Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 159 (2004) 66–82
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Discrete Optimization

Scheduling flexible flow lines with sequence-dependent setup times

Mary E. Kurz [a,*], Ronald G. Askin [b]

[a] Department of Industrial Engineering, Clemson University, Clemson, SC 29634-0920, USA
[b] Department of Systems and Industrial Engineering, The University of Arizona, Tucson, AZ 85721-0020, USA

Received 16 October 2001; accepted 19 May 2003
Available online 30 August 2003

Abstract

This paper examines scheduling in flexible flow lines with sequence-dependent setup times to minimize makespan. This type of manufacturing environment is found in industries such as printed circuit board and automobile manufacture. An integer program that incorporates these aspects of the problem is formulated and discussed. Because of the difficulty in solving the IP directly, several heuristics are developed, based on greedy methods, flow line methods, the Insertion Heuristic for the Traveling Salesman Problem and the Random Keys Genetic Algorithm. Problem data is generated in order to evaluate the heuristics. The characteristics are chosen to reflect those used by previous researchers. A lower bound has been created in order to evaluate the heuristics, and is itself evaluated. An application of the Random Keys Genetic Algorithm is found to be very effective for the problems examined. Conclusions are then drawn and areas for future research are identified.

Keywords: Scheduling; Heuristics; Genetic algorithms; Flexible flow lines

1. Introduction

Traditional manufacturing systems have taken many general forms. In increasingly complex manufacturing environments, more complex manufacturing systems have been created in order to address such factors as limited capacity and complicated process plans. For example, the semiconductor industry uses reentrant flow lines, in which multiple machines may exist at each stage and jobs revisit previous stages many times in a cyclic manner. The printed circuit board and automobile industries make use of flow lines with multiple machines at some stages and allow jobs to skip stages (Piramuthu et al., 1994; Agnetis et al., 1997). Moreover, these industries encounter sequence-dependent setup times which result in even more difficult scheduling problems. The scheduling objective in such industries may vary. Due date related criteria may be

* Corresponding author. Tel.: +1-864-6564652; fax: +1-864-6560795.
E-mail addresses: mkurz@clemson.edu (M.E. Kurz), ron@sie.arizona.edu (R.G. Askin).

0377-2217/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0377-2217(03)00401-6

important. The makespan criterion has been used by many researchers and has been selected for this research. Scheduling to minimize makespan in flow lines with multiple parallel machines, jobs that may skip stages, and sequence-dependent setup times is the focus of this paper. This kind of manufacturing environment introduces new difficulties that scheduling in simple flow lines, for example, did not address.

To begin, we define the scope of the problem considered in this research. We use the term "hybrid" flow line to indicate flow lines with the presence of multiple identical machines in parallel at some or all stages, though jobs still require processing at exactly one machine per stage. A flexible flow line is a hybrid or (regular) flow line where at least one job need not be processed on any machines in at least one stage. That is, every job must be processed on at most one machine per stage. A flexible flow line consists of several stages in series. A job may not revisit a stage that it has already visited. Each stage has at least one machine, and at least one stage must have more than one machine. At this point, this structure may be considered a hybrid flow line or a flow line with multiple machines. However, the feature that makes our application a *flexible* flow line is that jobs may skip stages. This could occur in an industry in which some jobs do not require an operation. This situation exists in the printed circuit board manufacturing line modeled by Wittrock (1985, 1988). Three of the thirteen part types required processing on only two of the three stages.

The potential variants of the basic flexible flow line described above that can be studied are nearly limitless. Now we shall describe the particular features of this research. All data in this problem are known deterministically when scheduling is undertaken. Machines are available at all times, with no breakdowns or scheduled or unscheduled maintenance. Jobs are always processed without error. Job processing cannot be interrupted (no preemption is allowed) and jobs have no associated priority values. Infinite buffers exist between stages and before the first stage and after the last stage; machines cannot be blocked because the current job has nowhere to go. There is no travel time between stages; jobs are available for processing at a stage immediately after completing processing at the previous stage. The ready time for each job is the larger of 0 and the time it completes processing on the previous stage. Machines in parallel are identical in capability and processing rate. A key characteristic of this research topic is that non-anticipatory sequence-dependent setup times exist between jobs at each stage. After completing processing of one job and before beginning processing of the next job, some sort of setup must be performed. The length of time required to do the setup depends on both the prior and the current job to be processed; that is, the setup times are sequence-dependent. Piramuthu et al. (1994) incorporate sequence-dependent setup times in their model of an actual printed circuit board line. Rios-Mercado and Bard (1998) note that sequence-dependent setup times are found in the container manufacturing industry as well as the printed circuit board industry. The formulations in Rios-Mercado and Bard (1998) indicate the assumption has been made that setup can only be performed after the machine is no longer processing any job and the job for which setup is being performed is ready. The examples in Rios-Mercado and Bard (1999a,b) discussing the container manufacturing industry state that the machines must be adjusted whenever the dimensions of the containers are changed, which presumably cannot be done until the machine is idle. We follow this concept and require the machine on which setup is to be performed to be idle and the job for which setup is required to be available as well.

This paper continues with a review of related research in Section 2. An integer programming model is presented and described in Section 3. Lower bounds are developed in Section 4 for use in evaluating schedules produced with the four heuristics described in Section 5. Using randomly generated test problems, described in Section 6, the heuristics are compared in Section 7. The quality of the lower bounds is also discussed. Section 8 concludes the paper.

2. Literature review

This literature review will have one component regarding scheduling and a second regarding the random keys genetic algorithm. Scheduling in flexible flow lines and the less general hybrid flow lines can be

organized by the approaches used to solve them. We will categorize approaches based on the use of branch-and-bound, extensions of previous flow line techniques, applications of metaheuristics and development of new techniques.

Salvador (1973) first considered multiple machines at serial stages (hybrid flow line) with no buffers between stages. Branch-and-bound techniques are applied to determine the optimal permutation schedule in terms of makespan. Brah and Hunsucker (1991) used branch-and-bound in the hybrid flow shop with an arbitrary number of stages and intermediate buffers. Moreover, they provide a means by which non-permutation schedules or schedules with inserted idle time can be created. Rajendran and Chaudhuri (1992) also utilized branch-and-bound but restricted the resultant schedule to the set of permutation schedules. Santos et al. (1995) combined permutation schedules with the FIFO queuing discipline at all stages after the first stage. The jobs enter the line according to one of the $n!$ permutations and then begin processing at every stage thereafter in the order they completed processing at the previous stage. The best permutation solution is found by considering every possible permutation of jobs that can enter the line, tempered by the introduction of a lower bound on the optimal makespan.

Extending heuristics developed for the single line flowshop has been considered by numerous authors. Sriskandarajah and Sethi (1989) examined worst case performance for various heuristics based on Johnson's Rule applied to two stage hybrid flow shops. Lee and Vairaktarakis (1994) developed heuristics for multistage hybrid flow shops by extending results for a two stage hybrid flow shop and aggregating machines at each stage. Johnson's Rule was also applied by Gupta (1997) to the case with one machine in the first stage and multiple machines in the second stage. Oguz et al. (1997) examined a three stage flexible flow line with one machine at each stage where the difference in job routings were handled as different job types (jobs visit either stages 1 and 3 or 2 and 3). Johnson's Rule was used for each of the types of jobs. Ding and Kittichatphayak (1994) modeled the hybrid flow line as an extension of a single line flow shop, adapting Campbell et al.'s (1970) method for flow shops or placing jobs at the end of the current sequence considering the idle time of the machines. Riane et al. (1998) considered a three stage hybrid flow shop with two machines at the second stage and one at each of the other stages. They developed a dynamic programming-based heuristic based on the Campbell et al. heuristic for single flow lines and a branch-and-bound heuristic. Santos et al. (1996) also considered the use of heuristics developed for the single flow shop case as a method to generate an initial permutation schedule that would then be followed by the application of FIFO. Brah and Loo (1999) evaluated how heuristics developed for the single flow line case would perform in the hybrid environment.

Nowicki and Smutnicki (1998) built on their prior experience with tabu search in single machine (Nowicki and Smutnicki, 1994) and flow shop (Nowicki and Smutnicki, 1996) scheduling. Robust local search improvement techniques for flexible flow line scheduling were considered by Leon and Ramamoorthy (1997). Rather than considering neighborhoods of the schedules, they considered neighborhoods of the problem data. The perturbed data is then used by a problem specific heuristic to generate a solution whose quality is assessed using the original problem data. Lee et al. (1997) have applied genetic algorithms to the joint problem of determining lot sizes and sequence to minimize makespan in flexible flow lines. Though this research included sequence-dependent setup times, buffers between stages were limited and jobs must be processed in the same order on the machines. Combining genetic algorithms with simulated annealing was also considered.

The examination of flexible flow lines as defined in this paper and the development of heuristics specifically for this problem began with Wittrock (1985, 1988). Kochhar and Morris (1987) model flexible flow lines in a more complete manner in that they allow for setups between jobs, finite buffers which may cause blocking, and machine down-time. They extend a Wittrock algorithm and evaluate several policies with a deterministic simulation. Sawik (1992) has developed numerous results for the flexible flow line scheduling problem. The basic model includes factors such as transportation time between stages and non-zero release times. However, sequence-dependent setup times are not included. The *Route Idle Time Minimization*

(*RITM*) heuristic aims to minimize makespan by minimizing the idle time of the machines. In this case, buffers are limited in size so that blocking can occur. Later, Sawik (1994, 1995) extended the *RITM* heuristic to the case of no buffers between stages.

While many papers have been written in the area of scheduling hybrid and flexible flow lines, many of them are restricted to special cases of two stages or specific configurations of machines at stages. There does not seem to be any published work addressing heuristics for flexible flow lines (multiple serial stages that need not have the same number of machines per stage and jobs that need not visit all stages) with sequence-dependent setup times. Rios-Mercado and Bard (1998, 1999a,b) do consider flow shops with sequence-dependent setup times in several papers, but these papers all require exactly one machine per stage and permutation schedules (all jobs visit each stage in the same order).

Genetic algorithms were introduced by Holland (1975). He provided the basic framework for genetic algorithms: chromosomes represent solutions which reproduce based on how well they solve the problem at hand in a manner analogous to survival of the fittest. A key feature of genetic algorithms is their randomness. Chromosomes are chosen to reproduce randomly and experience changes based on random chance, as organisms do in the natural environment. Theoretically, the best chromosome will survive to the final generation of chromosomes. The chromosomal representation of the scheduling information can take many forms and influence the types of genetic operators. A common problem for combinatorial applications of genetic algorithms is that some operations may create feasibility problems. Bean (1994) has introduced an alternative method to encode problem solutions using random numbers called a Random Keys Genetic Algorithm (RKGA), which has been applied to resource allocation problems, quadratic assignment problems, multiple machine tardiness scheduling problems, jobshop makespan minimization problems and the generalized traveling salesman problem (Bean, 1994; Norman and Bean, 1999; Snyder and Daskin, 2001). RKGA will be discussed further in a later section.

3. Integer programming model

The problem addressed in this research can be expressed formally as an integer program. Let g be the number of stages. Let n be the number of jobs to be scheduled and m^t be the number of machines at stage t. We assume that machines are initially setup for a nominal job 0 and must finish setup for a tear down job $n+1$ at every stage. We have the following definitions.

n	number of true jobs to be scheduled
g	number of serial stages
g_j	last stage visited by job j
m^t	number of machines at stage t
p_i^t	processing time for job i at stage t (assumed to be integral)
s_{ij}^t	setup time from job i to job j at stage t
S_i	set of stages visited by job i
S^t	set of jobs that visit stage $t = \{i : p_i^t > 0\}$
c_i^t	completion time for job i at stage t
x_{ij}^t	1 if job i is scheduled immediately before job j at stage t and 0 otherwise

The processing times of jobs 0 and $n+1$ are set at 0 and the setup times are times to move from and to the nominal set point state. We assume that all jobs currently in the system must be completed at each stage before the jobs requiring scheduling may begin setup. The completion times of job 0 at each machine at each stage are set to the earliest time setup may begin at that stage. We include the restriction that every stage must be visited by at least as many jobs as there are machines in that stage. This is expressed by the

inequality $|S^t| \geqslant m^t$, $t = 1, 2, \ldots, g$, so $n \geqslant \max_{t=1,2,\ldots,g}\{m^t\}$. If a stage is visited by fewer jobs than there are machines, there is not a difficult sequencing decision to be made, because each job could be assigned its own machine. The formulation also assumes that a job that does not visit a stage has a processing time of 0. That is, $p_i^t = 0$ if $i \notin S^t$. Additionally, we assume $p_i^t \geqslant 1$ if $i \in S^t$. The formulation becomes

$$P: \min \quad z, \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{oj}^t = m^t, \quad t = 1, \ldots, g, \tag{2}$$

$$\sum_{j \in \{S^t, n+1\}} x_{ij}^t = 1, \quad i = 1, \ldots, n, \ \ t \in S_i, \tag{3}$$

$$\sum_{i \in \{0, S^t\}} x_{ij}^t = 1, \quad j = 1, \ldots, n, \ \ t \in S_i, \tag{4}$$

$$c_j^t - c_i^t + M^t(1 - x_{ij}^t) \geqslant s_{ij}^t + p_j^t, \quad i = 0, \ldots, n, \ \ j = 1, \ldots, n, \ \ t \in S_i, \tag{5}$$

$$c_j^t - c_j^{t-1} + M_j^t(1 - x_{ij}^t) \geqslant s_{ij}^t + p_j^t, \quad i = 0, \ldots, n, \ \ j = 1, \ldots, n, \ \ t \in S_i - \{1\}, \tag{6}$$

$$\begin{cases} x_{ij}^t \leqslant p_j^t, \\ x_{ji}^t \leqslant p_j^t, \end{cases} \quad i, j \in \{0, 1, \ldots, n, n+1\}, \ \ t = 1, \ldots, g, \tag{7}$$

$$\left\{ \begin{array}{ll} c_j^1 - c_0^1 \geqslant M^t p_j^1, & j = 1, \ldots, n, \\ c_j^t - c_j^{t-1} \geqslant M^t p_j^t, & j = 1, \ldots, n, \ \ t = 2, \ldots, g \end{array} \right\}, \tag{8}$$

$$c_j^t \geqslant c_0^t, \quad j = 1, \ldots, n, \ \ t = 1, \ldots, g, \tag{9}$$

$$z \geqslant c_j^{g_j}, \quad j = 1, \ldots, n, \tag{10}$$

$$\begin{aligned} &x_{ij}^t \in \{0, 1\}, \quad i, j \in \{0, 1, \ldots, n, n+1\}, \ \ t = 1, \ldots, g, \\ &x_{ij}^t = 0, \quad i = j, \ \ t = 1, \ldots, g, \end{aligned} \tag{11}$$

$$c_j^t \geqslant 0, \quad j = 0, \ldots, n, \ \ t = 1, \ldots, g. \tag{12}$$

This formulation is based on the TSP. Each stage exists independently except that stage t's completion times are stage $t+1$'s ready times. Eq. (1) defines the objective function which is to minimize the makespan z. We assume setup times satisfy some sufficient condition to ensure that an optimal solution will always exist that utilizes all m^t machines at each stage, as in the single stage case. For instance,

$$s_{0j}^t \leqslant \min_{i \neq 0} (p_i^t + s_{ij}^t) \quad \forall j$$

and

$$p_j^t + s_{0j}^t \leqslant \frac{\sum_{i \neq j} (p_i^t + \min_k s_{ki}^t)}{m^t - 1} \quad \forall j$$

are both sufficient. Constraint set (2) ensures that m^t machines are scheduled in each stage. Constraint sets (3) and (4) ensure that each job is scheduled on one and only one machine in each stage. Constraint set (5) forces job j to follow job i by at least i's processing time plus the setup time from i to j if i is immediately before j. The value M^t is an upper bound on the time stage t completes processing, similar to the upper bound A in Rios-Mercado and Bard (1998).

$$M^1 = \sum_{i=1}^{n} \left(p_i^1 + \max_{j \in \{0, \ldots, n\}} s_{ji}^1 \right) \quad \text{and} \quad M^t = M^{t-1} + \sum_{i=1}^{n} \left(p_i^t + \max_{j \in \{0, \ldots, n\}} s_{ji}^t \right).$$

Constraint set (6) forces job j at stage t to complete after it completes at stage $t-1$, plus its processing time at stage t, plus the setup time from its predecessor to j. The value M_j^t is set to $M_j^t = \max_i (s_{ij}^t) + p_j^t$. Constraint sets (5) and (6) together ensure that a job cannot begin setup until it is available (done at the previous stage) and the previous job at the current stage is complete. Constraint sets (5) and (6) also serve as sub-tour elimination constraints. Constraint set (7) ensures that jobs that do not visit a stage are not assigned to that stage. Constraint sets (8) and (9) ensure that the completion time of a job at stage t, that does not visit stage t, is set to the job's completion time at stage $t-1$. The value M^t is an upper bound on the time stage t completes processing and is the same as that used in constraint set (5). Constraint set (10) links the decision variables c_j^g and z. Constraint sets (11) and (12) provide limits on the decision variables.

Due to the fact that each machine at each stage is a TSP once jobs have been assigned to the machine, this problem is NP-hard. In this research, we need not only sequence jobs on machines, we must consider which jobs are to be assigned to the machines. Fourteen small problems were considered in order to evaluate the feasibility of solving this MIP directly. Each problem has integer processing times selected from a uniform distribution between 50 and 70 and integer setup times selected from a uniform distribution between 12 and 24. Table 1 contains additional problem characteristics. Problems 1–9 all have 1 machine at every stage. Problems 10–14 have stages with different numbers of machines at each stage; the number of machines at each stage is shown in order. Problems 4, 5, 11 and 12 have different sets of jobs visiting each stage; the number of jobs that visit each stage is shown. In the other problems, all jobs visit all stages. These problems have been solved in CPLEX 7.5 on a Sun Microsystems Enterprise 6000 with UltraSPARC-II 336 MHz cpus and 4.0 GB of memory. Each problem was allowed a maximum of 7200 seconds of CPU time (two hours) using the CPLEX setting "set timelimit 7200". Of these, only two were solved to optimality in the two hour time limit. These problems both had 2 stages, 1 machine per stage and 6 jobs which visited both stages. They were solved in 810.37 and 695.24 seconds respectively. The other twelve problems were stopped due to the time limit before finding an optimal integer solution, and in three cases, stopped before finding an integer feasible answer. Table 1's final column contains summary information from the attempted solution with CPLEX.

Table 1's contents indicate that a quite sizable gap still exists for most of these problems after a fairly lengthy amount of time. The only problems that were solved to optimality are very small. Heuristic approaches will therefore be used in this research.

Table 1
Problems given to CPLEX

Problem	Number of stages	Number of machines per stage	Number of jobs per stage	CPLEX results
1	2	1	6	Optimal solution found
2	2	1	30	Gap: 97.61%
3	2	1	30	Optimal solution found
4	2	1	29/29	Gap: 98.26%
5	2	1	99/95	Gap: infinite
6	4	1	6	Gap: 65.05%
7	4	1	30	Gap: 98.73%
8	8	1	6	Gap: 88.32%
9	8	1	30	Gap: infinite
10	2	5/10	30	Gap: 86.78%
11	2	7/8	28/30	Gap: 87.47%
12	2	3/7	95/91	Gap: infinite
13	4	1/10/10/7	30	Gap: 95.14%
14	8	10/3/7/9/3/1/9/10	30	Gap: 95.68%

4. Lower bounds

This section contains a theorem with three lower bounds for the problem P. The notation "$\min_{[k]}$" will be used to indicate the $(k+1)$st from the lowest value and $\min_{[0]} \equiv \min$. For example, given a list of values {2, 5, 7, 8, 9}, $\min_{[1]} = 5$.

Theorem. The following are lower bounds on any feasible solution to P:

$$\mathrm{LB}^{(1)} = \max_{i=1,\ldots,n} \left\{ \sum_{t \in S_i} \left(p_i^t + \min_{j=0,\ldots,n} s_{ji}^t\right) \right\}, \tag{13}$$

$$\mathrm{LB}^{(2)} = \max_{t=1,\ldots,g} \left\{ \min_{i \in S^t} \sum_{\tau=1}^{t-1} \left(p_i^\tau + \min_{j=0,\ldots,n} s_{ji}^\tau\right) + \frac{\sum_{i \in S^t} \left(p_i^t + \min_{j=0,\ldots,n} s_{ji}^t\right)}{m^t} + \min_{i \in S^t} \sum_{\tau=t+1}^{g} \left(p_i^\tau + \min_{j=0,\ldots,n} s_{ji}^\tau\right) + \frac{1}{m^t} \sum_{k=1}^{m^t-1} \left[\min_{i \in S^t[k]} \sum_{\tau=1}^{t-1} \left(p_i^\tau + \min_{j=0,\ldots,n} s_{ji}^\tau\right) - \min_{i \in S^t} \sum_{\tau=1}^{t-1} \left(p_i^\tau + \min_{j=0,\ldots,n} s_{ji}^\tau\right) \right] \right\}. \tag{14}$$

Proof

$\mathrm{LB}^{(1)}$: This is a job-based bound. Every job i must be processed at each stage and must also be setup, which requires at least the minimal amount of time required to setup job i. Solutions which are feasible to constraint sets (5) and (6) satisfy this condition.

$\mathrm{LB}^{(2)}$: This is a machine based bound. Every stage t needs time to process job 0 and then the preemptive processing and minimal setup time for the rest of the jobs. In addition to minimum setup and processing at each stage, we can add in the minimum time to get to the stage plus the minimum time to finish after the stage. Furthermore, we may be able to bound idle time for parallel machines at each stage waiting for the first available job. This yields the bound $\mathrm{LB}^{(3)}$ originally proposed in Kurz and Askin (2001) as an extension to Leon and Ramamoorthy (1997). The first term represents the minimum time required for a job to reach stage t. The second term assumes that jobs are processed preemptively at stage t. The third term represents the minimum time for a job to finish processing and setup on the stages after t. The final term requires the observation that the second machine at stage t does not begin processing until the second job arrives, and so on for all the machines at stage t. We find the minimum times for the second, third, etc. jobs to reach stage t and allocate this time to all the machines at this stage. Solutions which are feasible to constraint sets (5) and (6) satisfy this condition. □

5. Heuristics

The first heuristic is a naïve approach that simply assigns jobs to machines in a greedy fashion. The second expands on a multiple machine insertion heuristic used in previous work done on the single stage problem, in order to take advantage of the sequence-dependent nature of the setup times. The third is based on Johnson's Rule. The fourth is an application of the random keys genetic algorithm. Note that the second caters to setup aspects of the problem while the third derives from standard flow shops. No restrictions on the form of the resultant schedules is made.

Let [i] indicate the ith job in an ordered sequence in the following. In many of the following heuristics, a modified processing time is used. It is denoted by $\tilde{p}_i^t$ for job i in stage t and is defined as $\tilde{p}_i^t = p_i^t + \min_j s_{ji}^t$. This time represents the minimum time at a stage t that must elapse before job i could be completed.

5.1. SPT cyclic

This is a naïve greedy heuristic that assign jobs to machines with little or no regard for setup times or the interactions between stages. Because of its simplistic nature, it provides a basis of comparison for the other heuristics. In the SPT Cyclic Heuristic (SPTCH), the jobs are ordered at stage 1 in increasing order of the modified processing times $\tilde{p}_i^1$. At subsequent stages, jobs are assigned in earliest ready time order. Jobs are assigned to the machine in every stage that allows it to complete at the earliest time as measured in a greedy fashion.

1. Create the modified processing times $\tilde{p}_i^1$.
2. Order the jobs in non-decreasing order (SPT) of $\tilde{p}_i^1$.
3. At each stage $t = 1, \ldots, g$, assign job 0 to each machine in that stage.
4. For stage 1:
   a. Let $bestmc = 1$.
   b. For $[i] = 1$ to n, $i \in S^1$:
      For $mc = 1$ to m^1:
         Place job $[i]$ last on machine mc.
         Find the completion time of job $[i]$. If this time is less on mc than on $bestmc$,
            let $bestmc = mc$.
      Assign job $[i]$ to the last position on machine $bestmc$.
5. For each stage $t = 2, \ldots, g$:
   a. Update the ready times in stage t to be the completion times in stage $t - 1$.
   b. Arrange jobs in increasing order of ready times.
   c. Let $bestmc = 1$.
   d. For $[i] = 1$ to n, $i \in S^t$:
      For $mc = 1$ to m^t:
         Place job $[i]$ last on machine mc.
         Find the completion time of job $[i]$. If this time is less on mc than on $bestmc$,
            let $bestmc = mc$.
      Assign job $[i]$ to the last position on machine $bestmc$.

5.2. Flowtime Multiple Insertion Heuristic

The Flowtime Multiple Insertion Heuristic (FTMIH) is a multiple insertion heuristic to minimize the sum of flowtimes (completion-ready times) at each stage. It is a multiple machine, multiple stage adaptation of the Insertion Heuristic for the TSP. This multiple insertion heuristic with completion time criteria was found to be effective for the single stage case. Setup times are accounted for by integrating their values into the processing times using $\tilde{p}_i^t$. The Insertion Heuristic can then be performed using these modified processing times at each stage. Once jobs have been assigned to machines, the true processing and setup times can be used. The FTMIH has the following steps for each stage t:

1. Create the modified processing times $\tilde{p}_i^t$.
2. Order the jobs in non-increasing order (LPT) of $\tilde{p}_i^t$.
3. For $[i] = 1$ to n, $i \in S^t$:
   a. Insert job $[i]$ into every position on each machine.
   b. Calculate the true sum of flowtimes using the actual setup times.
   c. Place job i in the position on the machine with the lowest resultant sum of flowtimes.
4. Update the ready times in stage $t + 1$ to be the completion times in stage t.

5.3. The g/2, g/2 Johnson's Rule

Johnson's Rule (1954) finds the optimal makespan solution for $F2\|C_{\max}$. Variants have been created, for example by Campbell et al. (1970), for the flow shop with more than two stages. This heuristic is an extension of Johnson's Rule to take into account the setup times. The aggregated first half of the stages and the aggregated last half of the stages are considered to create the order for assignment in stage 1. The value $\tilde{p}_i^1$ is the sum of modified processing times for stages 1 to $\lfloor g/2 \rfloor$ and $\tilde{p}_i^g$ is the sum over stages $\lfloor g/2 \rfloor + 1$ to g.

1. Create the modified processing times $\tilde{p}_i^1$ and $\tilde{p}_i^g$.
2. Let $U = \{j|\tilde{p}_j^1 < \tilde{p}_j^g\}$ and $V = \{j|\tilde{p}_j^1 \geqslant \tilde{p}_j^g\}$.
3. Arrange jobs in U in non-decreasing order of $\tilde{p}_i^1$ and arrange jobs in V in non-increasing order of $\tilde{p}_i^g$. Append the ordered list V to the end of U.
4. At each stage $t = 1, \ldots, g$, assign job 0 to each machine in that stage.
5. For $[i] = 1$ to n, $i \in S^1$:
   a. For $mc = 1$ to m^1:
      Place job $[i]$ last on machine mc.
      If this placement results in the lowest completion time for job $[i]$, let $m = mc$.
   b. Place job $[i]$ last on machine m.
6. For each stage $t = 2, \ldots, g$:
   a. Update the ready times in stage t to be the completion times in stage $t - 1$.
   b. Arrange jobs in increasing order of ready times.
   c. For $[i] = 1$ to n, $i \in S^t$:
      (1) For $mc = 1$ to m^t:
         Place job $[i]$ last on machine mc.
         If this placement results in the lowest completion time for job $[i]$, let $m = mc$.
      (2) Place job $[i]$ last on machine m.

5.4. Random keys genetic algorithm

RKGA differs from traditional genetic algorithms most notably in the solution representation. Random numbers serve as sort keys in order to decode the solution. The decoded solution is evaluated with a fitness function that is appropriate for the problem at hand. For example, Norman and Bean (1999) suggest using the following solution representation for an identical multiple machine problem. Each job is assigned a real number whose integer part is the machine number to which the job is assigned and whose fractional part is used to sort the jobs assigned to each machine. Once the job assignments and order on each machine are found through the decoding, a schedule can be built incorporating additional factors such as non-zero ready times and sequence-dependent setup times. The desired performance measure can then be found using the schedule. In this research, this representation is used for the jobs in the first stage. The assignment of jobs to machines in subsequent stages follows the method used in SPTCH and the Johnson's Rule Based Heuristics, where each job is assigned to the machine that allows it to complete at the earliest time as measured in a greedy fashion.

The genetic operators and related parameters used in this research are based on those in Bean (1994). Each generation has a population of 100 chromosomes. The initial population is generated randomly. An elitist strategy is used for reproduction. Each chromosome is decoded and the resulting schedule is evaluated for the makespan. Chromosomes with lower makespans are more desirable, so 20% of the chromosomes with the lowest makespan values are automatically copied to the next generation. Parametrized uniform crossover is used to select 79% of the chromosomes in the next generation. For each chromosome in the next generation, the following is performed. Two chromosomes in the current generation are selected

	Job	1	2	3	4
(a)	Parent 1	1.23	2.03	1.45	2.89
	Parent 2	2.15	2.45	1.85	1.03
(b)	Crossover	0.45	0.23	0.68	0.75
(c)	Child	1.23	2.03	1.45	1.03

Fig. 1. Parametrized uniform crossover example.

at random. For each job, a random number is generated. If the value is less than 0.7 (following Bean, 1994), the value from the "first" chromosome is copied to the new chromosome, otherwise the value from the "second" chromosome is selected. The remaining 1% of the next generation is filled through "immigration", in which new chromosomes are randomly generated. The above procedures are repeated until we are fairly sure that the population has settled into a good location in the search space. In this research, we continue until 100 generations have been examined without finding an improved schedule. This value was selected empirically.

A small example of the parametrized uniform crossover technique is provided in Fig. 1 to illustrate the structure of the chromosome as well as the parametrized uniform crossover technique itself. Consider a 4 job, single stage, 2 machine problem. Two parent chromosomes have been selected in Fig. 1(a). The first chromosome tells us that there is a schedule where machine 1 has jobs 1 and 3, in that order and machine 2 has jobs 2 and 4, in that order. The second parent chromosome tells us that there is a schedule where machine 1 has jobs 4 and 3, in that order and machine 2 has jobs 1 and 2, in that order. This information, when combined with the other problem data such as processing time, ready times, etc, is used to determine when each job completes processing on the machines for each schedule. The parametrized uniform crossover technique requires four random numbers to be generated, as shown in Fig. 1(b). The first three random numbers tell us to copy the genes from Parent 1 to the resulting child and the last random number tells us to copy the genes from Parent 2 to the resulting child. The resulting child is shown in Fig. 1(c). The child chromosome is decoded to tell us that there is a schedule where machine 1 has jobs 4, 1 and 3, in that order and machine 2 has job 2 only.

6. Generation of test data

An experiment was conducted to test the performance of the heuristics. Integer data was generated. Data required for a problem consist of the number of jobs, range of processing times, number of stages and whether all stages have the same number of machines or not. Each stage requires data defining how many machines exist at that stage, the sequence dependent setup times, the processing times and the ready times. The ready times for stage 1 are set to 0 for all jobs. The ready times at stage $t + 1$ are the completion times at stage t, so this data need not be generated. Processing times are distributed uniformly over two ranges with a mean of 60:[50–70] and [20–100]. Flexible flow lines are considered by allowing some jobs to skip some stages. Following Leon and Ramamoorthy (1997), the probability of skipping a stage is set at 0, 0.05, or 0.40. The setup times are uniformly distributed from 12 to 24 which is 20% to 40% of the mean of the processing time. The setup time matrices are asymmetric and satisfy the triangle inequality. The setup time characteristics follow Rios-Mercado and Bard (1998).

The problem data can be characterized by six factors: the probability that a job skips a stage, range of processing times, number of stages, whether the number of machines per stage is constant or variable, range

Table 2
Factor levels

Factor	Levels	
Skipping probability	0.00 0.05 0.40	
Processing times	Unif (50–70) Unif (20–100)	
Number of stages	2 4 8	
Machine distribution	Constant Variable	
	Depends on machine distribution	
	Constant	Variable
Number of machines	1 2 10	Unif(1,4) Unif(1,10)
Number of jobs	6 30 100	

in number of machines per stage and number of jobs. Each of these factors can have at least two levels. These levels are shown in Table 2.

In general, all combinations of these levels will be tested. However, some further restrictions are introduced. The variable machine distribution factor requires that at least one stage have a different number of machines than the others. Also, the largest number of machines in a stage must be less than the number of jobs. Thus, the combination with 10 machines at each stage and 6 jobs will be skipped and the combination of 1–10 machines per stage with 6 jobs will be changed to 1–6 machines per stage with 6 jobs. There are 252 test scenarios and ten data sets are generated for each one.

7. Experimental results

This section discusses the effectiveness of the proposed lower bounds and the proposed construction heuristics. The heuristics were implemented in C, compiled with Microsoft Visual C++ and run on a PC with a Pentium III 800 MHz processor with 512 MB of RAM. "Loss" is the (makespan − lower bound)/lower bound. The best lower bound was used for each problem. The running times were found using the clock() function.

7.1. Comparing heuristics

Every heuristic considered here was run on the same 2520 data sets. RKGA was run 16 times and the minimum and average loss over the 16 runs were found for each of the 2520 data sets. Summary statistics over all the data sets are presented in Table 3. RKGA achieves the lowest values for the loss statistics and finds the minimum loss schedules many more times than the other heuristics. The variation seen within the 16 RKGA runs will be discussed later. A single factor ANOVA for the algorithm (ALGO$) was performed,

Table 3
Loss statistics for heuristics

Heuristic	Loss			Number of times
	Average	Standard deviation	Maximum	Minimum[a]
SPTCH	0.25	0.16	0.98	64
FTMIH	0.24	0.15	0.94	180
$g/2$, $g/2$ Johnson's	0.21	0.13	0.79	123
RKGA	0.16	0.11	0.60	2386

[a] Using the best of the 16 RKGA runs.

Table 4
ANOVA results

Source	Sum of squares	df	Mean square	F-ratio	P
ALGO	13.682	3	4.561	228.033	0.000
Error	201.497	10076	0.020		

and the results are shown in Table 4. These results indicate that there is at least one heuristic that is different in mean response. This motivated the use of Fisher's least significant difference method (Montogomery and Runger, 2003), with the results summarized in Table 5. These values indicate that RKGA is preferred with 99% confidence.

SPTCH and the $g/2$, $g/2$ Johnson's Based Rule never required more than 0.01 seconds of CPU time in these experiments. FTMIH required up to 25 seconds and RKGA required more time before stopping for every single one of the 2520 problem instances, up to 380 seconds for some larger problems. The results of the Fisher's least significant difference method indicates that RKGA is better than the $g/2$, $g/2$ Johnson's Based Rule in general, but the larger running time of RKGA induces the question of when RKGA should not be used, and instead perhaps the $g/2$, $g/2$ Johnson's Based Rule should be used. The low running time of the heuristics other than RKGA certainly indicate that perhaps they could be run in the initialization step of RKGA to provide three potentially good solutions to the first generation.

RKGA generally dominates the $g/2$, $g/2$ Johnson's Based Rule in loss performance, with the $g/2$, $g/2$ Johnson's Based Rule only outperforming the best of 16 runs of RKGA 15 of the 2520 problem instances. These fifteen instances represent six different configurations, as shown in Table 6. Every one of these fifteen instances involved problems with exactly 10 machines per stage, eleven involved problems with two stages and twelve involved problems with the smaller range of processing times. This is reasonable because this kind of problem somewhat matches the scenario for which Johnson's Rule was designed. In this case, we have a two stage flow line and each job must visit each stage. However, instead of only one machine for each stage, we have ten. This means that a tie in Johnson's order does not force a job that should be first on

Table 5
Fisher's least significant difference method

Heuristics	Difference of means	Significant difference at 99% level? (Yes if difference > 0.01)
SPTCH vs RKGA	0.09	Yes
SPTCH vs $g/2$, $g/2$	0.04	Yes
SPTCH vs FTMIH	0.01	No
FTMIH vs RKGA	0.08	Yes
FTMIH vs $g/2$, $g/2$	0.03	Yes
$g/2$, $g/2$ vs RKGA	0.05	Yes

Table 6
Problems where $g/2$, $g/2$ Johnson's based rule outperforms RKGA

Skipping probability	Processing time range	Number of stages	Number of machines per stage	Number of jobs	Number of files
0.00	50–70	2	10	30	1
0.00	50–70	2	10	100	5
0.00	20–100	2	10	30	2
0.00	20–100	2	10	100	1
0.05	50–70	2	10	100	2
0.05	50–70	4	10	100	4

a machine to be second; the job can be first on another machine at that stage. In this way, we allow several jobs that are early in the Johnson's order to move towards the beginning of the schedule on the first stage. In the second stage, by considering jobs in ready time order and allowing jobs to be placed on the machine on which it ends soonest, we continue with the Johnson's paradigm. We allow jobs that arrive to the second stage and will move through it quickly (before later jobs from stage 1) to be scheduled earlier. These jobs should not impact the overall makespan much because the later jobs in stage 1 will not arrive in stage 2 very soon regardless. The low range of processing times and the fact that all jobs visit both stages indicates that the stages are fairly balanced in terms of workload.

7.2. Discussion of RKGA runs

It has been noted before that 16 RKGA runs were made of the 2520 problem instances, with an average loss (that is, relative deviation from the lower bound) of 0.16. Since RKGA is a stochastic method, meaning that each run may not provide the same answer, this section intends to discuss the variation seen in the 16 RKGA runs. The average loss of 16 runs varies from 0, meaning all 16 runs attained the lower bound, to 0.5962, meaning the average deviation from the lower bound for that problem instance was 59.62%. The values of the best RKGA run in the set of 16 varies from 0 to 0.5759. The standard deviation of the 16 RKGA runs has been calculated for each problem instance as well and varies from 0 to 0.0792. The width of a 99% confidence interval on the mean loss, with the variance unknown, has been calculated for each problem instance as well, and varies from 0 to 0.1119. Fig. 2 plots the loss values of the best vs worst RKGA runs. The points plotted exactly on the $x = y$ line correspond to the runs with 0 variance. The graph

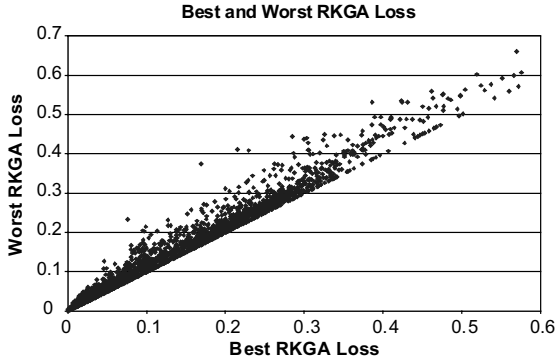

Fig. 2. Best vs worst RKGA performance over 16 runs.

illustrates that the RKGA is settling into a fairly consistent solution value, though that value sometimes appears to be far from the lower bound.

7.3. Quality of the lower bounds

Two lower bounds were presented in a previous section. Recall that $LB^{(1)}$ is the job based bound, giving the maximum time over all jobs to process and minimally setup each job in all stages. $LB^{(2)}$ is the machine based bound, giving the maximum time over all stages to preemptively process and minimally setup all jobs in that stage, plus the time to reach each stage. While $LB^{(2)}$ is the larger 2250 out of 2520 times, $LB^{(1)}$ is larger 270 times. Since $LB^{(1)}$ is easy to compute, it should be included even though it is not as good a lower bound in general as $LB^{(2)}$.

It is noted that RKGA occasionally found solutions very close to the lower bound $LB^{(2)}$. Of the 2520 test files, the best RKGA solution (of the 16 runs) was equivalent to the lower bound 3 times and within 1% of the lower bound 29 additional times. Of the 40320 individual RKGA runs, the lower bound was hit 44 times and RKGA was within 1% of the lower bound 344 additional times.

A comparison between the lower bound and optimal solution of each of these problems could be made, if these problems were easily solved to proven optimality. However, we have discussed this difficulty earlier. We compare the value of $LB^{(2)}$ for the 14 specific problems discussed earlier to the best value found by our heuristics (always by RKGA in these cases) and to the values returned by CPLEX, when CPLEX was given a two hour CPU time limit. These comparisons are summarized in Table 7.

Recall that problems 1 and 3 were solved to optimality within this time limit and no integer feasible solutions to problems 5, 9 and 12 were found within this time limit. Because the RKGA runs and the CPLEX runs were performed on different machine types, it seems difficult to compare the running times. However, we note that the longest running time of any of the RKGA runs (not just the 14 discussed here) on a 800 MHz was about 380 seconds. Whenever CPLEX was unable to solve the problem to optimality, it provided a current MIP best bound, which was always much lower than the value of $LB^{(2)}$. In the two cases where CPLEX did solve the problem to optimality, the current MIP best bound was higher than $LB^{(2)}$, but not much higher. Of course, having $LB^{(2)}$ be the same as the optimal solution is ideal, but not always attainable. This as well as the discussion above regarding the performance of RKGA lead us to conclude that $LB^{(2)}$ is an effective lower bound.

Table 7
Effectiveness of $LB^{(2)}$

Problem	$LB^{(2)}$	Best heuristic solution value	Current MIP best bound	Best integer feasible solution
1	515	522	521.95	522
2	2222	2272	89.00	3721
3	533	543	542.95	543
4	1935	2191	604.93	3485
5	6874	7236	499.39	None
6	669	708	363.51	1040
7	2384	2489	1082.78	8508
8	980	1040	293.00	2509
9	2689	2876	192.16	None
10	519	544	113.00	855
11	384	407	109.00	870
12	2056	2407	60.99	None
13	2355	2384	149.00	3065
14	2640	2715	247.01	5720

Note: Problems (1) and (3) were solved to optimality.

8. Conclusions and future work

This paper has examined four heuristics to find schedules minimizing makespan in flexible flow lines with sequence-dependent setups. These methods included a simplistic greedy method, approaches based on the TSP nature of the problem and the flow line nature of the problem and an application of the Random Keys Genetic Algorithm approaches. Lower bounds have been used in the evaluation of these heuristics. The lower bounds were investigated for performance and found to be efficient. The data characteristics investigated were designed to reflect characteristics used by other researchers, resulting in 252 types of data files, with 10 of each type generated. The heuristics were compared on 2520 data files. Through examination of the experimental results, it was determined that RKGA performed best on the problems examined here. However, in the specific situation of two stages, jobs that visit each stage and 10 machines per stage, the $g/2$, $g/2$ Johnson's Based Heuristic was found to be effective as well. Intuition has been provided to explain this result.

There are potentially unlimited opportunities for research in scheduling to minimize makespan in flexible flow lines with sequence-dependent setup times. In this paper, we have addressed only a few areas. The research in this paper has lead to many more questions regarding flexible flow lines with sequence-dependent setup times. For example, we wonder whether there a definition of a permutation schedule that is general enough to handle multiple machines in serial stages where not all jobs visit each stage and sequence-dependent setup times exist? Potential definitions may include a condition that cannot be violated, but does not tell explicitly what all the permutations may look like. For example, a permutation schedule may be one in which the relative ordering of start of setup times at each stage does not change from stage to stage. This definition in no way tells us what jobs should be assigned to what machine in stages with multiple machines and in fact allows for several alternative assignments at each stage. However, this definition also includes the traditional definition of a permutation schedule in a flow line. By creating a general permutation schedule definition, we may be able to find a class of schedules that contains the optimal makespan schedule for some special case, such as two stages with one machine at the first stage and two machines at the second, with sequence-dependent setup times at both.

We note that the integer programming formulation for the flexible flow line with sequence-dependent setup times has not been thoroughly investigated. It has been solved for some specific cases, but its performance on much larger problems is unknown. Work applying branch-and-bound, Lagrangean relaxation and alternative formulations of this problem are on-going.

The performance of the $g/2$, $g/2$ Johnson's Based Rule, coupled with the performance of a variant in Kurz and Askin (2003), indicates that perhaps this could be a basis for the heuristic space based search neighborhoods discussed in Storer et al. (1992). By considering several partitioning schemes on the g stages to create the two processing times, along the lines of Campbell et al. (1970), a space of the heuristics can be defined. (We thank an anonymous referee for this insight.)

Acknowledgements

This research was supported by the Engineering Research Program of the Office of Basic Energy Sciences at the Department of Energy and by the National Science Foundation under Grant DMII 99-00052.

References

Agnetis, A., Pacifici, A., Rossi, F., Lucertini, M., Nicoletti, S., Nicolo, F., Oriolo, G., Pacciarelli, D., Pesaro, E., 1997. Scheduling of flexible flow lines in an automobile assembly plant. European Journal of Operational Research 97, 348–362.

Bean, J.C., 1994. Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 6, 154–160.

Brah, S.A., Hunsucker, J.L., 1991. Branch and bound algorithm for the flow shop with multiple processors. European Journal of Operational Research 51, 88–99.

Brah, S.A., Loo, L.L., 1999. Heuristics for scheduling in a flow shop with multiple processors. European Journal of Operational Research 113, 113–122.

Campbell, H.G., Dudek, R.A., Smith, M.L., 1970. A heuristic algorithm for the *n* job, *m* machine sequencing problem. Management Science 16 (10), B630–B637.

Ding, F.-Y., Kittichatphayak, D., 1994. Heuristics for scheduling flexible flow lines. Computers and Industrial Engineering 26 (1), 27–34.

Gupta, J.N.D., 1997. A flowshop scheduling problem with two operations per job. International Journal of Production Research 35 (8), 2309–2325.

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor.

Johnson, S.M., 1954. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1, 61–67.

Kochhar, S., Morris, R.J.T., 1987. Heuristic methods for flexible flow line scheduling. Journal of Manufacturing Systems 6 (4), 299–314.

Kurz, M.E., Askin, R.G., 2001. Note on "An adaptable problem-space-based search method for flexible flow line scheduling". IIE Transactions 33, 691–693.

Kurz, M.E., Askin, R.G., 2003. Comparing scheduling rules for flexible flow lines. International Journal of Production Economics 85 (3), 371–388.

Lee, C.-Y., Vairaktarakis, G.L., 1994. Minimizing makespan in hybrid flowshops. Operations Research Letters 16, 149–158.

Lee, I., Sikora, R., Shaw, M.J., 1997. A genetic algorithm-based approach to flexible flow-line scheduling with variable lot sizes. IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 27 (1), 36–54.

Leon, V.J., Ramamoorthy, B., 1997. An adaptable problem-space-based search method for flexible flow line scheduling. IIE Transactions 29, 115–125.

Montogomery, D.C., Runger, G.C., 2003. Applied Statistics and Probability for Engineers. John Wiley, New York.

Norman, B.A., Bean, J.C., 1999. A genetic algorithm methodology for complex scheduling problems. Naval Research Logistics 46, 199–211.

Nowicki, E., Smutnicki, C., 1996. A fast tabu search algorithm for the permutation flow-shop problem. European Journal of Operational Research 91, 160–175.

Nowicki, E., Smutnicki, C., 1994. An approximation algorithm for a single-machine scheduling problem with release times and delivery times. Discrete Applied Mathematics 48, 69–79.

Nowicki, E., Smutnicki, C., 1998. The flow shop with parallel machines: A tabu search approach. European Journal of Operational Research 106, 226–253.

Oguz, C., Lin, B.M.T., Cheng, T.C.E., 1997. Two-stage flowshop scheduling with a common second-stage machine. Computers and Operations Research 24 (12), 1169–1174.

Piramuthu, S., Raman, N., Shaw, M.J., 1994. Learning-based scheduling in a flexible manufacturing flow line. IEEE Transactions on Engineering Management 41 (2), 172–182.

Rajendran, C., Chaudhuri, D., 1992. Scheduling in *n*-job, *m*-stage flowshop with parallel processors to minimize makespan. International Journal of Production Economics 27, 137–143.

Riane, F., Artiba, A., Elmaghraby, S.E., 1998. A hybrid three-stage flowshop problem: Efficient heuristics to minimize makespan. European Journal of Operational Research 109, 321–329.

Rios-Mercado, R.Z., Bard, J.F., 1998. Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. Computers and Operations Research. 25 (5), 351–366.

Rios-Mercado, R.Z., Bard, J.F., 1999a. An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup times. Journal of Heuristics 5, 53–70.

Rios-Mercado, R.Z., Bard, J.F., 1999b. A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. IIE Transactions 31, 721–731.

Salvador, M.S., 1973. A solution to a special case of flow shop scheduling problems. In: Elmaghraby, S.E. (Ed.), Symposium on the Theory of Scheduling and Its Applications. Springer-Verlag, Berlin, pp. 83–91.

Santos, D.L., Hunsucker, J.L., Deal, D.E., 1996. An evaluation of sequencing heuristics in flow shops with multiple processors. Computers and Industrial Engineering 30 (4), 681–692.

Santos, D.L., Hunsucker, J.L., Deal, D.E., 1995. FLOWMULT: Permutation sequences for flow shops with multiple processors. Journal of Information and Optimization Sciences 16 (2), 351–366.

Sawik, T.J., 1992. A scheduling algorithm for flexible flow lines with limited intermediate buffers. Proceedings of the 8th International Conference on CAD/CAM, Robotics and Factories of the Future, vol. 2. pp. 1711–1722.

Sawik, T.J., 1994. New algorithms for scheduling flexible flow lines. Proceedings of the 1994 Japan-USA Symposium on Flexible Automation, vol. 3. pp. 1091–1094.

Sawik, T.J., 1995. Scheduling flexible flow lines with no in-process buffers. International Journal of Production Research 33 (5), 1357–1367.

Snyder, L.V., Daskin, M.S., 2001. A random-key genetic algorithm for the generalized traveling salesman problem. Working Paper, Department of Industrial Engineering and Management Sciences, Northwestern University.

Sriskandarajah, C., Sethi, S.P., 1989. Scheduling algorithms for flexible flowshops: Worst and average case performance. European Journal of Operational Research 43, 143–160.

Storer, R.H., Wu, S.D., Vaccari, R., 1992. New search spaces for sequencing problems with application to job shop scheduling. Management Science 38 (10), 1495–1509.

Wittrock, R., 1988. An adaptable scheduling algorithm for flexible flow lines. Operations Research 36 (3), 445–453.

Wittrock, R., 1985. Scheduling algorithms for flexible flow lines. IBM Journal of Research and Development 29 (24), 401–412.