Available online at www.sciencedirect.com

SCIENCE DIRECT®

Int. J. Production Economics 85 (2003) 371–388

international journal of production economics

www.elsevier.com/locate/dsw

Comparing scheduling rules for flexible flow lines

Mary E. Kurz[a,*,1], Ronald G. Askin[b,2]

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

Abstract

This paper explores scheduling flexible flow lines with sequence-dependent setup times. Three major types of heuristics are explored. Insertion heuristics (based on insertion heuristics for the traveling salesman problem) attempt to simultaneously equalize workload on all processors at a stage, and minimize total or single-stage flowtimes. Johnson's algorithm for two-stage flow shops and its heuristic extensions to m-machine flow shops are modified for parallel processors and the flexible flow-line environment. A set of naïve greedy heuristics is investigated for comparison purposes. The performance of the heuristics is compared on a set of test problems. Results indicate the range of conditions under which each method performs well.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Sequence-dependent setup times; Flexible flow lines; Heuristics; Scheduling; Makespan

1. Introduction

As manufacturing processes become more complex and factories hope to extend their capacity in terms of the raw number of jobs able to be processed and the number of different yet basically similar job types that can be processed, factories have moved away from the basic flow-line system. In a flow line, all jobs are processed by the same set of machines in a linear fashion, from the first to last stage and one machine performs all the processing for each stage. In order to extend the capacity of a single stage, additional parallel machines may be purchased. This extension of a flow line to allow multiple (usually identical) machines in stages transforms the flow line into a hybrid flow line. Capacity can again be extended when considering optional steps that some jobs may require and others may not. By removing the restriction that all jobs must visit all stages, we move from a (regular) flow line to a flexible flow line. In a flexible flow line, we still require jobs to move from the first to the last stage in order. Flexible flow lines occur in many different environments, including automobile manufacture (Agnetis et al., 1997) and printed circuit board manufacture (Piramuthu et al., 1994). We explicitly note the following system characteristics.

- There are g stages of processing which occur in a linear fashion from $1, 2, \ldots, g$.

*Corresponding author. Tel.: +864-656-4652; fax: +864-656-0795.
E-mail address: mkurz@clemson.edu (M.E. Kurz).
[1] Supported by the Engineering Research Program of the Office of Basic Energy Sciences at the Department of Energy.
[2] Supported by the National Science Foundation under Grant DMII 99-00057.

0925-5273/03/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0925-5273(03)00123-3

- Each of the n jobs visits the stages in this order, though all jobs need not visit all stages. Stages may be skipped for a particular job, but the process flow for each job is known in advance.
- There is no transportation time between stages.
- There are infinite buffers between all stages, so there is no blocking of machines.
- Each stage has a predetermined number of parallel identical machines. However, the number of machines can vary from stage to stage.
- There is no machine breakdown and the machines are continuously available.
- Each part is processed on at most one machine at each stage.
- The processing time for every job on every stage that it visits is known in advance and is constant.

Fig. 1 illustrates the physical relationship between machines and stages in an example flexible flow line. This flexible flow line has three stages. Stage 1 has one machine, stage 2 has three parallel identical machines and stage 3 has two parallel identical machines. The jobs flow from stage 1 to stage 3, but need not visit all stages. For a three-stage flexible flow line, there are seven types of jobs, based on what stages they visit. Jobs can follow one of the following routes, where commas separate the routes and the number indicates the stages visited: 1, 2, 3, 1–2, 1–3, 2–3, 1–2–3. Once we know that a job must visit stage 2, we must still decide which of the three machines will process the job.

The specific problem of interest in this work is finding a schedule for a given set of n jobs on a g-stage flexible flow line with sequence-dependent setup times to minimize the makespan. Sequence-dependent setup times are an additional characteristic beyond those differentiating a flexible flow line from a (regular) flow line. We consider that between the processing of two consecutive jobs on the same machine, some setup must be performed that depends on the ordering of these two jobs. This may occur in a painting operation, where different initial paint colors require different levels of cleaning when being followed by other paint colors. In addition, we assume that setup is non-anticipatory, meaning that we require the job for which setup is required to be available and the machine on which the job is to be processed to be idle. Sequence-dependent setup times are known for each pair of jobs on each stage and are constant.

We will use makespan as our scheduling criterion. Minimizing makespan is equivalent to minimizing the maximum completion time for all jobs. Each job completes processing on the last stage it visits at some time that we call its completion time. The largest of these times, considering all the jobs, is called the makespan. We distinguish makespan from flowtime in the following ways. Each job has a flowtime which is the difference between its completion time and its ready time (the time at which the job is available for processing on the first stage it visits, not the time at which it actually begins processing). A common flowtime-based criterion is total flowtime, the sum of the flowtimes for all jobs. In this criterion, changing any job's completion time changes the value of the objective, while in makespan, only one job defines the value of the objective. Makespan is a traditional objective for much of the flexible flow-line literature. In addition to its applicability to a periodic multiple product scheduling environment, the makespan criterion can also be considered as a surrogate for capacity maximization and flowtime criterion.

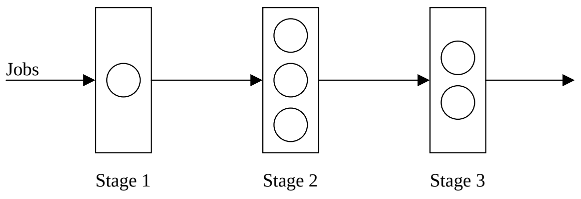

Fig. 1. 3 Stage flexible flow line.

Our desired output is a schedule of jobs on machines. We define a schedule as an assignment of jobs to machines at stages with the determination of the time setup begins for each job, the time job processing begins and the time job processing completes. We do not restrict the assignment of jobs to machines in stages beyond the fact that some jobs simply do not visit some stages. However, once a schedule is set for a problem by a heuristic, the schedule does not change.

The rest of the paper is organized as follows. Section 2 contains a literature review and a discussion of where this research fits in relation to the literature. Section 3 contains a detailed description of the problem. The proposed heuristics are described in Section 4. Lower bounds to assist in evaluating the performance of the proposed heuristics are presented in Section 5. The experimental design is described and the experimental results are reported in Section 6. Section 7 concludes the paper.

2. Literature review

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 use of branch-and-bound, extensions of previous flow-line techniques, applications of metaheuristics and development of new techniques.

Salvador (1973) first considered scheduling a hybrid flow line for makespan. However, there are no buffers between stages and no sequence-dependent setup times, in contrast to our problem. Branch-and-bound techniques are applied to determine the optimal permutation schedule in terms of makespan. A permutation of a set of n jobs is one of the $n!$ possible ways in which n jobs can be ordered. In Salvador's context, a permutation schedule requires that the jobs begin processing at the first stage in one of the permutation orders, and each job starts at such a time that it completes processing on each stage with no time spent in buffer. 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 for a (regular) flow line but restricted the resultant schedule to the set of permutation schedules. In the (regular) flow-line context, a permutation schedule requires all stages to process the jobs in the same order. Santos et al. (1995) combined permutation schedules with the FIFO queuing discipline at all stages after the first stage. In this context, 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.

Extending heuristics developed for the (regular) flow line, such as Johnson's Rule (Johnson, 1954) has been considered by numerous authors. Recall that Johnson's Rule basically tells us that, for a two-stage flow line, an optimal schedule can be found by putting the jobs that can quickly move through stage 1 first and putting the jobs that can move quickly through stage 2 last in a permutation schedule. 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 differences 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 (regular) flow lines and a branch-and-bound heuristic. Santos et al. (1996) also considered the use of heuristics developed for the (regular) 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 (regular) 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 schedules, they considered neighborhoods of problem data, using ideas from Storer et al. (1992). Problem data neighborhoods are created by changing some 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. In Lee et al. (1997), each job is comprised of splittable lots and therefore has a processing time which is not known until the lot size is determined, distinguishing their situation from ours. Though this research included sequence-dependent setup times, buffers between stages were limited and a permutation schedule was required. 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 in 1985 and 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 has developed numerous results for the flexible flow-line scheduling problem. His 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 (Sawik, 1992) 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 extended the RITM heuristic to the case of no buffers between stages (Sawik, 1994, 1995).

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.

3. Problem description

In describing the problem at hand, we have the following definitions:

n = number of true jobs to be scheduled,
g = number of serial stages,
m^t = number of machines at stage t,
p_i^t = processing time for job i at stage t,
s_{ij}^t = setup time from job i to job j at stage t,
$S^t = i : p_i^t > 0$, the set of jobs that visit stage t,
$S_i = t : p_i^t > 0$, the set of stages visited by job i.

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. The initial state of the machines could be considered "empty and idle" with job 0 representing this state at the beginning of the schedule and job $n+1$ representing this state at the end of the schedule. The processing times of jobs 0 and $n+1$ are set at 0. The setup time from job 0 to other jobs is the time to move from the nominal job, and the setup time to job $n+1$ from the other jobs is the time to move to the tear down job. These jobs are included in order to deal with the sequence-dependent setup times at the beginning and end of the schedules. 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 $n \geqslant \max_{t=1,2,\ldots,g} \{m^t\}$. Job processing at each stage is performed independently except that a job's ready time at stage $t+1$ is its completion time at stage t. The objective function is to minimize the makespan z. To ensure that an optimal solution will always exist that utilizes all m^t machines at each stage, we assume setup times satisfy some sufficient condition, such as $s^t_{0j} \leqslant \min_{i \neq 0} (p^t_i + s^t_{ij}) \forall j$ or $p^t_j + s^t_{0j} \leqslant \sum_{i \neq j} (p^t_i + \min_k s^t_{ki})/m^t - 1$. These ensure that it is never advantageous to place only job 0 on a machine. Recall that we assume non-anticipatory setups.

4. Construction heuristics

Eight construction heuristics are considered here. The first class of heuristics (cyclic heuristics) is based on simplistic assignment of jobs to machines with little or no regard for the setup times. The second class of heuristics is based on the insertion heuristic for the travelling salesman problem (TSP) and can be considered an adaptation of a heuristic by Nawaz et al. (1983). The insertion heuristic for the TSP begins with a set of cities to be visited in order and a set of cities that have not yet been visited. The unvisited cities are sorted according to some criteria and then added to the current set of cities to be visited. The location of the new city is determined by comparing the increase in the time to visit all cities, when this current job is inserted in every possible location. For example, if a list of cities is [2, 5, 7], and city 10 is to be added, we compare the time to visit all cities in the following cases: [10, 2, 5, 7], [2, 10, 5, 7], [2, 5, 10, 7] and [2, 5, 7, 10]. We then fix city 10's location by choosing the option with the smallest time. The parallel is drawn between cities and the sequence-dependent travel time between cities in the TSP to jobs and the time for sequence-dependent setup and processing in the current environment. The makespan is analogous to the time to visit all cities. For minimizing makespan on parallel machines with sequence-dependent setup times, extending the insertion heuristic has been found to be effective (Kurz and Askin, 2001a). The extension basically included the fact that now, every location on every machine must be considered. The third class of heuristics is based on Johnson's Rule. Note that the second class caters to setup aspects of the problem while the third derives from standard flow shops.

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

4.1. Cyclic heuristics

4.1.1. Basic cyclic heuristic (CH)

The basic cyclic heuristic (CH) arranges jobs in numerical order and then assigns them to machines in a cyclic order. For example, if jobs 1, 2, 4 and 9 visit a stage with two machines, the first machine will have jobs 1 and 4 in that order and the second machine will have jobs 2 and 9. The procedure follows the steps:

1. Arrange jobs in numerical order from 1 to n
2. At each stage $t = 1, \ldots, g$, assign job 0 to each machine in that stage
3. For each stage $t = 1, \ldots, g$:

```
   (a) Let $mc = 1$
   (b) For $i = 1$ to $n$, $i \in S^t$:
       If $mc > m^t$, let $mc = 1$
       Place job $i$ last on machine $mc$
       Let $mc = mc + 1$
```

Though simplistic, CH has a time complexity of O(ng).

4.1.2. Ready time cyclic heuristic (RCH)

The ready time cyclic heuristic (RCH) is nearly identical to CH. However, in RCH the jobs are sequenced in increasing order of ready times at stages 2 to g before being placed on machines. Jobs are assigned to stage 1 in the same manner as in CH. If jobs arrive to stage 2 in the order 1, 4, 2, and 5, and stage 2 has two machines, RCH will assign stage 2 with machine 1 having jobs 1 and 2 in that order and machine 2 having jobs 4 and 5 in that order. The rule is myopic in the sense that it does not consider the impact of current scheduling decisions on future stages. The procedural steps for RCH are as follows:

```
1. Arrange jobs in numerical order from 1 to $n$
2. At each stage $t = 1,\ldots,g$, assign job 0 to each machine in that stage
3. For each stage $t = 1,\ldots,g$:
   (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 for use in step 3d
   (c) Let $mc = 1$
   (d) For $[i] = 1$ to $n$:
       If $mc > m^t$, let $mc = 1$
       Place job $[i]$ last on machine $mc$
       Let $mc = mc + 1$
```

RCH's time complexity depends on the complexity of the sorting routine in Step 3b. If a selection sort (O(n^2)) is used, the overall complexity is O($n^2g + ng$) = O(n^2g).

4.1.3. SPT cyclic (SPTCH)

In the shortest processing time cyclic heuristic (SPTCH), the jobs are ordered at stage 1 in increasing order of the modified processing times $\tilde{p}_i^1$ and then assigned to machines in stage 1 as in RCH. At subsequent stages, jobs are sorted in earliest ready time order, exactly as in RCH. However, jobs are assigned to the machine in every stage that allows it to complete at the earliest time. That is, each job has a completion time calculated for each machine at the current stage under the assumption that the job is placed last on that machine, after the jobs that have been previously placed on that machine. The job is permanently assigned to the machine on which it has the lowest completion time. SPTCH has a time complexity of O(n^2g). The SPT rule attempts to activate stage 2 processors as soon as possible.

4.2. Multiple insertion heuristics

4.2.1. Flowtime multiple insertion heuristic (FTMIH)

The flowtime multiple insertion heuristic (FTMIH) is a multiple insertion heuristic to minimize the sum of flowtimes at each stage. The flowtime for a job is calculated as the difference between its completion time and ready time within that stage. It is a multiple machine, multiple-stage adaptation of the insertion heuristic for TSP. Setup times are heuristically accounted for by adding the minimum value required to setup job i to job i's processing time using the modified processing time $\tilde{p}_i^t = p_i^t + \min_j s_{ji}^t$. 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 in order to evaluate the correct completion time of each job.

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 $\tilde{p}_i^t$ for use in step 3
3. For $[i]=1$ to n, $i \in S^t$:
   (a) consider inserting job $[i]$ into every position on each machine
   (b) calculate the sum of flowtimes for all jobs scheduled so far in the current stage using the actual setup times, considering only the time spent in this stage
   (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

Step 3a requires that each job $[i]$ be placed in $[i]-1+m^t$ positions for evaluation in step 3b. Each evaluation requires $[i]$ calculations. Step 3 requires $\sum_{[i]=1}^{n}([i]-1+m^t)([i]) = 1/6(n)(n+1)(2n+1) + (m^t-1)n(n+1)/2$ steps. Since this must be done for each stage and $n \geq m^t$, the overall complexity of FTMIH is $O(n^3 g)$ for a given limit on processors per stage.

4.2.2. *Completion time multiple insertion heuristic (CTMIH)*

The completion time multiple insertion heuristic (CTMIH) is a multiple insertion heuristic to minimize the sum of completion times at each stage. It differs from the FTMIH only in that the sum of the completion times in the current stage is considered, regardless of the values of the ready times in that stage. The overall complexity of CTMIH is $O(n^3 g)$ following the same reasoning as for FTMIH.

4.2.3. *Makespan multiple insertion heuristic (MMIH)*

The makespan multiple insertion heuristic (MMIH) is a multiple insertion heuristic to minimize the maximum completion time at each stage. It differs from the CTMIH in that only the maximum completion time (makespan) at each stage is considered. The overall complexity of MMIH is $O(n^3 g)$ following the same reasoning as for FTMIH.

4.3. *Johnson's Rule-based heuristics*

4.3.1. $(1, g)$ *Johnson's Rule-based heuristic*

This heuristic is an extension of Johnson's Rule to take into account the setup times. Only the first and last stages are considered to create the order for assignment in stage 1. In the case that $g=2$ (a two-stage flexible flow line), this is exactly Johnson's Rule except we use modified processing times to incorporate the effect of the sequence-dependent setups and this ordering is only used for stage 1. The intent is to start the jobs in an order that gets them through stage 1 quickly but considers that the last stage is important as well. Subsequent stages are sequenced in the same manner as in RCH.

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 $V = j|\tilde{p}_j^1 \geqslant \tilde{p}_j^g$. The set U is the set of jobs that move through stage 1 faster than they move through the last stage g. The set V is the set of jobs that move through stage g faster than they move through stage 1
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, creating an ordered list for use in step 5
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 $m^1$:
       Consider placing job $[i]$ last on machine $mc$ in stage 1
       If this placement results in the lowest stage 1 completion time for job $[i]$, let $m = mc$
   (b) Place job $[i]$ last on machine $m$ in stage 1
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 for use in step 6c
   (c) For $[i]=1$ to $n$, $i \in S^t$:
       (1) For $mc = 1$ to $m^t$:
           Consider placing 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 $m$
```

(1, g) Johnson's Rule has time complexity of O(n^2g) following the same reasoning as for SPTCH

4.3.2. ($g/2, g/2$) *Johnson's Rule-based heuristic*

This rule differs from (1, g) Johnson's Rule only in that $\tilde{p}_i^1$ is replaced by $\sum_{t=1}^{\lfloor g/2 \rfloor} \tilde{p}_i^t$, the sum of modified processing times for stages 1 to $\lfloor g/2 \rfloor$ and $\tilde{p}_i^g$ is replaced by $\sum_{t=\lfloor g/2 \rfloor+1}^{g} \tilde{p}_i^t$, the sum over stages $\lfloor g/2 \rfloor + 1$ to g. Again, if $g = 2$, this is exactly Johnson's Rule except we use modified processing times to incorporate the effect of the sequence-dependent setups and this ordering is only used for stage 1. However, in contrast to $(1, g)$ Johnson's Rule, this heuristic attempts to include the impacts of the processing time of the intermediate stages on the initial ordering for stage 1. $(g/2, g/2)$ Johnson's Rule has time complexity of O(n^2g) following the same reasoning as for (1, g) Johnson's Rule.

5. Experimental design and results

The heuristics described in the previous section were evaluated on a set of test problems. This section describes the parameters used to generate the data set, the method used to compare the performance of the heuristics and the results of the comparisons.

5.1. *Data generation*

Problem data will be characterized by six factors: 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 in number of machines per stage, and number of jobs. Each of these factors can have at least two levels. When coding the test cases, the letters "L", "M", "H" and "A" will be used to form a sextuple $\langle u, v, w, x, y, z \rangle$ in the order described above. The levels of these factors are shown in Table 1. The order in Table 1 is the same order used to name the test cases. We set the mean processing time to 60 and generate setup times that are uniformly distributed between 20% and 40% of the mean processing time (see, for instance, the problem generation methods in Rios-Mercado and Bard (1998)).

In general, all combinations of these levels were tested. However, some further restrictions are introduced. The variable machine distribution factor requires that at least one stage has 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 six jobs was skipped and the combination of 1–10 machines per stage with six jobs was changed to 1–6 machines per stage with six jobs.

Table 1
Factor levels

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

There are 342 test scenarios and 10 data sets were generated for each scenario. Each of the 10 data sets varies within the same general characteristics.

5.2. Experimental results

The eight construction heuristics were run on each of the 3420 data sets discussed previously. Each data set will have eight associated makespan values, one per heuristic, which may or may not be the same. In an ideal world, we could compare the makespans found by the heuristics to the optimal makespan. However, the size of the problems and the complexity of scheduling in this environment makes this impossible, at least for all the problems investigated. We compare the heuristics using "loss" as the figure of merit, where "loss" is defined as (makespan−lower bound)/lower bound which provides some sense of the relative quality of each heuristic solution. We use the same lower bound value for each problem instance and the value is independent of the schedule found and of the method used to find the schedule. The lower bound used is

$$\mathrm{LB} = \max(\mathrm{LB}^{(1)}, \mathrm{LB}^{(2)}),$$

where

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

Table 2
Summary of experimental results

Heuristic	Average loss	Standard deviation loss	Maximum loss	Number of times minimum
CH	0.33	0.23	1.54	72
RCH	0.27	0.17	1.10	146
SPTCH	0.25	0.16	1.23	256
FTMIH	0.23	0.15	1.00	1041
CTMIH	0.79	1.77	13.77	821
MMIH	0.44	0.36	2.24	503
$(1, g)$ Johnson's	0.21	0.13	0.85	918
$(g/2, g/2)$ Johnson's	0.20	0.12	0.82	1336

and

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

$\mathrm{LB}^{(1)}$ simply states that the makespan is at least as large as the longest time, across all jobs, for a job to be processed on all of its stages and be setup for processing on those stages, assuming the shortest setup possible in those stages. $\mathrm{LB}^{(2)}$ was originally proposed in Kurz and Askin (2001b) as an extension to Leon and Ramamoorthy (1997). It basically states that the makespan must be at least as long as the time it takes a stage to process all of the jobs that visit it plus the time that that stage is idle before jobs could possibly reach it and also after the last job leaves it but before it exits the system. A more detailed explanation is given in Appendix A. When applied to the data in Leon and Ramamoorthy, this bound was able to prove that their solutions for a flexible flow-line scheduling problem were optimal (Kurz and Askin, 2001b).

Table 2 indicates that the $(g/2, g/2)$ Johnson's heuristic performs best in these data sets in terms of lowest average loss and smallest standard deviation. The $(g/2, g/2)$ Johnson's heuristic also has the lowest maximum loss and the most number of times finding the schedule with the smallest loss. The CTMIH performs worst overall, with the highest average loss, largest standard deviation and highest maximum loss. As a class, the cyclic heuristics (CH, RCH, and SPTCH) perform well enough on average, though they simply do not find as many good solutions as the Johnson's Rule-based heuristics. As the data was not preprocessed to be sorted according to, for example, the modified processing time, the order used in CH can be viewed as arbitrary. Any solution that does not perform as well as CH can certainly be rejected. The multiple insertion heuristics have varying degrees of success. Interestingly, though our overall goal is to find the best makespan schedule, using the flowtime as the intermediate measure (as in FTMIH) performed much better. We suggest that this is because in the intermediate steps, we do not look ahead to see the impact of assigning certain jobs to certain machine. By trying to keep each job moving through the stages as rapidly as possible, we hedge our bets regarding which job will be the makespan-defining job. The Johnson's Rule-based heuristics perform quite similarly to each other, which is not at all surprising considering that in the case of two stages, the variants are actually identical. Figs. (1)–(5) show the average losses arranged by the number of machines per stage. For clarity in the graphs, only one representative of each type of heuristic will be included. In each class, the specific heuristic with the best average loss and largest number of times finding the minimum loss solution is chosen. The cyclic representative is SPTCH, the insertion representative is FTMIH, and the Johnson's Rule-based representative is the $(g/2, g/2)$ Johnson's heuristic.

The results from the variants within the class of cyclic heuristics indicates the value of information. Ordering assignments beyond stage 1 according to ready times, instead of simply at random, had a significant impact. Average loss dropped from 0.33 to 0.27, a 6% improvement relative to the lower bound. A similar reduction was observed in the standard deviation of performance. Average loss reduced an additional 2% by using SPT at stage 1 to activate stage 2 processors as soon as possible.

We discuss results for each level of stage structure in the following sections. The values of the fourth and fifth factor are used to group the data sets into these groups. For example, data sets with names $\langle \mathrm{u, v, x, L, L, z} \rangle$ have a single machine per stage and data sets with names $\langle \mathrm{u, v, x, H, H, z} \rangle$ have between 1 and 10 machines in each stage.

5.2.1. *Single machine per stage* $\langle u, v, x, L, L, z \rangle$

Fig. 2 shows the average loss for the pure flow-line case. In the case where every job visits every stage with low range of processing times (casenames are $\langle \mathrm{L, L, x, L, L, z} \rangle$, shown as LLxz in Fig. 1), the heuristics are nearly indistinguishable in performance. The performance of all worsen as the number of stages and number of jobs increase. However, as the range of processing times increases (casenames are $\langle \mathrm{L, H, x, L, L, z} \rangle$, shown as LHxz in Fig. 1), the differences between the heuristics becomes more apparent. The $g/2$, $g/2$ Johnson's heuristic performs better with 6 or 20 jobs in general and the FTMIH performs

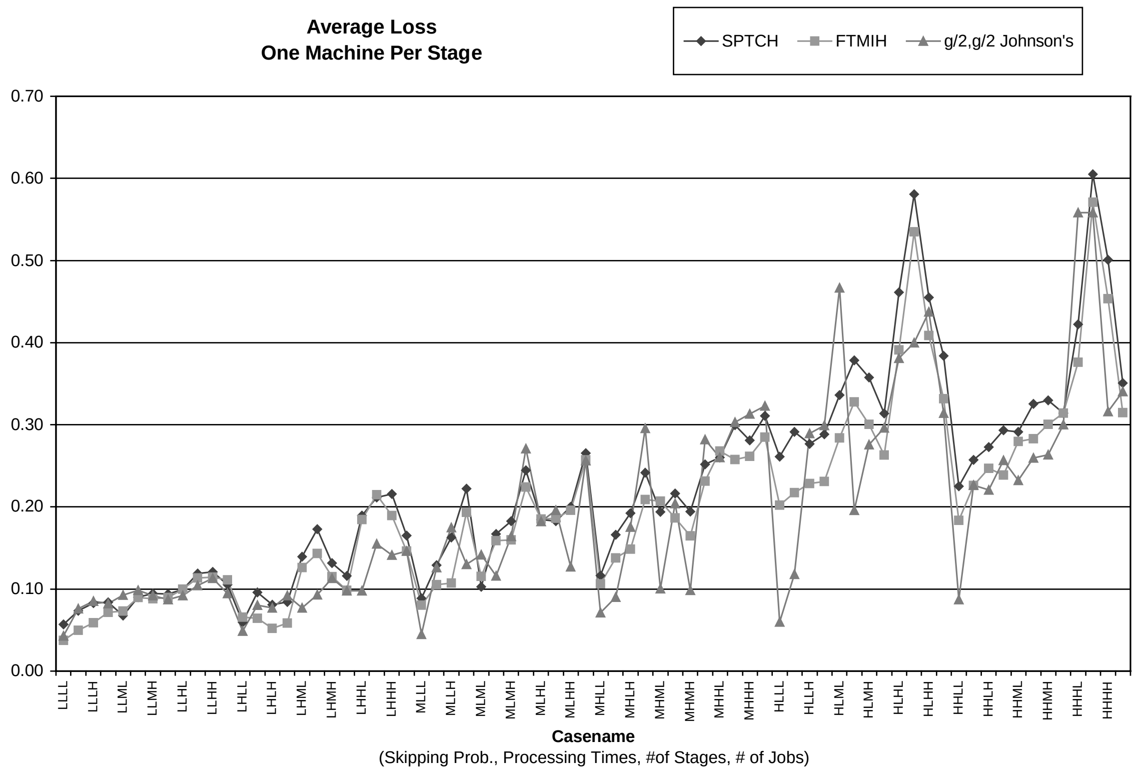

Fig. 2. Average loss with one machine per stage.

better with 30 or 100 jobs. The SPTCH does not perform competitively on average except when there are four stages and six jobs. Of the 720 tests in this category, SPTCH finds 47 lowest loss solutions, FTMIH finds 281 and the $g/2$, $g/2$ Johnson's heuristic finds 247. Note that FTMIH derives its motivation from the TSP. With one machine per stage, the flow line becomes a multistage TSP. Considering each stage in isolation, the TSP model (for which FTMIH uses a heuristic) would yield an optimal solution. The relevance of this analogy is diminished with multiple machines per stage. Moreover, the FTMIH heuristic evaluates more candidate solutions each time a job is to be scheduled. Thus, it should be expected to find a good placement more often than the other heuristics, especially as the number of jobs grows.

5.2.2. Two machines per stage $\langle u,v,x,L,M,z \rangle$

Fig. 3 shows the average loss for the case where every stage has two machines. Again, when the skipping probability is low and the processing time range is low, the heuristics performs basically the same. Clearly though, the FTMIH has a problem scheduling with two machines per stage and 100 jobs, even when the skipping probability is high, as evidenced by the occasional peaks in the graph. All the heuristics seem to have trouble in these cases, however, except when the skipping probability is low. The trend is that the higher the skipping probability, the more volatile the reaction to the other factors. Especially in the cases with the high probability of skipping a stage, low range of processing times and two stages ($\langle H,L,L,L,M,z \rangle$, shown in Fig. 2 as HLL*), the reaction of the $g/2$, $g/2$ Johnson's heuristic is the opposite of the other two. However, with 4 or 8 stages ($\langle H,L,M,L,M,z \rangle$ and $\langle H,L,H,L,M,z \rangle$, shown in Fig. 2 as HLM* and HLH*), the movement in the graph is the same for all three heuristics. Of the

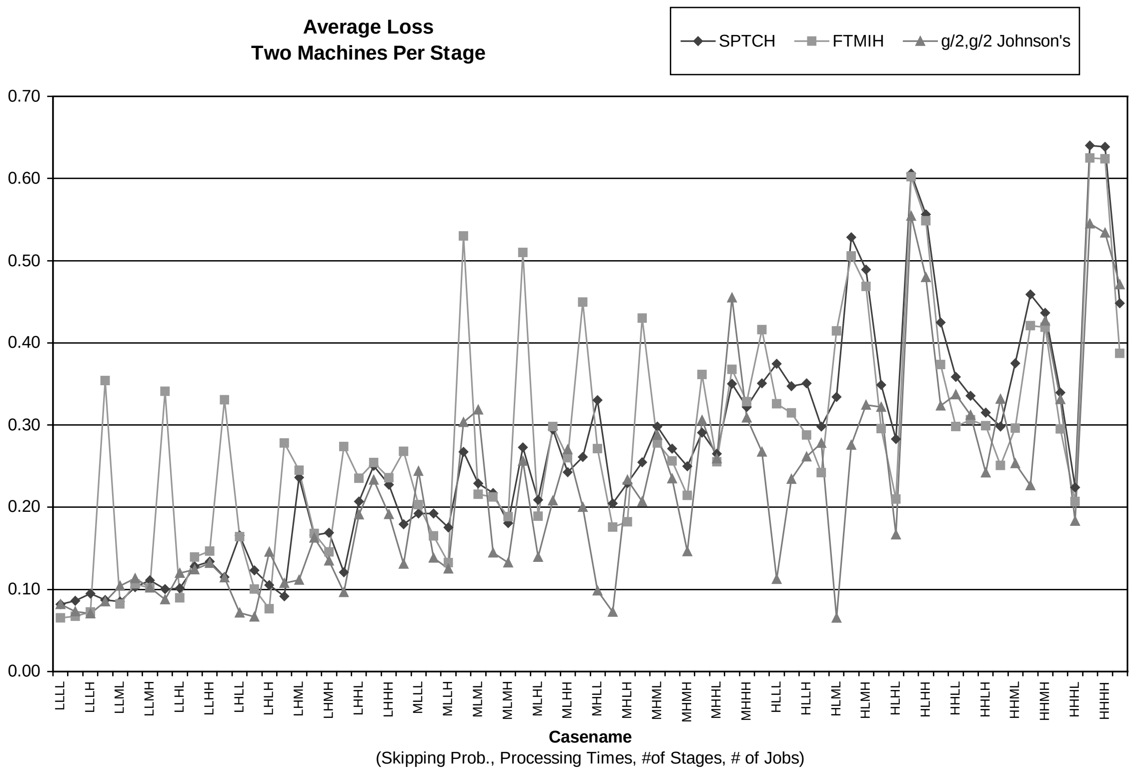

Fig. 3. Average loss with two machines per stage.

720 tests in this category, SPTCH finds 67 lowest loss solutions, FTMIH finds 162 and the $g/2$, $g/2$ Johnson's heuristic finds 351. In general, the $g/2$, $g/2$ Johnson's heuristic performs best in these cases.

5.2.3. *Ten machines per stage* $\langle u, v, x, L, H, z \rangle$

Fig. 4 shows the average loss for the case where every stage has 10 machines. The case with six jobs is not considered here, so the number of jobs factor (z) only takes on the M, H, and A levels. The performance of all the heuristics is highly dependent on the number of jobs to be scheduled, as evidenced by the repeating pattern of three seen throughout this graph. Except for a few cases where the average performance of FTMIH is better, the $g/2$, $g/2$ Johnson's heuristic has much better performance than the other two heuristics. SPTCH's performance is worst on average in most of the cases. Of the 540 tests in this category, SPTCH finds 26 lowest loss solutions, FTMIH finds 109 and the $g/2$, $g/2$ Johnson's heuristic finds 286.

5.2.4. *One to four machines per stage* $\langle u,v,x,H,L,z \rangle$

Fig. 5 shows the average loss for the case where every stage has between one and four machines. In this case, the patterns are not so evident. However, we can observe that SPTCH does not perform worst or best in most of these cases, falling somewhere in between. FTMIH and the $g/2$, $g/2$ Johnson's heuristic each have many cases in which one is better than the other but a pattern is not readily apparent. For instance, the $g/2$, $g/2$ Johnson's heuristic performs poorly with six jobs in $\langle$L, H, M, H, L, L$\rangle$, $\langle$L, H, H, H, L, L$\rangle$, and $\langle$M, L, M, H, L, L$\rangle$, but very well in cases $\langle$M, L, L, H, L, L$\rangle$, $\langle$M, L, H, H, L, L$\rangle$, $\langle$H, H, L, H, L, L$\rangle$ and $\langle$H, H, M, H, L, L$\rangle$. The Johnson's Rule-based heuristics perform better relatively with high skipping

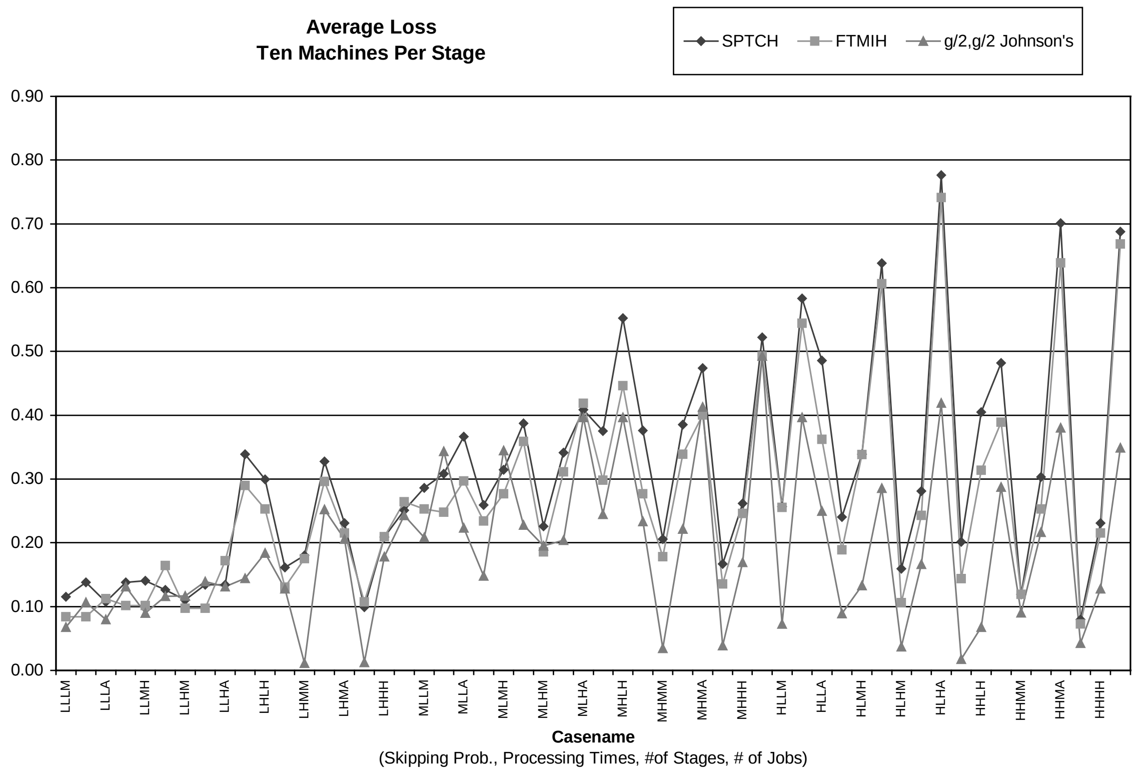

Fig. 4. Average loss with 10 machines per stage.

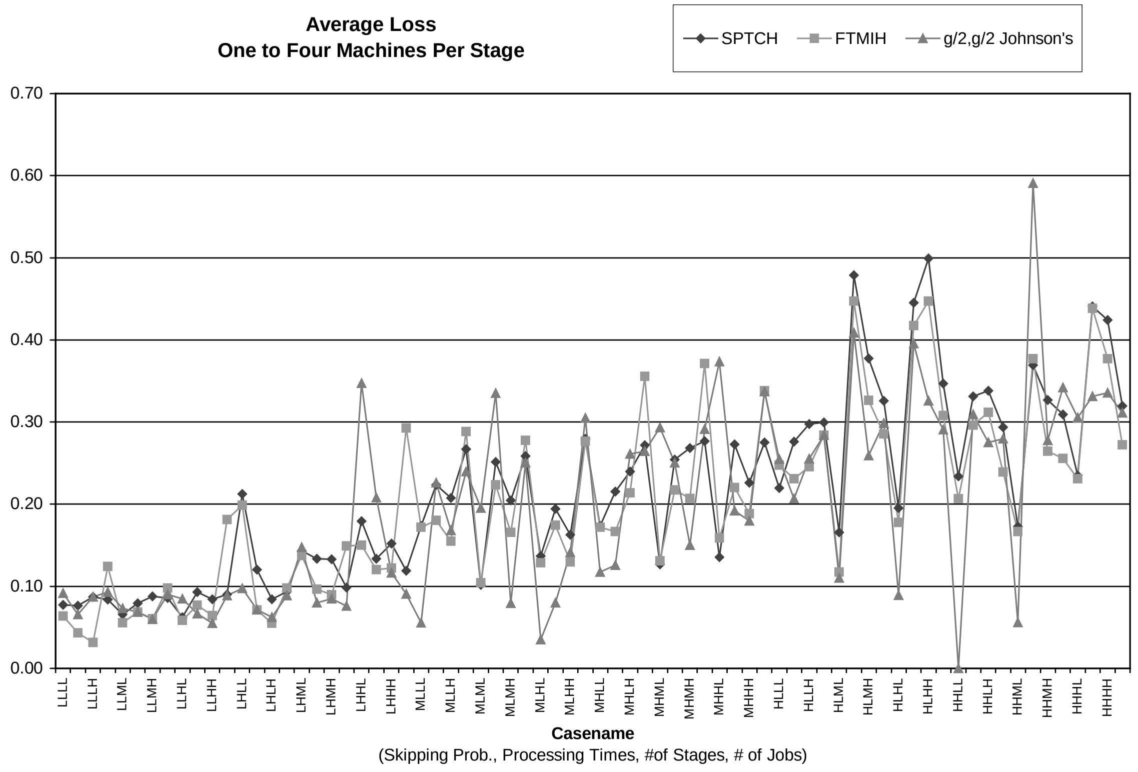

Fig. 5. Average loss with 1–4 machines per stage.

probabilities. One explanation is that the $g/2$, $g/2$ heuristic takes a global view of processing time for the complete job instead of emphasizing a myopic view encompassing only the next stage. The next stage numerically may not be relevant to a specific job, particularly when skipping probabilities are high. Of the 720 tests in this category, SPTCH finds 54 lowest loss solutions, FTMIH finds 254 and the $g/2$, $g/2$ Johnson's heuristic finds 221. The running time of the heuristics will be an important factor for this case.

5.2.5. *One to ten machines per stage* $\langle u,v,x,H,H,z \rangle$

Fig. 6 shows the average loss for the case where every stage has between one and 10 machines (six machines when there are only six jobs). These results are very similar to the last section. Of the 720 tests in this category, SPTCH finds 62 lowest loss solutions, FTMIH finds 235 and the $g/2$, $g/2$ Johnson's heuristic finds 231.

5.2.6. *Running times*

The heuristics were implemented in C on a PC with a Pentium II 200 MHz processor with 64 MB of RAM and compiled with Microsoft Visual C++. The running times were found using the clock() function and key statistics are summarized in Table 3. We include the time complexities and note that the running times are compatible with the time complexities.

Every heuristic provided some solutions in as little as 0.01 seconds. CH, RCH, SPTCH and the Johnson's Rule-based heuristics never had running times greater than 0.01 seconds, as can be seen in Table 3. The insertion heuristics had running times similar to each other. FTMIH has the highest average, minimum,

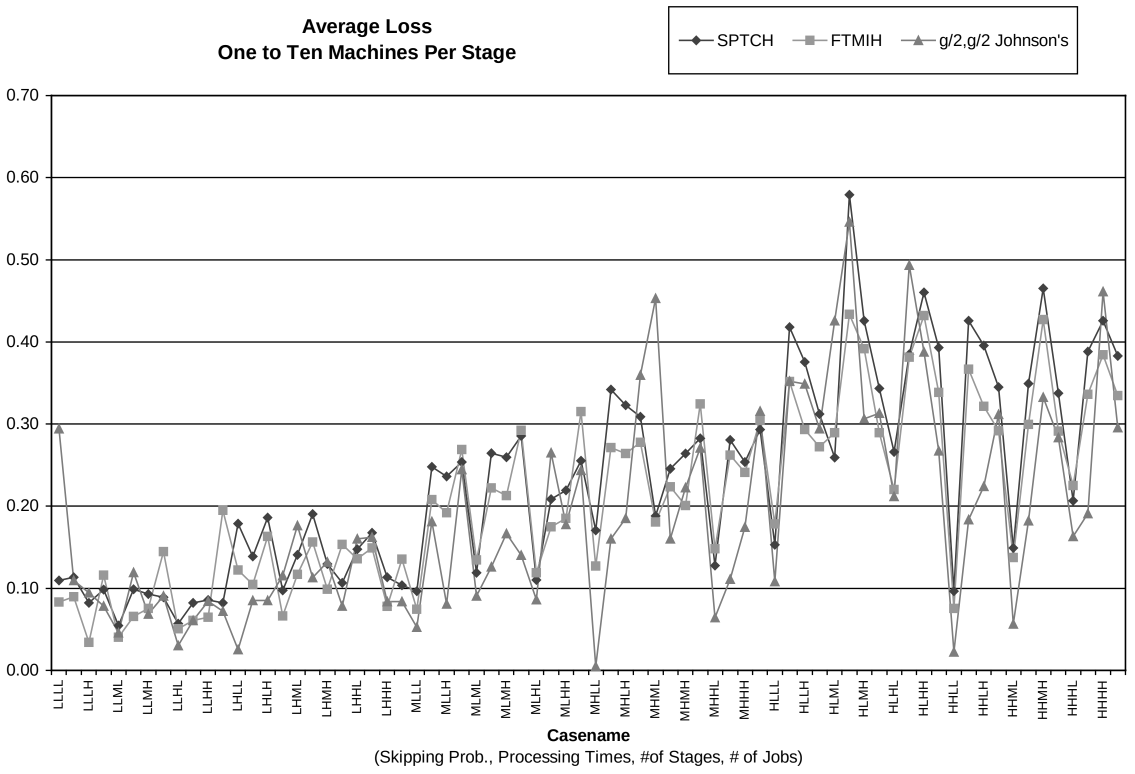

Fig. 6. Average loss with 1–10 machines per stage.

maximum and standard deviation in running time. The increase in running time does not seem to be accompanied by improved quality of solutions. Indeed, the Johnson's Rule-based heuristics perform much better despite extremely low running times. Whenever a choice must be made between the insertion heuristics and the Johnson's Rule-based heuristics, the lower running times of the Johnson's Rule-based heuristics should be considered, particularly when the problem characteristics would indicate that the insertion heuristics would perform competitively with the Johnson's Rule-based heuristics.

6. Conclusions

When decisions must be made in a real-time decision mode, the Johnson's Rule-based heuristics offer good alternatives. Though the insertion heuristics had been reasonably successful in the parallel machine scheduling application, they did not perform as well as less time consuming methods such as the Johnson's Rule-based heuristics in the multistage environment. Because of the low running times of the Johnson's Rule-based heuristics, it would be worth the time to try several combinations of two aggregated times to evaluate multiple schedules in this context or to find improved schemes for finding two values to use in the single machine and 1–4 and 1–10 machine cases. It should be noted that some factors empirically appeared to affect solution quality. In particular, the stage structure and number of machines per stage strongly affects the solution quality. When no jobs skip stages and the range of processing times is low, the heuristics appear to be nearly identical in performance, except when there are two machines per stage and 100 jobs.

Table 3
Running times in seconds

Heuristic	Average running time (s)	Minimum running time (s)	Maximum running time (s)	S.D. running time (s)	Complexity (assuming selection sort used)
CH	—	—	0.01	—	O(ng)
RCH	—	—	0.01	—	O(n^2g)
SPTCH	—	—	0.01	—	O(n^2g)
FTMIH	20.45	—	410.68	56.57	O(n^3g)
CTMIH	19.87	—	397.38	54.96	O(n^3g)
MMIH	19.82	—	397.05	54.87	O(n^3g)
1,g Johnson's Rule	—	—	0.01	—	O(n^2g)
$g/2$, $g/2$ Johnson's Rule	—	—	0.01	—	O(n^2g)

(—) less than 0.01 s.

The poor performance of FTMIH with two machines per stage and 100 jobs could be explained by the fact that two machines is an awkward compromise between one machine and many machines. With one machine, this heuristic will find a good location for each job. With 10 machines, this heuristic will find a good machine for each job, which may be a new machine. However, with only two machines, the first jobs do not have many options if the jobs already scheduled are poor matches for the current job to be scheduled. Since the jobs are ordered based on LPT at the stage, the first few jobs placed may have very discrepant ready times. In this case, the heuristic has no reason to choose one machine over the other and initial placements may be poor. With only two machines at the prior stage, ready times will be spread out and there is an advantage to matching job orderings from stage to stage. With 10 machines at the previous stage, job ready times will have a smaller variance making it easier for the heuristic to recognize the need to distributed jobs across processors early in the schedule construction process. This problem might be avoided by sequencing jobs based on ready times in lieu of pseudo-processing times following stage 1.

Appendix A

This appendix intends to describe LB$^{(2)}$, shown below. LB$^{(2)}$ was originally proposed in Kurz and Askin (2001b) as an extension to Leon and Ramamoorthy (1997):

$$\mathrm{LB}^{(2)} = \max_{t=1,\dots,g} \left\{ \begin{array}{l} \min_{i\in S^t} \sum_{\tau=1}^{t-1}(p_i^\tau + \min_{j=0,\dots,n} s_{ji}^\tau) + \dfrac{\sum_{i\in S^t}(p_i^t + \min_{j=0,\dots,n} s_{ji}^t)}{m^t} \\ + \min_{i\in S^t} \sum_{\tau=t+1}^{g}(p_i^\tau + \min_{j=0,\dots,n} s_{ji}^\tau) + \dfrac{1}{m^t}\sum_{k=1}^{m^t-1}\Big[\min_{i\in S^t[k]} \sum_{\tau=1}^{t-1}(p_i^\tau + \min_{j=0,\dots,n} s_{ji}^\tau) \\ - \min_{i\in S^t} \sum_{\tau=1}^{t-1}(p_i^\tau + \min_{j=0,\dots,n} s_{ji}^\tau)\Big] \end{array} \right\}.$$

LB$^{(2)}$ is stage-based, meaning that a value is derived for each stage, and the lower bound is the largest of each of the individual values. We will discuss the bound from the point of view of an arbitrary stage denoted t.

The basis of the bound is the second term which indicates that stage t must process and setup all of its jobs, requiring $\sum_{i\in S^t}(p_i^t + \min_{j=0,\dots,n} s_{ji}^t)$ amount of time. If this workload is spread evenly over all the machines in that stage, we will need to spend $\sum_{i\in S^t}(p_i^t + \min_{j=0,\dots,n} s_{ji}^t)/m^t$ at that stage. However, the machines at stage t will not begin processing until a job reaches that stage. The first term in LB$^{(2)}$ represents the earliest time at which stage t can become active, which is the time for the fastest job to get through stages 1 to $t-1$, or $\min_{i\in S^t} \sum_{\tau=1}^{t-1}(p_i^\tau + \min_{j=0,\ldots,n} s_{ji}^\tau)$.

Additionally, the stages after stage t must process at least one job after stage t has processed all of its jobs. The minimum amount of time required is represented by the third term, $\min_{i\in S^t} \sum_{\tau=t+1}^{g}(p_i^\tau + \min_{j=0,\ldots,n} s_{ji}^\tau)$.

Furthermore, we are able to bound the amount of time the parallel machines at stage t spend waiting for their first jobs. 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 $\{2, 5, 7, 8, 9\}$, $\min_{[1]} = 5$. The final term of $\mathrm{LB}^{(2)}$ represents the additional time each additional machine at stage t spends waiting for a job. The earliest time the second machine can become active is the arrival time of the second fastest job through stages 1 to $t-1$, the earliest time the third machine can become active is the arrival time of the third fastest job, and so on. These four concepts describe the terms used to derive the lower bound $\mathrm{LB}^{(2)}$.

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.

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.

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., 2001a. Heuristic scheduling of parallel machines with sequence-dependent set-up times. International Journal of Production Research 39 (16), 3747–3769.

Kurz, M.E., Askin, R.G., 2001b. Note on "An adaptable problem-space-based search method for flexible flow line scheduling". IIE Transactions 33 (8), 691–693.

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.

Nawaz, M., Enscore, E.E., Ham, I., 1983. A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem. OMEGA 11 (1), 91–95.

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

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.

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, Berlin, pp. 83–91.

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.

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.

Sawik, T.J., 1992. A scheduling algorithm for flexible flow lines with limited intermediate buffers. Proceedings of the Eighth International Conference on CAD/CAM, Robotics and Factories of the Future, Metz, France, 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, Kobe, Japan, 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.

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

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