Annals of Operations Research 50(1994)263–279

An efficient heuristic based on machine workload for the flowshop scheduling problem with setup and removal

Wenxue Han and Pierre Dejax

Laboratoire Productique et Logistique, Ecole Centrale Paris, Grande Voie des Vignes, 92295 Châtenay-Malabry Cedex, France

We are concerned in this paper with solving an n jobs, M machines flowshop scheduling problem where the objective function is the minimization of the makespan. We take into account setup, processing and removal times separately. After drawing up a synthesis of existing work which addresses this type of problems, we propose a new heuristic algorithm which is based on the machine workload to find an efficient permutation schedule. Computational experiences show that our algorithm yields excellent results, particularly when bottleneck machines are present.

1. Introduction

Consider a static flowshop scheduling problem where n given jobs are to be processed on M specified machines in the same technological order. The objective function is the minimization of the makespan (also called the maximum flowtime) defined as the total throughput time in which all jobs complete processing on all machines. This problem was first formulated and solved by Johnson [20] as a two-machine problem. In formulating this problem, several simplifying assumptions are made by Gupta [15]. Specifically, it is assumed that changeover times (where changeover time is defined as the time required to change from one job to another at a given machine) are sequence independent and are either included in the processing times or are negligible and can be ignored. However, in some practical situations, changeover times of a job are separable and may depend on the immediately preceding job. As an illustration, consider the scheduling problem in a group technology environment. For each family of parts, a long setup time is required to initiate family parts, after which a short changeover time is required to initiate family parts, after which a short changeover time is required which may depend on the sequence of jobs preceding a particular family part (job) being processed [17]. In such cases, one has to consider these changeover times explicitly in the identification of an optimal schedule.

When the changeover times are separable and sequence dependent, and infinite storage space is supposed to be available to hold partially processed jobs, mathematical

programming approaches and implicit enumeration approaches based on graph-theoretical representation can be used to solve the problem [14,26]. Furthermore, if only one of the two machines involves sequence-dependent separable changeover times, a dynamic programming approach can be used to minimize the makespan [8]. However, the computational complexity of these exact optimization algorithms is quite large even for problems of moderate size. In fact, flowshop problems with sequence-dependent changeover times are NP-hard, implying that it is unlikely that an efficient optimization algorithm that will solve all problems in a polynomially bounded computational time can ever be found. Realizing this fact, Gupta and Darrow [17] described heuristic algorithms to find approximate solutions to the two-machine sequence-dependent flowshop problem. An analysis of the general M-machine sequence-dependent flowshop problem is provided by Gupta [16] who also describes some heuristic approaches based on the travelling salesman formulation of the problem.

In many practical cases, changeover times are separable and can be broken down into sequence-independent components. In doing so, the changeover time is defined as the sum of two components, namely the actual setup time (S_{im}) needed to perform the setup for job i at machine m and the actual removal time (R_{km}) for job k at machine m so that machine m can process job i. In such a case, the setup time is the time needed to set up the tools, jigs, fixtures, etc., for job i at machine m while removal time is the time needed to remove the tools, jigs, and fixtures or to undertake cleaning operations once job k has completed processing on machine m. While the changeover times can now be considered independent of the sequence, the resulting problem is still NP-hard. Therefore, in addition to exact optimization algorithms, heuristic algorithms are both desired and necessary. The approaches taken by Yoshida and Hitomi [31], Sule [27], Sule and Huang [28], Szwarc [29], and Proust et al. [23,24] can be used to develop approximately optimal solutions to the problem.

In this paper we consider the flowshop problem where the setup, processing, and removal times are separable and significant enough to be considered explicitly in finding an optimal schedule. The objective function considered here is the minimization of the makespan (or the maximum flowtime of any job). In section 2, we define precisely the notations and assumptions of the problem. In section 3 we survey the corresponding literature. In section 4, we present a new heuristic procedure (PHD_{SR}) based on the explicit consideration of the machine workload. Section 5 reports the results of our computational experience. In particular we compare the performance of our heuristic with one of the most recent and efficient heuristics (DFFP) for this problem, as well as to the optimal solutions obtained from a newly developed branch and bound procedure. Section 6 presents our conclusions.

2. Problem, notations, assumptions and definition

Following Graham et al. [13], a shop scheduling problem may be characterized by three parameters, namely the machine environment, job characteristics and the

optimality criteria. Using the symbol F for the flowshop, M for the number of machines and C_{max} for the maximum flowtime, a flowshop problem may be represented as an $FM||C_{max}$ problem. Furthermore, with the separation of no sequence-dependent (nsd) setup and removal times from the processing time p_{ij} needed to process job i at machine j, the above representation is modified as a $FM|S_{nsd}, R_{nsd}|C_{max}$ problem. Furthermore, in formulating the problem, the following assumptions are made.

(1) All jobs and machines are available at time zero.

(2) All processing times on the machines are known, finite, and independent of the order in which all the jobs are processed. Setup, processing and removal times are separable and independent of the order in which jobs are processed on any of the M machines.

(3) Each job is processed through each of the M machines once and only once. Furthermore, a job does not become available to the next machine until it completes processing on the current machine. Splitting of jobs is not allowed.

(4) Once a job starts on a machine, it must be completed before another job can be scheduled on the same machine (i.e. no pre-emption of jobs is allowed).

(5) Sufficient storage capacity exists to stock any number of available jobs between any two consecutive machines.

(6) Only permutation schedules (i.e. schedules which maintain the same order of all available jobs on all machines) are considered.

While the above assumptions may seem very restrictive, the associated problem has been shown to be NP-hard by Garey and Johnson [12] even when setup and removal times are not considered. Realizing this fact, several heuristic algorithms have been developed to find efficient solutions to the $FM||C_{max}$ problem. Notable among these are the approaches taken by Campbell et al. [4], Dannenbring [9], Park et al. [22], and Escudero [10]. For a recent state of the art on this subject, consult Han and Dejax [18]. For the general job-shop scheduling problem, Adams et al. [1] have proposed an approximation method based on the identification of bottlenecks and for minimizing the makespan. It consists of repeatedly identifying and sequencing the bottleneck machines as well as reoptimizing the previously established sequences. Both the bottleneck identification and local reoptimization procedures are based on solving certain one-machine scheduling problems. Besides the straight version of the Shifting Bottleneck Procedure, they have also implemented a version that applies the procedure to the nodes of a partial search tree. It should be noted that Carlier and Pinson [5,6] have also proposed a bottleneck approach for the job-shop problem. But unfortunately, tools setup and removal times were not taken into account in these works. For other research on the subject of job-shop scheduling, see also Balas [3], Baker [2], Conway et al. [7], French [11], and Rinnooy Kan [26].

Let us now define the completion time for each job–machine pair (i,m). For that purpose, let p_{im}, S_{im}, and R_{im} be the processing, setup, and removal times, respectively. We suppose that σ is a schedule of the jobs already scheduled before job i. Consider now a schedule σi formed by concatenating job i at the end of σ. Then, the completion time of σi at machine m can be found by using the following recursive relationship:

$$C(\sigma i, m) = \max\{C(\sigma i, m-1) - R_{i,m-1}; C(\sigma, m) + S_{im}\} + p_{im} + R_{im}, \quad (1)$$

where $C(\varnothing, m) = C(\sigma, 0) = 0$ for all σ and m.

To calculate $C(\sigma i, m)$, $C(\varnothing, m) = 0$ is useful if σ is empty (i is the first job to be scheduled); and $C(\sigma, 0) = 0$ is useful if m is the first machine to be considered.

In defining the above relationship (1), it is assumed that the setup for job i at machine m starts as soon as the removal operation of the last job of σ has been completed. Furthermore, job i can be moved from machine $(m-1)$ to machine m as soon as it completes processing at machine $(m-1)$ and does not wait at machine $(m-1)$ for the removal of its tools, jigs, fixtures, etc. Then, the permutation flowshop scheduling problem consists of finding a schedule σi such that $F[C(\sigma i, M)]$ is minimum where σ ranges over all permutations of $(n-1)$ jobs not containing job i and i ranges from 1 through n not contained in the specific σ associated with σi.

As previously stated, the goal of the approach is to describe an approximate algorithm to obtain the job's inputting sequence for the machines system, and the optimality criterion used in this paper is the minimization of makespan (or maximum flowtime). However, Sule and Huang [28], Proust et al. [24] and Szwarc and Gupta [30] differ in their interpretation of makespan. Sule and Huang [28] and Proust et al. [24] consider $C(S,M)$ as defined by relation (1) above as makespan if S is a complete schedule of n jobs. However, this includes removal time associated with the last job in S at machine M. Szwarc and Gupta [30] think that this removal time should not be included in the definition of makespan. In this paper, Sule and Huang's interpretation will be used and makespan will be as obtained by using relation (1).

3. Heuristic algorithms for the $FM|S_{\text{nsd}},R_{\text{nsd}}|C_{\max}$ problem

The problem considered in this paper was first formulated by Yoshida and Hitomi [31], who developed a simple extension of Johnson's optimal rule [20] to solve the two-machine case where removal and setup times are not separated from each other and are sequence-independent. Sule [27] modified Yoshida and Hitomi's results to allow for the separation of setup, processing, and removal times. Sule and Huang [28] extended the two-machine analysis to solve approximately the three-machine case. Szwarc [29] was interested in more general scheduling problems than the one we are studying here, but it can be noted that he also suggested, as a special case, a solution to our problem. Proust et al. [24] draw up a synthesis of

Campbell et al.'s [4] CDS heuristic algorithm and Sule's algorithm mentioned above and propose a heuristic algorithm, called DFFP for the general M-stage problem. This heuristic has been considered to be the best up to now. An improved version, HEURES [23], which consists in making adjacent permutations on the schedule selected with DFFP was proposed. The following is a summary of the work that has come to our attention.

3.1. THE SH AND SH-1 ALGORITHMS

In a variation on the classical problem, Yoshida and Hitomi [31] allowed for a setup time before each operation of each job on each of the machines. For the two-machine case and with allowance for setup, processing and removal times, Sule [27] showed that job i precedes job j in an optimal schedule if:

$$\min(S_{i1} - S_{i2} + p_{i1}, p_{j2} + R_{j2} - R_{j1}) \leq \min(S_{j1} - S_{j2} + p_{j1}, p_{i2} + R_{i2} - R_{i1}). \quad (2)$$

The above relation (2) leads to the following algorithm:

Step 1: Let

$$k = \arg\min_{i \in K} \{\min\{S_{i1} - S_{i2} + p_{i1}, p_{i2} + R_{i2} - R_{i1}\}\},$$

where K is the set of jobs still available. Initially all n jobs are concerned.

Step 2: In case of $S_{k1} - S_{k2} + p_{k1} \leq p_{k2} + R_{k2} - R_{k1}$, place job k at the start of the schedule; otherwise, place job k at the end of the schedule.

Step 3: Suppress job k from the set K of remaining jobs to be placed and go to step 1.

The above algorithm is a polynomial procedure (henceforth called the SH algorithm) to find the optimal solution of the $F2|S_{\text{nsd}}, R_{\text{nsd}}|C_{\max}$ problem.

Sule and Huang [28] proposed a heuristic algorithm for the three-machine problem. This heuristic consists in extending the above two-machine algorithm to approximately solve the three-machine case by creating an auxiliary two-machine problem where the first stage is a combination of the first two machines and the second stage is a combination of the last two machines. In other words, two real machines are combined into one auxiliary machine and the corresponding times are added together. The steps of the algorithm are as follows:

Step 1: Let

$$k = \arg\min_{i \in K} \{\min\{S_{i1} - S_{i3} + p_{i1} + p_{i2}, p_{i2} + p_{i3} + R_{i3} - R_{i1}\}\},$$

where K is the set of jobs still available. Initially all n jobs are concerned.

Step 2: In case of $S_{k1} - S_{k3} + p_{k1} + p_{k2} \leq p_{k2} + p_{k3} + R_{k3} - R_{k1}$, place job k at the start of the schedule; otherwise, place job k at the end of the schedule.

Step 3: Suppress job k from the set K of remaining jobs to be placed and go to step 1.

The above algorithm (henceforth called the SH-1 algorithm) generates an approximate solution to the original $F3|S_{nsd}, R_{nsd}|C_{max}$ problem. This heuristic was tested at first on 200 problems with five or six jobs. The optimal sequence was obtained in 93% of the trial runs. In the other cases, the makespan of the proposed solution was more than 2% off the optimal makespan.

3.2. SZWARC ALGORITHM

A generalization of the problems proposed in Yoshida and Hitomi [31], Sule [27], Sule and Huang [28], and Proust et al. [24] can be defined by considering the $FM|\text{lag time}|C_{max}$ problem. Szwarc [29] studied this problem and proposed heuristic approaches which can be modified to solve the problem being considered here if we consider the lag time as no sequence-dependent setup and removal times. Based on Szwarc's work, we propose the following rough algorithm:

Step 0: Initialize and transform the data. For each job i and each machine m, calculate:

$$\begin{aligned} t'_{im} &= S_{im} + p_{im} + R_{im}; \\ a_{i1} &= 0; \\ a_{im} &= -R_{i,m-1} - S_{im}. \end{aligned} \tag{3}$$

Let P denote the total number of auxiliary two-machine problems to be solved, such that $P \leq (M-1)$. Set $k = 1$.

Step 1: For each job i, generate the processing times for auxiliary machines 1 and 2 as follows:

$$\begin{aligned} A_i^k &= \sum_{m=k}^{M-1} t'_{im} + \sum_{m=k+1}^{M} a_{im}, \\ B_i^k &= \sum_{m=k+1}^{M} t'_{im} + \sum_{m=k+1}^{M} a_{im}. \end{aligned} \tag{4}$$

Step 2: Solve the auxiliary two-machine problem k whose processing times are given in (4) by using Johnson's rule for the $F2||C_{max}$ case. Let S^k be the schedule obtained and $C_{max}(S^k)$ be its makespan obtained by using the original problem data. If $k < P$, set $k = k + 1$ and return to step 1, otherwise go to step 3.

Step 3: Among the P schedules obtained above, select the schedule S^k with the lowest value of $C_{max}(S^k)$ as the final solution.

Note that the above algorithm in fact consists in generating $M-1$ solutions by applying Johnson's algorithm on two auxiliary machines. The schedule which has the smallest $C_{\max}$ is kept as the final solution. The processing times for the kth ($k = 1, \ldots, M-1$) auxiliary two-machine problem are respectively:

$$A_i^k = \sum_{m=k}^{M-1} p_{im} + S_{ik} - S_{iM},$$
$$B_i^k = \sum_{m=k+1}^{M} p_{im} + R_{iM} - R_{ik}. \tag{5}$$

3.3. THE DFFP ALGORITHM

Proust et al. [23] extended the logic of Campbell et al. [4] to solve the problem being considered here and proposed the new heuristic algorithm DFFP (see also Proust et al. [24]). It also consists in considering a maximum of $M-1$ auxiliary two-machine problems. Each of these auxiliary two-machine problems is solved by using Johnson's algorithm. The solution to the original problem is the best of the $M-1$ which has the lowest makespan. In fact, this heuristic is a synthesis of Campbell et al.'s [4] CDS heuristic algorithm for the $FM||C_{\max}$ scheduling problem and Sule's algorithm mentioned above. The steps of the algorithm are as follows:

Step 0: Initialize the data. Let P be as above (i.e. it is the total number of auxiliary two-machine problems to be solved), such that $P \leq (M-1)$. Set $k = 1$.

Step 1: For each job i, generate the processing times for auxiliary machines 1 and 2 as follows:

$$A_i^k = \sum_{m=1}^{k} p_{im} + S_{i1} - S_{i,M-k+1},$$
$$B_i^k = \sum_{m=M-k+1}^{M} p_{im} + R_{im} - S_{ik}. \tag{6}$$

Step 2: Solve the auxiliary two-machine problem k whose processing times are given in (6) by using Johnson's algorithm for the $F2||C_{\max}$ case. Let S^k be the schedule obtained and $C_{\max}(S^k)$ be its makespan by using the original problem data. If $k < P$, set $k = k + 1$ and return to step 1, otherwise go to step 3.

Step 3: Among the P schedules obtained above, select the schedule S^k with the lowest value of $C_{\max}(S^k)$ as the final solution.

Note that the difference between the algorithm of Szwarc and DFFP only lies in the way they compute the processing times for Johnson's algorithm. Both algorithms

have been compared in Proust et al. [23]. The results show an advantage for DFFP in 70.19% of the cases and for Szwarc's in 19.31% of the cases. The algorithm DFFP has been compared by its authors with the algorithm SH-1 in the particular case of three machines. For 320 problems, 22, or 6.88%, are better solved with SH-1 than with DFFP; 195 problems, or 60.93%, are solved identically with the two heuristic algorithms; 103 problems, or 32.18%, show an improvement with DFFP compared with SH-1. A comparison of the results of DFFP with the optimal sequence was also made. Out of 268 problems, 27, or 10.07%, result in solutions which deviate by more than 5% from the optimum; 142, or 52.99%, result in solutions which deviate by less than 5% from the optimum; 99, or 36.94% result in the optimal solution.

3.4. THE HEURES HEURISTIC

This heuristic algorithm has two phases. In phase 1, and initial schedule is obtained by using the algorithm DFFP. Phase 2 attempts to improve the solution by making adjacent permutations on the initial schedule, by means of Dannenbring's RAES procedure [9]. This is a general procedure to carry out improvements to any initial schedule. The steps of the algorithm HEURES are as follows:

Step 0: Let $S = (a_1, a_2, \ldots, a_n)$ be the schedule obtained by using heuristic DFFP and $C_{\max}(S)$ its makespan, where a_i gives the ith job in the schedule. Let $HEU = S$ and $C_{\max}(HEU) = C_{\max}(S)$. Go to step 1.

Step 1: Let $i = 1$ and Best_i = NULL.

Step 2: Let S'' be the schedule obtained by interchanging a_i and a_{i+1}, in HEU and $C_{\max}(S'')$ its makespan. If $C_{\max}(S'') < C_{\max}(HEU)$ then $C_{\max}(HEU) = C_{\max}(S'')$ and Best_i = i. Set $i = i + 1$. If $i < n$ return to step 2, otherwise go to step 3.

Step 3: If Best_i is not NULL, then let HEU be the schedule obtained by swapping the (Best_i)th and (Best_i + 1)th elements in HEU and return to step 1, otherwise go to step 4.

Step 4: Accept schedule HEU as an approximate solution to the problem.

4. The PHD_{SR} heuristic algorithm

We propose a new heuristic algorithm, called PHD_{SR} (Procedure Han and Dejax), which is based on the explicit consideration of machine workloads to solve $FM|S_{nsd}, R_{nsd}|C_{\max}$ scheduling problems. We take a job-flow management approach to this problem. In other words, minimizing the makespan is equivalent to having the flow pass as quickly as possible through the machines while respecting the imposed constraints. It is the bottleneck machine, that is, the most heavily loaded

one, which determines the flow speed. Therefore the bottleneck machine must begin to run as soon as possible and the time of work remaining to be done on the downstream machines must be speeded once the work on the bottleneck machine is finished. Also, the bottleneck machine should not stop running due to lack of workload immediately in front of it.

Starting with a bottleneck machine, we break all the machines down into the auxiliary machines 1 and 2. The first one combines the machines that are upstream of the bottleneck machine; the second assembles the downstream machines. The bottleneck machine is situated between the two auxiliary machines. In the case where the bottleneck machine is the first (respectively the last), it is considered to be the first (respectively the last) auxiliary machine. We can thus solve the original M-machine problem by creating M auxiliary two-machine problems. Each of these two-machine problems is solved by using Sule's SH algorithm. The calculation is repeated with each machine, ranked in order of decreasing load, being considered successively as the bottleneck machine. The proposed solution to the original problem is the schedule which leads to the lowest makespan $C_{\max}$.

It should be noted that the choice of the values $M-1$ and $(k-1)/(M-k)$ in the formal description below is made so that the processing times for the two auxiliary machines are comparable.

4.1. FORMAL DESCRIPTION

We formally describe the algorithm as follows:

Step 0: Initialize the data. Let J be the set of machines m, $m = 1, \ldots, M$. Calculate the load of each machine. For machine m, the load W_m is included by the setup, processing and removal times. It is defined by:

$$W_m = \sum_{i=1}^{n} \left(S_{im} + p_{im} + R_{im}\right), \quad \forall m \in J. \tag{7}$$

Step 1: Choose a bottleneck machine. Select machine k so that

$$k = \underset{m \in J}{\operatorname{argmax}}\{W_m\}. \tag{8}$$

Step 2: For each job i and from the bottleneck machine k, calculate setup, processing and removal times for the two auxiliary machines. Let S_{i1}, P_{i1}, R_{i1} and S_{i2}, P_{i2}, R_{i2} be the setup, processing and removal times for the auxiliary machines 1 and 2, respectively.

If $k = 1$, then:

$$\begin{aligned} &S_{i1}^k = S_{i1}, \quad P_{i1}^k = (M-1)p_{i1}, \quad R_{i1}^k = R_{i1}; \\ &S_{i2}^k = S_{i2}, \quad P_{i2}^k = \sum_{j=2}^{M} p_{ij}, \qquad R_{i2}^k = R_{iM}. \end{aligned} \tag{9}$$

If $k = M$, then:

$$S_{i1}^k = S_{iM}, \quad P_{i2}^k = \sum_{j=1}^{M-1} p_{ij}, \qquad R_{i1}^k = R_{i,M-1};$$
$$S_{i2}^k = S_{iM}, \quad P_{i2}^k = (M-1)p_{iM}, \quad R_{i2}^k = R_{iM}. \tag{10}$$

Otherwise, in the general case:

$$S_{i1}^k = S_{i1}, \quad P_{i1}^k = \sum_{j=1}^{k-1} p_{ij}, \qquad R_{i1}^k = R_{ik};$$
$$S_{i2}^k = S_{ik}, \quad P_{i2}^k = \frac{k-1}{M-k} \sum_{j=k+1}^{M} p_{ij}, \quad R_{i2}^k = R_{iM}. \tag{11}$$

Step 3: Solve the auxiliary two-machine problem whose bottleneck machine is k and the setup, processing and removal times are given in (9), (10) or (11), respectively, by using Sule's algorithm for the $F2|S_{\text{nsd}}, R_{\text{nsd}}|C_{\max}$ case. Let S^k be the schedule obtained and $C_{\max}(S^k)$ its makespan by using the original problem data. Remove k from the set J. If $J = \varnothing$, go to step 4, otherwise go to step 1.

Step 4: Among all the schedules obtained above, select the schedule S^k with the lowest value of $C_{\max}(S^k)$ as the final solution.

4.2. REMARKS

In most cases setup and removal times are small related to processing times. So, as most authors do, we assume:

- no job is such that the setup time on a given machine is greater than the sum of its setup and processing times on any preceding machine;
- the completion time of the last operation of the last job on the last machine is counted at the end of removal. No other time is assumed to be greater than this on the preceding machines.

The two special cases are illustrated in fig. 1.

4.3. COMPLEXITY ANALYSIS

The PHD_{SR} algorithm relies on the application of Sule's algorithm M times. The complexity of Sule's algorithm is $O(n \log n)$. Therefore the complexity of the PHD_{SR} algorithm is $O(Mn \log n)$.

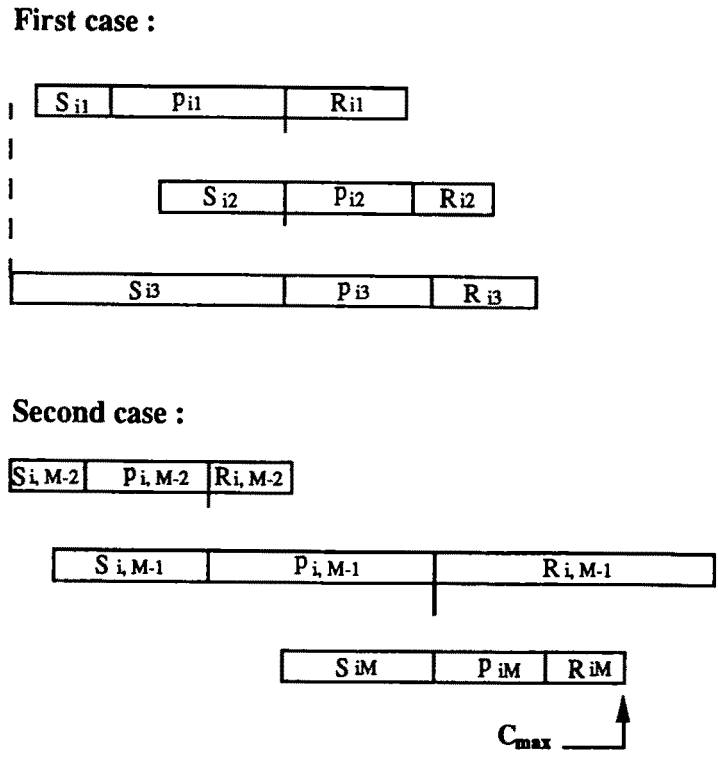

Fig. 1. Special cases not taken into account.

5. Computational experience

To analyze the performance of the algorithms PHD_{SR} and DFFP for finding feasible schedules, we compare them with an exact algorithm on randomly generated problems of sizes up to 100 jobs and 20 machines. The optimal solution was obtained using a branch and bound procedure based on a new lower bound described in Han and Dejax [19]. Previously published optimal methods, see for example Proust et al. [24], could not be used to solve problems of the size presented here. The results of computational experience are reported below.

5.1. GENERATION OF PROBLEM DATA

Our experiments were conducted by comparing the results, obtained using the heuristics PHD_{SR} and DFFP, with those of an optimal branch and bound procedure which we have developed [19]. To generate the problem data, a uniform distribution on an interval $[a, b]$ was employed for a set of 4 scenarios. In scenario 1, no specific bottleneck is generated. In scenario 2, we generate bottleneck machines with various

coefficients. In the third scenario we analyze the performance of the algorithms in the case of varying interval widths for the setup and removal times. In the fourth scenario, we analyze the effect of the interval lower bound for the setup and removal times.

For the first scenario, the sampling intervals for the processing times were $[a_p, b_p] \in \{[10,60], [5,55]\}$ and the sampling intervals for the setup and removal times $[a_{sr}, b_{sr}]$ were $[1,5]$. The problems of size $n \times M$ are $\{(8 \times 4), (8 \times 5), (8 \times 6), (10 \times 4), (10 \times 6), (10 \times 8), (13 \times 4), (13 \times 6), (13 \times 8), (13 \times 10)\}$. For each problem size we generated 30 tests, a total of 300 test problems.

For the second scenario, a coefficient with the value of 2, 3, 4, or 5 is chosen at random and attributed to any column for each problem so as to make a bottleneck appear in the system. We chose a scenario for which the sampling intervals $[a_p, b_p]$ for the processing times are $\{[5,55], [10,50], [5,45], [10,60]\}$ and the sampling intervals for the setup and removal times $[a_{sr}, b_{sr}]$ are $\{[1,5], [1,10]\}$. The problems of size $n \times M$ are $\{(10 \times 5), (15 \times 8), (20 \times 8), (25 \times 10), (30 \times 10), (40 \times 13), (50 \times 15), (60 \times 6), (80 \times 8), (100 \times 20)\}$. For each problem size we generated 30 tests, a total of 300 tests.

For the third scenario, the sampling intervals for the setup and removal times $[a_{sr}, b_{sr}]$ are $\{[1,5], [1,10], [1,15]\}$ for each one of three sampling intervals of the processing times $\{[20,80], [30,80], [40,80]\}$. The problems of size $n \times M$ are $\{(8 \times 5), (9 \times 6), (10 \times 7)\}$. For each problem size and each corresponding sampling interval for the setup, removal times and processing times we generated 10 tests, a total of 90 tests.

In the fourth scenario, we kept the sampling intervals for the processing times and the width of the sampling intervals for the setup and removal times constant and made the lower bound intervals for the setup and removal times vary. The sampling intervals for the processing times were $\{[20,80], [30,80], [40,80]\}$ for each one of three sampling intervals for the setup and removal times $\{[1,5], [5,10], [10,15]\}$. The problems of size $n \times M$ are $\{(8 \times 5), (10 \times 6), (12 \times 8)\}$. For each problem size and each corresponding sampling interval for the setup, removal times and processing times we generated 10 tests, a total of 90 tests.

5.2. MEASURE OF EFFECTIVENESS

Since an optimum solution for the generated problems could be found, the percentage deviation of the makespan $C_{\max}(\mathrm{A})$ of the heuristic A from its optimal makespan $C_{\max}(\mathrm{OPT})$ can be calculated by the following equation:

$$p_i(\mathrm{A}) = 100 \frac{[C_{\max}(\mathrm{A}) - C_{\max}(\mathrm{OPT})]}{C_{\max}(\mathrm{OPT})}. \tag{12}$$

If $p_i(\mathrm{A}) = 0$, then algorithm A yields an optimal solution on the tested problem i. The numerical value of $p_i(\mathrm{A})$ is the relative deviation between the makespans

obtained by using heuristic algorithm A and the optimal algorithm. For comparison of the effectiveness of various algorithms, the average (p_{avg}) and maximum (p_{max}) were computed for N test problems where $p_{avg} = [\sum p_i]/N$. The percentage deviation for the heuristic PHD_{SR} and for DFFP was calculated by letting respectively $C_{max}(A) = C_{max}(PHD_{SR})$ and $C_{max}(A) = C_{max}(DFFP)$. If $p_i(PHD_{SR}) < p_i(DFFP)$, then algorithm PHD_{SR} performs better than algorithm DFFP in finding a minimum makespan schedule for test problem i while $p_i(PHD_{SR}) = p_i(DFFP)$ indicates that both algorithms yield identical results.

5.3. COMPUTATIONAL EXPERIENCE

The PHD_{SR}, DFFP and a new branch and bound procedure were implemented in the C language and tested on a working station SUN 3 (processor Motorola 68020, 20MHz). We present the results of the comparisons below.

5.3.1. Comparisons in the general case

In the general case (first scenario), a total of 300 problems was solved respectively by the heuristic algorithms PHD_{SR}, DFFP and the optimal branch and bound algorithm. In 134 of these problems (44.46%) the algorithm PHD_{SR} found the optimal solution. The algorithm DFFP optimally solved only 66 problems (22%). The PHD_{SR} algorithm resulted in a lower than 5% deviation value in 149 (49.67%) problems and the DFFP in 193 (64.33%). DFFP found their solution with more than 5% deviation from the optimal solution in 41 of these problems (13.67%) and only in 17 or 5.67% problems for the algorithm PHD_{SR}. On average p_{avg} DFFP's makespan was 0.93% greater. The maximum value of the deviation p_{max} is 10.05% for PHD_{SR} and 12.90% for DFFP. The results of the first scenario show that the algorithm PHD_{SR} performs better than the DFFP algorithm. The general results are given in table 1 which can be read as follows: In the first two columns, each line corresponds to a number of jobs (n) and machines (M). Each of the four following columns corresponds to the interval in which the comparison indicator p_i is situated. In each line we indicated the number of problems leading to an indicator in the corresponding interval. The average p_{avg} and the maximum value of the deviation p_{max} are indicated in the seventh and eighth columns. The left part (for PHD_{SR}) and right part (for DFFP) are symmetrical.

5.3.2. Comparisons with an explicit bottleneck machine

In case of the presence of an explicit bottleneck machine (second scenario), a total of 300 problems was solved respectively by the heuristic algorithms PHD_{SR}, DFFP and the optimal branch and bound algorithm. In 237 of these problems (79%) the algorithm PHD_{SR} found an optimal solution. The algorithm DFFP optimally

Table 1

Comparison in the general case.

		PHD_{SR}						DFFP					
n	M	[0]	]0,2]	]2,5]	]5,∞]	p_{max}	p_{avg}	[0]	]0,2]	]2,5]	]5,∞]	p_{max}	p_{avg}
8	4	4	11	10	5	10.05	2.82	3	8	11	8	7.47	3.29
8	5	11	8	10	1	5.30	1.52	9	6	7	8	12.75	3.11
8	6	16	7	5	2	6.19	1.15	7	7	13	3	6.96	2.37
10	4	18	9	3	0	4.93	0.70	7	12	9	2	9.90	1.92
10	6	16	9	3	2	6.55	0.93	6	14	5	5	10.83	2.19
10	8	13	12	5	0	4.62	0.81	13	13	4	0	4.69	0.77
13	4	17	7	5	1	5.12	1.07	3	12	11	4	10.24	2.31
13	6	15	11	4	0	4.08	0.88	4	15	9	2	8.78	1.94
13	8	13	12	5	0	3.37	0.73	7	16	5	2	12.90	1.60
13	10	11	7	6	6	8.96	2.28	7	10	6	7	10.54	2.73
Total		134	93	56	46	10.05	1.29	66	113	80	41	12.90	2.22
In %		44.66	31	18.67	5.67			22	37.67	26.67	13.67		

Table 2

Comparison with an explicit bottleneck machine.

		PHD_{SR}						DFFP					
n	M	[0]	]0,2]	]2,5]	]5,∞]	p_{max}	p_{avg}	[0]	]0,2]	]2,5]	]5,∞]	p_{max}	p_{avg}
10	5	17	11	2	0	3.09	0.36	1	15	11	3	8.33	2.29
15	8	27	3	0	0	1.07	0.06	11	16	3	0	3.51	0.76
20	8	18	12	0	0	1.41	0.27	4	20	6	0	4.60	1.25
25	10	25	5	0	0	0.79	0.06	4	26	0	0	1.48	0.41
30	10	24	6	0	0	0.34	0.03	4	24	2	0	2.41	0.57
40	13	25	5	0	0	0.43	0.02	2	28	0	0	0.98	0.34
50	15	22	8	0	0	0.45	0.05	3	26	1	0	2.14	0.40
60	6	28	2	0	0	0.04	0.00	1	29	0	0	1.74	0.50
80	8	28	2	0	0	0.04	0.00	3	27	0	0	0.28	0.11
100	20	23	7	0	0	0.15	0.02	1	29	0	0	0.69	0.15
Total		237	61	2	0	3.09	0.08	34	240	23	3	8.33	0.68
In %		79	20.33	0.67	0			11.33	80	7.67	1		

solved only 34 problems (11.33%). The PHD_{SR} algorithm resulted in a lower than 5% deviation value in 63 (21%) problems and the DFFP in 263 (87.67%). DFFP found their solution with more than 5% deviation from the optimal solution in 3 of these problems (1%) and in zero problems for the algorithm PHD_{SR}. On average, p_{avg}DFFP's makespan was 0.60% greater. The maximum value of the deviation p_{max} is 3.09% for PHD_{SR} and 8.33% for DFFP. The general results are given in table 2 which is similar to table 1.

5.3.3. *Sensitivity analysis*

In the third scenario, we kept the sampling intervals for the processing times constant and made the sampling intervals for the setup and removal times vary. The results show that the behavior of the two algorithms is worse as the width of sampling intervals for the setup and removal times is increased while keeping the sampling intervals for the processing times constant. But the PHD_{SR} algorithm always performs better than the DFFP algorithm. The general results are given in table 3.

Table 3

Comparison with $a_{sr} = C$.

$[a_p, b_p] \in$	$[a_{sr}, b_{sr}] \in$ $n \times M$	[1, 5] 8×5	[1, 10] 9×6	[1, 15] 10×7
[20, 80]	$\frac{p_i(\text{DFFP})}{p_i(\text{PHD}_{SR})}$	2.84 / 1.44	3.45 / 2.84	4.47 / 3.24
[30, 80]		3.26 / 2.44	3.41 / 3.07	5.41 / 3.80
[40, 80]		3.32 / 2.18	3.30 / 2.39	3.76 / 2.79

Table 4

Comparison with $b_{sr} - a_{sr} = C$.

$[a_p, b_p] \in$	$n \times M$	$[a_{sr}, b_{sr}] \in$	[1, 5]	[5, 10]	[10, 15]
[20, 80]	8×5	$\frac{p_i(\text{DFFP})}{p_i(\text{PHD}_{SR})}$	4.25 / 2.77	3.31 / 2.10	3.29 / 1.85
$[a_p, b_p] \in$	$n \times M$	$[a_{sr}, b_{sr}] \in$	[5, 10]	[10, 15]	[15, 20]
[30, 80]	10×6	$\frac{p_i(\text{DFFP})}{p_i(\text{PHD}_{SR})}$	2.94 / 2.46	2.49 / 2.12	2.32 / 1.85
$[a_p, b_p] \in$	$n \times M$	$[a_{sr}, b_{sr}] \in$	[5, 15]	[15, 25]	[25, 35]
[40, 80]	12×8	$\frac{p_i(\text{DFFP})}{p_i(\text{PHD}_{SR})}$	2.60 / 1.96	2.26 / 1.94	1.26 / 0.96

In the fourth scenario, we kept the sampling intervals for the processing times and the width of the sampling intervals for the setup and removal times constant and made the lower bound of the intervals for the setup and removal times vary. The results show that the behaviour of the two algorithms is better as the lower bound of the intervals for the setup and removal times is increased. The performance of the two algorithms varies as the lower bound of the intervals for the processing times is increased. The PHD_{SR} algorithm performs better than the DFFP algorithm in all the cases. The general results are given in table 4.

The computing times required by the PHD_{SR} algorithm and the DFFP algorithm for solving any of the above problems are less than one second CPU time. Our branch and bound procedure found, in a little under sixty minutes, an optimal schedule to any of the problems mentioned above.

6. Conclusion

In this paper we have discussed the static flowshop scheduling problem where setup, processing, and removal times are separable and sequence independent. Through a synthesis of known results, we have proposed a new approach based on the machine workload to find a (hopefully) good solution to the problem when the objective function is the minimization of makespan. The computational results show that our algorithm yields excellent results. Additionally, this algorithm can be used to find an initial solution to be used in branch and bound algorithms in order to increase their efficiency.

Acknowledgements

The authors thank three anonymous referees whose comments greatly improved the quality and presentation of the paper.

References

[1] J. Adams, E. Balas and D. Zawact, The shifting bottleneck procedure for job shop scheduling, Manag. Sci. 34(1988)391–401.

[2] K.R. Baker, *Introduction to Sequencing and Scheduling* (Wiley, New York, 1974).

[3] E. Balas, Machine sequencing via disjunctive graphs: An implicit enumeration algorithm, Oper. Res. 17(1969)941–957.

[4] H.G. Campbell, R.A. Dudek and N.L. Smith, A heuristic algorithm for the *n*-job, *m*-machine sequencing problem, Manag. Sci. 16(1970)630–637.

[5] J. Carlier and E. Pinson, An algorithm for solving the job-shop problem, Manag. Sci. 35(1989) 164–176.

[6] J. Carlier and E. Pinson, A practical use of Jackson's preemptive schedule for solving the job-shop problem, Ann. Oper. Res. 26(1990)269–287.

[7] R.N. Conway, W.L. Maxwell and L.W. Miller, *Theory of Scheduling* (Addison–Wesley, Reading, MA, 1967).

[8] B.D. Corwin and A.O. Esogbue, Two-machine flowshop scheduling problems with sequence setup times: A dynamic approach, Naval Res. Log. Quarterly 21(1974)1174–1182.
[9] D.G. Dannenbring, An evaluation of flowshop sequencing heuristics, Manag. Sci. 23(1977) 1174–1182.
[10] L.F. Escudero, An inexact algorithm for part input sequencing with side constraints in FMS, Int. J. Flexible Manuf. Syst. 1(1989)143–174.
[11] S. French, *Sequencing and Scheduling: An introduction to the Mathematics of Job Shop* (Wiley, New York, 1982).
[12] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guided Tour to the Theory of NP-Completeness* (Freeman, San Francisco, CA, 1979).
[13] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discr. Math. 5(1979)287–326.
[14] J.N.D. Gupta, A search algorithm for generalized flowshop scheduling problem, Comp. Oper. Res. 2(1975)83–90.
[15] J.N.D. Gupta, A review of flowshop scheduling research, in: *Disaggregation: Problems in Manufacturing and Service Organizations*, ed. Ritzman et al. (Martinus Nijhoff, The Hague, 1979) pp. 363–388.
[16] J.N.D. Gupta, Flowshop schedules with sequence dependent setup times, J. Oper. Res. Soc. Japan 29(1986)206–219.
[17] J.N.D. Gupta and W.P. Darrow, The two-machine sequence dependent flowshop scheduling problem, Europ. J. Oper. Res. 24(1986)439–446.
[18] W. Han and P. Dejax, Une heuristique pour le problème d'ordonnancement de type $n/M/F/C_{\max}$ avec la présence de machines goulots, RAIRO, Recherche Opérationelle 24(1990)315–330.
[19] W. Han and P. Dejax, A new branch and bound method for the M-stage flowshop scheduling with set-up, processing and removal times separated, Cahiers d'Etudes et de Recherche, no. 91-1, Ecole Centrale de Paris (1991).
[20] S.M. Johnson, Optimal two and three-stage production schedules with setup times included, Naval Res. Log. Quarterly 1(1954)61–68.
[21] J.K. Lenstra, *Sequencing by Enumerative Methods*, Mathematical Center Tract (Mathematisch Centrum, Amsterdam, 1976).
[22] Y.B. Park, C.D. Pegden and E.E. Enscore, A survey and evaluation of static flowshop scheduling heuristic, Int. J. Prod. Res. 22(1984)127–141.
[23] C. Proust, M. Drogou, J.M. Foucher and E. Foucheyrand, Une heuristique pour le problème d'ordonnancement statistique de type *n/m/*flowshop avec prise en compte des temps de montage et démontage d'outils, RAIRO, APII, 22(1988)37–54.
[24] C. Proust, J.N.D. Gupta and V. Deschamps, Flowshop scheduling with set-up, processing and removal times separated, Int. J. Prod. Res. 29(1991)479–493.
[25] A.H.G. Rinnooy Kan, *Machine Scheduling Problems: Classification, Complexity and Computations* (Martinus Nijhoff, The Hague, 1976).
[26] B.N. Srikar and S. Ghosh, A MILP model for n-job, M-stage flowshop with sequence dependent setup times, Int. J. Prod. Res. 24(1986)1459–1474.
[27] D.R. Sule, Sequencing n jobs on two machines with setup, processing and removal times separated, Naval Res. Log. Quarterly 29(1982)517–519.
[28] D.R. Sule and K.Y. Huang, Sequencing on two and three machines with setup, processing and removal times, Int. J. Prod. Res. 24(1983)1459–1474.
[29] W. Szwarc, Flowshop problems with time lags, Manag. Sci. 29(1983)477–481.
[30] W. Szwarc and J.N.D. Gupta, A flow-shop problem with sequence-dependent additive setup times, Naval Res. Log. Quarterly 34(1987)619–627.
[31] T. Yoshida and K. Hitomi, Optimal two-stage production scheduling with setup times separated, AIIE Trans. 11(1979)261–264.