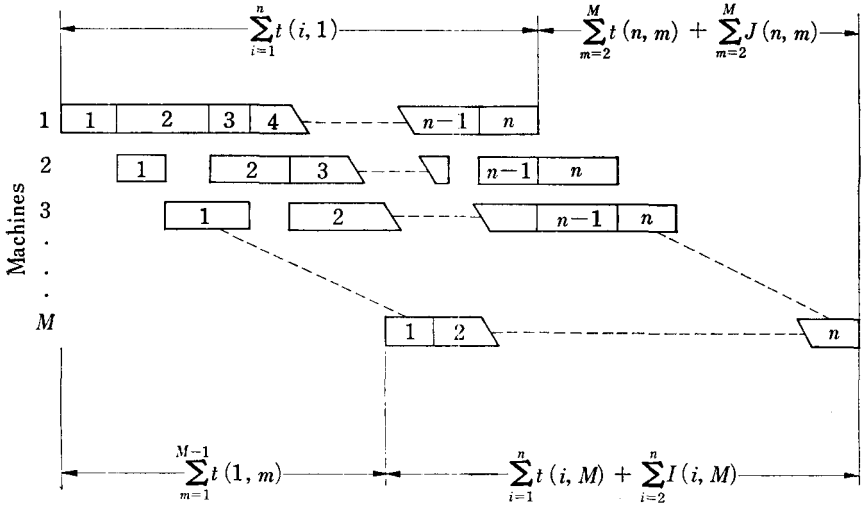


Fig. 1. Gantt chart of the flowshop.

pletes processing on the first machine. This type of idle time will be referred to as *machine slack*. The second situation is typified by the third job which, upon completion of processing on the first machine, must, before resuming processing, wait a unit of time while the second job completes processing on the second machine. This time spent by a job waiting in a queue will be referred to as *job slack*.

The makespan of a given schedule is readily seen to be expressible as the sum of three entities: the summation of processing times of the first job in the sequence on all machines but the last, the summation of processing times of all jobs on the last machine, and the summation of any machine slack occurring on the last machine.

If $t(i, m)$ is the processing time of the ith job on the mth machine[1], then one possible formulation of the total makespan $T(n, M)$ is seen to be

$$T(n, M)=\sum_{m=1}^{M-1} t(1, m)+\sum_{i=1}^{n} t(i, M)+\sum_{i=2}^{n} I(i, M) \tag{1}$$

where $I(i, M)$ denotes the machine slack on the last machine between the $(i-1)$th and ith jobs—*i.e.*, the last machine is idle for this period of time after completing the $(i-1)$th job and before processing the ith job. Some investigators treat the first term in (1) as machine slack also, but in this formulation it will be kept distinct.

For a well defined given schedule, the first two terms of the expression (1) are readily evaluated, since all $t(i, m)$ are given. The last term, the machine slack, requires further analysis.

A second possible formulation of makespan evident in Fig. 1 is as follows:

$$T(n, M)=\sum_{i=1}^{n} t(i, 1)+\sum_{m=2}^{M} t(n, m)+\sum_{m=2}^{M} J(n, m) \tag{2}$$

[1] The index i is used here to denote a job's order in the schedule of jobs, not its identity. The index m denotes both the order of the machine in the flowshop and its identity.

where $J(n, m)$ denotes the job slack experienced by the nth job on the mth machine—*i.e.*, after completing processing on the $(m-1)$th machine, the nth job waits this amount of time before beginning processing on the mth machine.

The similarity of (1) to (2) is quite apparent, and with (1) the last term of (2) needs further development.

Let $t'(i, h)$ be the *effective elapsed time* of the job at ith sequence position on machine h. Then, following the analytical framework of the flowshop scheduling problem (see reference [12] for details):

$$t'(i, h)=t(i, h)+J(i, h), \qquad 2\leqslant h\leqslant M,\ 1\leqslant i\leqslant n \tag{3}$$

where $J(i, h)\geqslant 0$, $\forall\, i$ and h, and

$$J(i, m)=\max\left[\sum_{p=h}^{m-1}[t'(i-1, p+1)-t(i, p)], 0\right]; \qquad 2\leqslant i\leqslant n;\ 2\leqslant h\leqslant m\leqslant M \tag{4}$$

where $h\equiv h(m)$ is the previous machine for which $J(i, h)>0$, otherwise $h=1$.

A similar development for the expression for machine slack will lead to:

$$I(i, m)=\max\left[\sum_{p=k}^{i-1}[t^*(p-1, m-1)-t(p, m)], 0\right]; \qquad 2\leqslant k\leqslant i\leqslant n,\ 2\leqslant m\leqslant M \tag{5}$$

where $k\equiv k(i)$ is the previous job for which $I(k, m)>0$, otherwise $k=1$ and

$$t^*(i, m)=t(i, m)+I(i, m); \qquad 2\leqslant m\leqslant M,\ 1\leqslant i\leqslant n \tag{6}$$

where $I(i, 1)=0$.

The augmented processing time $t^*(i, m)$ will be called the *effective occupied time*, since it represents the span of time within which machine m is either processing or waiting to process job i. This is the analogue to the effective elapsed time for a job, which represents

the span of time within which a job is either processing or waiting to be processed.

The job and machine formulations represent but two ways of formulating makespan, just as Fig. 1 represents but two of the many possible ways of dimensioning a Gantt chart. Any formulation of makespan other than these will contain both job and machine slack terms, however, and thus these two formulations represent extremes.

3. A Schedule Evaluation Algorithm

The above analysis of the behavior of a flowshop schedule may be used to describe a schedule evaluation algorithm. The statement of the algorithm that follows is described in terms of the machine slack formulation of the flowshop, but a completely analogous statement in terms of a job slack formulation is also possible.

1) Array the processing times of the jobs in their sequence order as follows:

Jobs	Machines: 1	2	M
	$t(1,1)$	$t(1,2)\cdots$	$\cdots t(1,M)$
	$t(2,1)$	$t(2,2)$	$\vdots$
	$t(3,1)$		$\vdots$
	$\vdots$		$\vdots$
	$t(n,1)$	$\cdots\cdots\cdots$	$t(n,M)$

2) Consider the columns representing the processing times on the first and second machines. Compare $t(2,1)$ with $t(1,2)$:

a. If $t(2,1)>t(1,2)$, write the difference beside a vertical line drawn between $t(1,2)$ and $t(2,2)$. This difference represents the machine slack of the second machine between the first and second jobs, $I(2,2)$.

b. If $t(2,1)<t(1,2)$, write the difference beside a horizontal line

drawn between $t(2, 1)$ and $t(2, 2)$. This difference represents the job slack of the second job between the first and second machine, $J(2, 2)$.

c. If $t(2, 1)=t(1, 2)$, simply move to Step 3).

3) Next compare $t(3, 1)$ with $t(2, 2)$ in the same manner. If, however, a horizontal line lies to the left of a process time, the indicated amount is to be added to the processing time on the *second* of the two machines. Continue until the end of the column is reached.

4) Next consider the second and third columns. Treat as before, except whenever a vertical line appears above a process time on the *first* of two machines, add the indicated amount to the process time when making the comparison. Continue in this manner until all columns are completed.

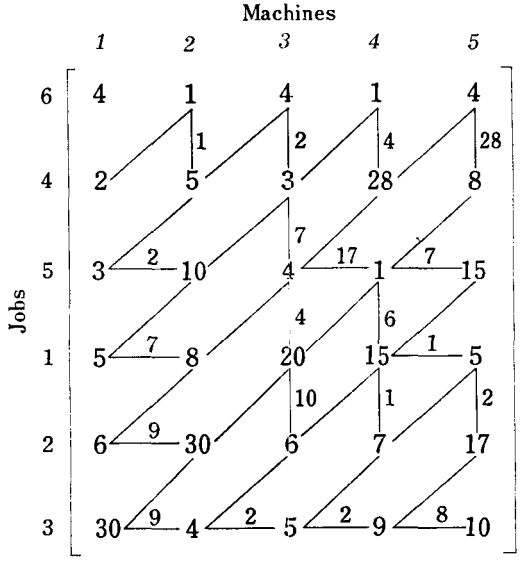


Fig. 2. Illustrative example for the evaluation algorithm.

As an illustration of the above algorithm, consider the evaluation of a schedule 6-4-5-1-2-3 for the 6×5 problem of Fig. 2 [2].

The diagonal lines indicate which processing times are being compared. The amounts written beside horizontal lines are the job slacks between machines, while amounts written beside vertical lines are machine slacks on those machines. The annotated processing time array is seen to contain in numerical form the information identical with that which would be contained in graphical form in the Gantt chart and, as will be shown, much more.

The completion time of any job i on any machine m may be found by simply summing the processing times in any shortest connected path from $t(1, 1)$ to $t(i, m)$, plus all the job and machine slacks encountered in the path. This algorithm, then, solves the flowshop *machine loading* problem. That is, it determines the start and completion times of all jobs on all machines, provided the schedule is known.

Of all the connected paths through the processing time array which yield makespan, there is one which is of more interest than the others. Heller [9] showed that the makespan of a given sequence may be expressed simply as the sum of just $(M+n-1)$ processing times. This implies that there is at least one connected path through the processing time array in which no job and machine slacks are encountered. Such a path is easily found. Starting with the last processing time in the array, $t(n, M)$, a connected path is traced in reverse (leftward or up), always moving to the next processing time in such a way that no job or machine slack is encountered. Such a "move" is always possible, since a job-machine operation cannot have both job slack and machine slack associated with it.

Figure 3 shows this connected path through the processing time array for the 6×5 flowshop problem previously analyzed. The importance of this path lies in the fact that it is the *critical path* through

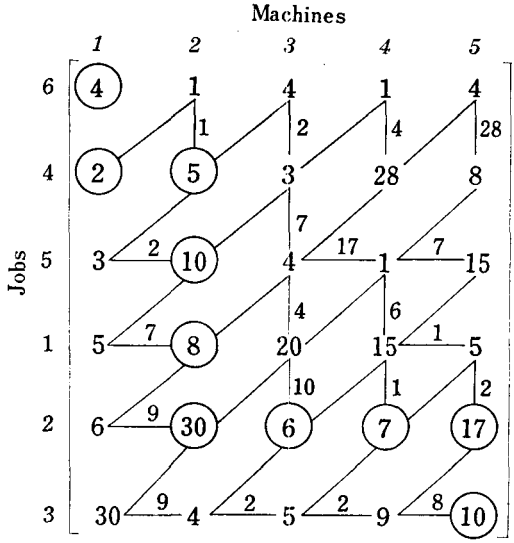


Fig. 3. Critical path for the sample problem.

the array. Any increase in the processing times of the job-machine operations making up this path will increase the makespan by a corresponding amount. Thus the critical path as defined here is analogous to the critical path in PERT and CPM—it represents operations that are critical to the overall schedule. All other processing times not on any critical path are not critical, and any one of these may increase (by at least one unit) without affecting makespan. The same is true, of course, for the completion time of any job-machine operation. Each has at least one critical path: a series of preceding operations that are critical to its completion time.

The above schedule evaluation algorithm, while useful in analyzing schedules, does not solve the flowshop scheduling problem. It does, however, furnish insight into schedule behavior and will serve as the basis for developing scheduling algorithms.

4. A Heuristic Job-Pairing Algorithm

The most elementary unit of a flowshop schedule that retains all the properties of a schedule is the job pair—two jobs processed sequentially. This is the smallest unit that can generate job slack and machine slack, and have a makespan that is job-order dependent. It is therefore logical to use the job pair as the building block when attempting to construct purposeful flowshop schedules.

The empirical results indicate that the selection of a low processing time job for the first position contributes to the creation of machine slack on the last machine at later schedule positions [12]. For problems of large size, this added machine slack generally more than offsets the initial advantage of the starting job. A selection criterion for the starting job pair based solely on minimum machine slack on the last machine is thus proposed for problems having more than twelve jobs. While the term

$$\sum_{m=1}^{M-1} t(1, m)+I(2, M)$$

is used for a problem containing fewer than twelve jobs.

No selection criterion is complete without a rule for resolving ties. This is especially true with regard to the minimum machine slack on the last machine, since many jobs will, in general, fill a given schedule position without creating machine slack at this machine. A tie-breaking rule thus affords an opportunity to further improve the characteristics of the schedule under construction. Such a tie-breaking criterion is suggested by the analysis provided by the schedule evaluation algorithm. Since the job slack producing potential of a job is expressed by its effective elapsed times, the sum of appropriately weighted effective elapsed times appears to be an ideal tie-breaking rule and is used in the proposed heuristic algorithm.

Before giving a formal statement of a job-pairing algorithm, a

concept will be introduced that proves useful in the execution of the algorithm. If the effective elapsed times of a particular job are arrayed as processing times, then this *synthetic job* will have schedule characteristics for the schedule of jobs following it identical with the partial schedule ending in this particular job. Thus any partial schedule may be represented as a single job for constructing the remainder of the schedule.

A Job-Pairing Algorithm

The rules just described may be used to form a heuristic algorithm for building a near optimum sequence. The step-by-step procedure of the algorithm follows.

Step 1. From all possible job pairs among the n jobs, which results in $n(n-1)$ job pairs. (a) If n is twelve or fewer, select that pair to begin the schedule that has the minimum sum of the first job on the last machine and machine slack on the last machine. In other words, select the job pair having the minimum sum of

$$\sum_{m=1}^{M-1} t(1, m)+I(2, M) .$$

(b) If n is greater than 12, select the pair to begin the schedule that has the minimum machine slack on the last machine, $I(2, M)$.

Step 2. If the application of the above criteria results in a tie, calculate the effective processing time of the second job of each tied pair. Break the tie by choosing the pair having a second job with the largest machine-position-weighted effective processing time sum[2]. That is, choose the pair having the maximum

[2] If this method fails to break the tie, a secondary rule, such as a higher power weighting function, may be used.

$$\sum_{m=1}^{M} (m-1)t'(2, m) .$$

Step 3. Form the synthetic job to represent the job pair just chosen.

Step 4. With this synthetic job always in the first position, form job pairs with all the remaining jobs. Select that job pair having the minimum machine slack on the last machine. Break ties as in Step 2.

Step 5. Repeat Steps 3 and 4 until all but two jobs are allocated to sequence positions.

Step 6. Form two complete sequences by allotting the remaining two jobs to both of the last sequence positions. Select, as the result, the sequence having the minimum makespan.

It is readily seen that Step 6 is necessary, since it insures that the machine slack developed by the final job pair is considered.

An Example

As an illustration of this algorithm, consider the following 6×3 flowshop sequencing problem.

i \ m	1	2	3
1	1	8	4
2	8	5	10
3	11	2	7
4	3	9	2
5	5	5	4
6	12	7	7

Step 1. Form all thirty possible job pairs. Since n is fewer than twelve, determine the sum

$$\sum_{m=1}^{2} t(1, m)+I(2, 3) .$$

For example, job pair (1, 2) would yield

$$\begin{bmatrix} 1 & 8 & 4 \\ & \diagup & \diagup \;|\; 1 \\ 8 & 5 & 10 \end{bmatrix}$$

and

$$\sum_{m=1}^{2} t(1, m)+I(2, 3)=(1+8)+1=10 .$$

A table of these sums for all job pairs (i, k) appears below.

			k			
i	1	2	3	4	5	6
1		10	10	14	10	16
2	13		13	13	13	17
3	14	17		16	14	23
4	18	15	14		16	20
5	14	14	14	15		20
6	20	19	19	21	19	

The job pairs (1, 2), (1, 3) and (1, 5) are seen to have identical sums of ten units.

Step 2. To break the ties that have occurred, determine the synthetic job to represent each pair (*i.e.*, determine the effective processing times of the second job). For (1, 2)

$$\begin{matrix} 1 \\ 2 \end{matrix}\begin{bmatrix} 1 & 8 & 4 \\ & \diagup & \diagup \;|\; 1 \\ 8 & 5 & 10 \end{bmatrix} .$$

The result is the synthetic job

$$1\text{–}2\ [8\ \ 5\ \ 10] .$$

Similarly for the job pair (1, 3) and (1, 5), the synthetic jobs are

$$1\text{–}3\ [11\ \ 2\ \ 7]$$

and

1–5 [5 8 4] .

Calculating the weighted effective elapsed times for job pair (1, 2) result in

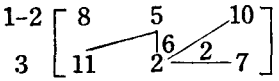
$$\sum_{m=1}^{3}(m-1)t'(2, m)=(1)5+(2)10=25 .$$

Similarly, the resulting weighted effective elapsed times are 16 for both job pairs (1, 3) and (1, 5). Thus job pair (1, 2) is selected to begin the sequence.

Step 3. The synthetic job representing the job pair just chosen has already been formed as

1–2 [8 5 10] .

Step 4. Next, form all possible pairs with this synthetic job and the remaining jobs. Evaluate these pairs to determine the machine slack on the last machine. For example, the job pair (1–2, 3) yields



This and the remaining pairs are tabulated below:

Pair	$I(3, 3)$
(1–2, 3)	0
(1–2, 4)	0
(1–2, 5)	0
(1–2, 6)	4

Again, ties are broken by calculating the weighted effective processing times. For the job pair (1–2, 3), for example:

$$\sum_{m=1}^{3}(m-1)t'(3, m)=(1)2+(2)9=20 .$$

The results for this and the remaining tied pairs are

Pair	Weighted effective processing time sum
(1-2, 3)	20
(1-2, 4)	17
(1-2, 5)	23

Job 5 is therefore selected to fill the third sequence position.

Step 5. Forming the synthetic job for the job pair (1-2, 5) yields

1-2-5 [5 5 9].

Forming job pairs with this synthetic job and all remaining unallocated jobs leads to the following machine slacks on the last machine and weighted effective processing time sums:

Pair	$I(4, 3)$	Weighted effective processing time sum
(1-2-5, 3)	0	18
(1-2-5, 4)	0	13
(1-2-5, 6)	5	—

and Job 3 is selected to fill the fourth sequence position.

Step 6. The last two sequence positions are determined by evaluating the makespan of the two sequences formed by having the two remaining unallocated jobs in both final sequence positions. These two sequences are 1-2-5-3-4-6 and 1-2-5-3-6-4 which result in makespans of fifty-four and fifty-five units respectively. The sequence 1-2-5-2-4-6 is therefore accepted as the result.

As a matter of interest, an optimal solution to this problem as given by the branch and bound technique results in the minimum makespan of forty-nine units. Had the min-idle heuristic of Gupta [5] been used, a sequence having a makespan of fifty-five units would have resulted.

5. Experimental Investigations

An evaluation of the job-pairing algorithm was conducted which consisted of the solution of a large number of randomly generated problems. The processing times were taken from a uniform distribution having a range of 0 to 99 inclusive. The size of these problems is sufficiently small to allow a solution for the minimum makespan of each problem. With the minimum makespan as the basis, the percentage error, ϵ, of a corresponding heuristic solution is defined to be

$$\epsilon = \frac{T_H - T_0}{T_0} \times 100$$

where T_H denotes the makespan obtained by a heuristic solution, and T_0 denotes the minimum makespan.

Table 1 summarizes the results of heuristic solutions obtained with both the job-pairing algorithm and for Gupta's min-idle rule [5]. This latter algorithm was selected for comparative purposes since it also uses job pairs in building schedules and in addition has other similarities to the job-pairing algorithm.

Over-all, the job-pairing algorithm out performs the min-idle rule, having a composite error of 10.56% as compared to 10.94% for that of the min-idle rule. Also, the number of optimum schedules is higher for the majority of the problem sets, as the total number of lower makespan schedules. The only statistic in which the job-pairing algorithm does not excel is that of maximum error. The solution quality of both algorithms is seen to be reasonably high.

A further evaluation was conducted with problems having a large number of jobs and machines. The size of these problems prohibits a minimum makespan solution and the performance was placed on a relative basis. The relative effectiveness of the two heuristics may

Table 1. Performance comparison of job-pairing algorithm and min-idle rule for small problems.

Size ($n \times M$)	Number of problems	Job-pairing algorithm			Min-idle rule			Number job-pair best[2]	Number min-idle best[2]
		Number optimum[1]	Maximum ϵ	Average ϵ	Number optimum[1]	Maximum ϵ	Average ϵ		
5×3	40	8	31.06	7.42	2	39.81	11.36	17	14
6×3	40	16	37.73	5.77	7	33.57	10.24	24	6
7×3	40	7	31.67	9.38	3	33.73	10.94	19	13
8×3	40	11	45.05	7.36	6	41.74	9.68	19	13
10×3	40	7	28.38	7.53	3	24.79	8.65	20	10
12×3	40	10	24.26	6.29	6	21.96	8.31	19	13
5×4	40	8	35.77	10.46	7	35.77	10.02	16	16
6×4	40	5	42.99	11.13	2	36.15	11.63	20	13
7×4	40	3	49.12	14.92	3	34.14	12.60	15	23
5×5	40	5	35.46	9.47	5	35.46	9.35	13	18
6×5	40	2	32.18	12.43	1	29.50	12.59	20	16
7×5	40	4	31.29	11.43	2	28.49	11.06	15	21
5×6	40	3	31.65	8.84	1	44.88	9.84	18	16
6×6	40	0	28.11	12.44	2	30.52	9.83	11	19
7×6	40	1	51.49	15.60	0	35.06	13.08	13	20
5×7	40	3	36.20	11.14	2	32.32	11.19	22	13
6×7	40	0	39.82	14.28	1	31.14	12.12	16	19
7×7	40	0	42.22	14.14	0	29.26	14.47	18	15

1. Number of problems solved optimally by the heuristic.
2. Number of problems for which superior sequences were produced by this algorithm.

Table 2. Relative performance of job-pairing algorithm with respect to the min-idle rule for large problems.

Size ($n \times M$)	Number of problems	Relative effectiveness		Number mod. job-pairing best	Number min-idle best
		Range of r	Mean r		
10×10	5	0.9825-1.0915	1.0130	2	1
10×20	5	0.9390-1.0092	0.9741	4	1
20×20	5	0.9379-1.0022	0.9728	3	1
20×40	5	0.9681-1.0146	0.9949	2	3
20×60	5	0.9711-1.0617	1.0051	3	1
40×20	5	0.9251-0.9777	0.9525	5	0
40×40	5	0.9765-1.0256	0.9988	3	2
40×60	5	0.9813-1.0190	0.9957	3	2
60×20	5	0.9026-1.0106	0.9575	4	1
60×40	5	0.9908-1.0069	1.0011	1	4
60×60	5	0.9663-1.0362	1.0104	1	4

be measured by the ratio $r = T_{jp}/T_{mi}$, where T_{jp} and T_{mi} are the make-spans obtained with the job-pairing algorithm and min-idle rule respectively.

Table 2 summarizes the relative performance of the two algorithms for these large problem sets. While the relative performance of the two algorithms is comparable, the job-pairing algorithm out performs the min-idle rule in terms of the number of superior schedules produced and average relative efficiency.

6. Conclusions

The basic objective of this research was to examine an $n \times M$ flowshop. A mathematical formulation of a flowshop provides a means for analyzing the behavior of jobs as they are being processed. The creation of job and machine slack as an expression of this behavior leads to an algorithm that provides a thorough description of a flow-

shop. This schedule evaluation algorithm consists of annotating the processing time array and produces an arithmetic form of the Gantt chart.

This algorithm is useful at two different levels. At the applied level it solves the machine loading problem. That is, it determines the start, idle, and completion time of all jobs at all machines. A simple extension identifies those jobs that are critical to the completion time of any other job—thus identifying the critical paths.

At a more basic level, the schedule evaluation algorithm provides insight into schedule behavior. Job and machine slack are found to propagate through the flowshop in an orthogonal manner. This characteristic may be exploited if job slack is used to offset machine slack, and leads to the creation of heuristic scheduling rules which, together with the invention of the synthetic job, are incorporated into a job-pairing algorithm for scheduling jobs in a flowshop. This algorithm is found to lead to high quality solutions. In most cases, solutions to both small- and large-sized problems are superior to an existing job-pairing heuristic, the min-idle rule of Gupta [5].

References

[1] Bakshi, M. S. and S. R. Arora, "The Sequencing Problem," *Management Science*, **16** (1969), 247-263.

[2] Brown, A. P. G. and Z. A. Lommicki, "Some Applications of the 'Branch-and-Bound' Algorithm to the Machine Scheduling Problem," *Operational Research Quarterly*, **17** (1966), 173-186.

[3] Day, J. E. and M. P. Hottenstien, "Review of Sequencing Research," *Naval Research Logistics Quarterly*, **17** (1970), 11-39.

[4] Emlaghraby, S. E., "The Machine Sequencing Problem—Review and Extensions," *Naval Research Logistics Quarterly*, **15** (1968), 205-232.

[5] Gupta, J. N. D., "Heuristic Rules for $n \times M$ Flowshop Scheduling Problem," *Opsearch*, **5** (1968), 165-170.

[6] Gupta, J. N. D., "A General Algorithm for the $n \times M$ Flowshop Scheduling Problem," *The International Journal of Production Research*, **7** (1969), 241-247.

[7] Gupta, J. N. D., "Economic Aspects of Production Scheduling Systems,"

Journal of Operations Research Society of Japan, **13** (1971), 11-35.

[8] Gupta, J. N. D., "*M*-Stage Scheduling Problem—A Critical Appraisal," *The International Journal of Production Research*, **9** (1971), 267-281.

[9] Heller, J., *Combinatorial, Probabilistic and Statistical Aspects of an $M \times J$ Scheduling Problem*, NYO-2540, Feb. 1, 1959. New York: AEC Computing and Applied Mathematics Center, Institute of Mathematical Sciences, New York University, 1959.

[10] Ignall, Edward and Linus Schrage, "Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems," *Operations Research*, **13** (1965), 400-412.

[11] Johnson, S. M., "Optimal Two- and Three-stage Production Schedules with Setup Times Included, *Naval Research Logistics Quarterly*, **1** (1954), 61-68.

[12] Maykut, A. R., "A Heuristic Approach to Flowshop Sequencing," Master's Thesis, University of Alabama in Huntsville, Huntsville, Alabama, 1971.

[13] Palmer, D. S., "Sequencing Jobs through a Multi-Stage Process in the Minimum Total Time—A Quick Method of Obtaining a Near Optimum," *Operational Research Quarterly*, **16** (1965), 101-107.

[14] Roy, B., "Cheminement et connexite dans les graphes-applications aux problemes d'ordonnacement," *Metra*, Serie Speciale No. 1, Paris: Societe d'economie et de mathematique appliquees, 1962.

[15] Smith, Richard D. and Richard A. Dudek, "A General Algorithm for the Solution of the n-Job, M-Machine Sequencing Problem of the Flow Shop, *Operations Research*, **15** (1967), 71-82.

[16] Smith, M. L., "A Critical Analysis of Fowshop Sequencing," Ph.D. Dissertion, Texas Technological College, Lubbock, Texas, 1968.