International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: https://www.tandfonline.com/loi/tprs20

Heuristic scheduling of parallel machines with sequence-dependent set-up times

M. E. Kurz & R. G. Askin

To cite this article: M. E. Kurz & R. G. Askin (2001) Heuristic scheduling of parallel machines with sequence-dependent set-up times, International Journal of Production Research, 39:16, 3747-3769, DOI: 10.1080/00207540110064938

To link to this article: https://doi.org/10.1080/00207540110064938

Published online: 14 Nov 2010.

INT. J. PROD. RES., 2001, VOL. 39, NO. 16, 3747–3769

Heuristic scheduling of parallel machines with sequence-dependent set-up times

M. E. KURZ† and R. G. ASKIN†*

In many manufacturing environments, multiple processing stations are used in parallel to obtain adequate capacity. Likewise, in many production environments, set-up activities are required for switching between items. This work addresses scheduling in parallel machines with sequence dependent set-up times and possibly non-zero ready times with the goal of minimizing makespan. Non-zero ready times allow for application in a continuous planning environment and will also support the expansion of the current model to a multistage production environment. An integer programming formulation is presented. Several heuristics, including approaches based on MULTI-FIT, genetic algorithms and the travelling salesman problem, are then developed and compared empirically. Seven factors are identified in order to generate problem data, including the number of parallel machines, the average number of jobs per machine, set-up time distribution parameters and processing time distribution parameters. The set-up time matrix can be either symmetric or asymmetric but must satisfy the triangle inequality. A modified insertion heuristic is found to perform best for these types of problems.

1. Introduction

In many manufacturing environments, multiple processing stations are used in parallel to obtain adequate capacity. Likewise, in many production environments, set-up activities are required for switching between items. For instance, new fixtures or machine tooling may be required. This work addresses scheduling in parallel machines with sequence-dependent set-up times and possibly non-zero ready times. Section 2 of this paper contains a literature review, while section 3 contains an integer programming model. The proposed heuristics are described in section 4 and the experimental design is described in section 5. Section 6 contains bounds. Section 7 contains the experimental results and section 8 concludes.

2. Literature review

The problem of interest in this work is the $P|r_i, s_{ij}|C_{\max}$ problem. McNaughton (1959) provided an algorithm (as part of a constructive proof) to minimize makespan on a number of parallel identical machines in the case of independent jobs with pre-emption. Hu (1961) developed an algorithm to minimize the makespan for jobs with a tree precedence constraint relationship and equal processing times, although he did not allow pre-emption. An important result of Hu's work is a labelling algorithm that assists in partitioning the set of jobs in many later algorithms. Muntz and

Revision received January 2001.

† Department of Systems and Industrial Engineering. The University of Arizona, Tucson, AZ 85721-0020, USA.

* To whom correspondence should be addressed. e-mail: ron@sie.arizona.edu

Coffman generalized Hu's labelling algorithm. In Muntz and Coffman (1969), an unequal processing time version of Hu's labelling algorithm is combined with McNaughton's lower bound on the makespan for the case of two machines, arbitrary precedence constraints and pre-emption. Muntz and Coffman (1970) provided a generalization of Hu's labelling algorithm as one step in an optimal method under the conditions of an assembly tree precedence structure. Later, Monma and Potts (1989) determined that minimizing the makespan on two parallel machines with pre-emptions and sequence-independent set-up times is NP-hard. In their problem, jobs are divided into batches, which may be re-partitioned. Monma and Potts (1993) also developed heuristics in the case of sequence-dependent set-ups—in one of which the batches may be re-partitioned while the other builds on McNaughton's results. However, the set-up times are only somewhat sequence-dependent; set-ups only occur when switching from jobs in one batch to jobs in another batch, and the time is the same regardless of whichever batch is next. Thus, from a job point of view, set-ups are sequence-dependent with two possible values, while from a batch point of view, set-ups are sequence independent.

The more relevant case of forbidding pre-emptions will be discussed in more detail here. Graham (1969) developed the well-known Longest Processing Time (LPT) heuristic and showed it to have a worst case bound of

$$\frac{\omega_L}{\omega_O} \leq \frac{4}{3} - \frac{1}{3m},$$

where m is the number of machines, ω_L is the LPT makespan and ω_O is the optimal schedule length. Coffman and Sethi (1976) improved this bound to

$$\frac{\omega_L}{\omega_O} \leq \frac{k+1}{k} - \frac{1}{km},$$

where $1 \leq k \leq |P_i|$, and $|P_i|$ is the number of jobs on machine i in the LPT schedule, $1 \leq i \leq m$. Sahni (1976) and Babel *et al.* (1998) interpreted the $P||C_{\max}$ problem as an m-partition of the jobs such that the largest partition size, measured by the summed processing times of the jobs in that partition, is minimized. MULTI-FIT—a widely utilized solution approach to this problem—was developed by Coffman *et al.* (1978). MULTI-FIT packs equal-sized bins, efficiently searching over bin sizes. Coffman *et al.* also found bounds for MULTI-FIT that were improved upon by Friesen (1984). Lee and Massey (1988) noted the strengths of both the LPT and MULTI-FIT heuristics and suggested combining them, using LPT to provide an initial solution and then MULTI-FIT as an improvement method. Blocher and Chand (1991) also combined two approaches to this problem in order to realize a solution within a desired percentage of optimal bounds, as well as developing improved bounds on the LPT heuristic. Guignard (1993) used Lagrangean decomposition to discover plant location and constrained 0–1 knapsack problems within the $P||C_{\max}$ problem. Punnen and Aneja (1995) developed lower bounds for the general minmax combinatorial problem, of which $P||C_{\max}$ is an application. Recently, genetic algorithms have been applied to this problem (for example Hou *et al.* 1994, Corrêa *et al.* 1999). In these papers, a schedule is represented by a set of strings such that each machine has a string. The string then contains the jobs assigned to that machine, in the order to be processed. Min and Cheng (1998) combined genetic algorithms and simulated annealing for the $P||C_{\max}$ problem and found that combining the two balanced the better solutions of the genetic algorithm with longer running times of the simulated annealing algorithm.

Set-up costs and/or times with special characteristics have been considered by many researchers. Sometimes, the set-up times are modelled as set-up costs, which are then minimized. The idea is that set-up costs and set-up times are related, so that minimizing set-up costs will lead to minimized makespan. Parker *et al.* (1976) considered the problem with multiple parallel machine, sequence-dependent set-up costs and an upper bound on the total workload of each machine where the total set-up costs should be minimized. They modelled this situation as a vehicle routing problem. Withdrawing the workload bound constraint, Bitran and Gilbert (1990) developed a travelling salesman based heuristic to minimize total set-up costs on parallel machines where the set-up costs are not only sequence-dependent but can be divided into two classes that vary in magnitude by degrees. The classes generally correspond to major set-up times, incurred, for example, when switching from one part family to another, and minor set-up times, which accrue when switching between part types within a family. The two-class division of set-up times has been continued, although with the direct impact of set-up times being modelled as part of the makespan. Tang (1990) considers a situation in which the set-up times depend only on the part type or family being switched *to* (not being switched *from*) and uses an adaptation of the MULTI-FIT method to minimize makespan. Moreover, there are no other restrictions on the set-up times. Rajgopal and Bidanda (1991) include sequence-dependent set-up times that can be partitioned into two classes with the aim of minimizing makespan. In this case, the major set-up times are identical for all families and the minor set-up times are the same for all part types. Ovacik and Uzsoy (1993) developed worst-case error bounds in the presence of sequence-dependent set-up times that were bounded by the processing times of the job set up, also noting that list schedules need not be optimal for this case. While the inclusion of set-up times has been addressed for these special cases (major and minor set-ups for switching between or within families, set-up times bounded by processing times), restricting sequence-dependent set-up times to satisfy only the triangle inequality has not been examined.

Non-zero ready times of jobs seem not to have been discussed directly. By 'ready times', we mean a time before which the job may not begin processing or, if set-ups are required, a time before which a set-up time for the job may not begin. However, the online version of the problem has been discussed recently (Bartal *et al.* 1995). In the online problem, only the jobs that have already arrived are known and jobs must be assigned to machines as they arrive; this is in contrast to the standard problem in which all jobs to be scheduled are known *a priori*. Zhang (1997) studied the online problem in the case of two parallel identical machines with one buffer location available to hold an unassigned job in anticipation of other jobs. However, the online feature of these works is beyond that required in this work. The authors are unaware of any direct discussion of the $P|r_i, s_{ij}|C_{\max}$ problem.

Minimizing the makespan on a single machine with sequence-dependent set-up times is analogous to the travelling salesman problem (TSP) (Baker 1974). Thus, the single-machine makespan problem is NP-complete. The TSP is solved where the distances between cities represent the sequence-dependent set-up times between jobs. This results in a sequencing of jobs that yields a schedule when combined with the processing times and sequence-dependent set-up times. Several heuristics exist to generate solutions to the TSP, including the Doubling and Christofides methods (Papadimitriou and Steiglitz 1988), insertion heuristics (Reinelt 1994, Johnson and Papadimitriou 1985) and genetic algorithms (Holland 1975). Once jobs are assigned to machines in the $P|s_{ij}|C_{\max}$ problem, each machine can be viewed as a single machine makespan problem, making these methods appropriate for application to the parallel machine makespan problem.

3. Integer programming model

An integer programming formulation for the $P|r_i, s_{ij}|C_{\max}$ problem is presented in this section. First, we will define the variables and then present the formulation with a discussion of the constraints.

Let n be the number of jobs to be scheduled and m be the number of machines where $n \geq m$. We assume that machines are initially set-up for a nominal job 0 and must finish set-up for a tear down job $n+1$. We have the following definitions.

p_i	processing time of job i, $i = 1, \ldots, n$,
r_i	ready time of job i, $i = 1, \ldots, n$,
s_{ij}	sequence-dependent set-up time between job i and job j, $i, j \in \{0, 1, \ldots, n\}$,
x_{ij}	$= \begin{cases} 1 & \text{if job } i \text{ is scheduled immediately before job } j \\ 0 & \text{otherwise,} \end{cases} \quad i, j \in \{0, 1, \ldots, n\}$,
c_i	completion time of job i, $i = 1, \ldots, n$,
z	makespan of the schedule.

$$\min z \tag{1}$$

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

$$\sum_{j=1}^{n+1} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{3}$$

$$\sum_{i=0}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n \tag{4}$$

$$c_0 = 0 \tag{5}$$

$$c_j \geq p_j + \sum_{i=0}^{n} s_{ij}(s_{ij} + c_i) \quad j = 1, \ldots, n \tag{6}$$

$$c_j \geq r_j + p_j + \sum_{i=0}^{n} x_{ij} s_{ij} \qquad j = 1, \ldots, n \tag{7}$$

$$z \geq c, \qquad j = 1, \ldots, n \tag{8}$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \quad \text{for all proper subsets } S \text{ of} N = \{0, 1, \ldots, n, n+1\} \tag{9}$$

$$x_{ij} \in \{0, 1\} \qquad i,\ j \in \{0, 1, \ldots, n, n+1\} \tag{10}$$

$$x_{ij} = 0 \qquad i = j$$

$$c_j \geq 0 \quad j = 0, 1, \ldots, n. \tag{11}$$

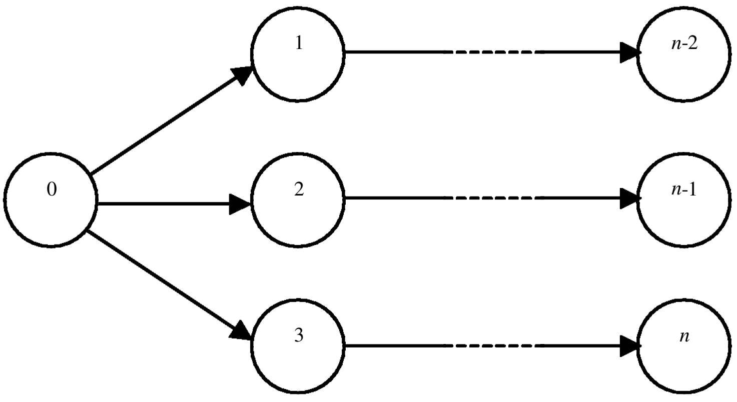

Figure 1. Graphical model of solution to the $P|r_i, s_{ij}|C_{\max}$ problem.

Equation (1) defines the objective function, which is to minimize the makespan z. We assume set-up times satisfy some sufficient condition to ensure that an optimal solution will always exist that utilizes all m machines. For instance, $s_{0j} \leq \min_{i \neq 0}(p_i + s_{ij}) \forall j$ and

$$p_j + s_{0j} \leq \frac{\sum_{i \neq j} \left(p_i + \min_k s_{ki} \right)}{m - 1}$$

are both sufficient. Constraint set (2) ensures that m machines are scheduled. Constraint sets (3) and (4) ensure that each job is scheduled on one and only one machine. Constraint (5) defines the completion time of the non-existent job 0 so that the recursive constraint sets (6) and (7) can be used to find completion times of the jobs. Constraint sets (6) and (7) also ensure that jobs only begin processing after both the previous job and the required set-up have been completed. Note that set-up for the current job cannot begin until both the current job is available and the preceding job has completed. Constraint set (8) links the decision variables c_i and z. Constraint set (9) is a cycle preventing constraint set. Constraint sets (10) and (11) provide limits on the decision variables. This formulation incorporates the $P||C_{\max}$ problem, which is a case of the $P|r_i, s_{ij}|C_{\max}$ problem where $r_i = s_{ij} = 0$ for all i and j. Since the single machine makespan problem is NP-complete, the $P||C_{\max}$ and $P|r_i, s_{ij}|C_{\max}$ problems are NP-complete as well.

Figure 1 shows a graphical model of a solution to the $P|r_i, s_{ij}|C_{\max}$ problem. In this example, $m = 3$. Each 'line' of the graph represents the ordering of jobs on a machine, beginning with job 0 and ending with job $n + 1$ on each machine. Note that once jobs are assigned to machines, each machine can be considered a TSP.

4. Heuristics

In this section, we describe several heuristics prepared for the problem $P|s_{ij}|C_{\max}$. Note that heuristics 3 and 4 can also be applied to the $P|r_i, s_{ij}|C_{\max}$ problem as well with no modifications.

4.1. *Heuristic 1: Slicing heuristic (SL)*

The first idea can be succinctly described as 'Cutting up a single machine solution'. The goal is to use a quick method to find a sequence for a single machine problem and quickly slice it up into m pieces. The general algorithm can be described as follows:

Step 1. Find a solution to the single machine problem with makespan M.
Step 2. Break the single machine sequence into m groups, one for each machine.

A target makespan for each machine is calculated as M/m. Each machine starts with job 0. Jobs are then taken from the single machine solution and added to the current machine until the schedule length of that machine exceeds M/m. At that time, the current machine is 'closed' and the next machine is 'opened'. This continues until all jobs have been assigned to a machine. The final job on a machine is the last real job, not the tear-down job. This job number can be saved and is considered the current set-up for the machine.

Important considerations include ensuring that all m machines are used, and what to do if this does not happen (or if one machine is used much less than another) and ensuring that all jobs are assigned to a machine. At this time, if $m' < m$ machines are used, the target is reset to m'/m times the original target. If not all jobs are assigned to machines then the target is reset to (number of jobs to be scheduled/number of placed jobs) times the original target. Let j be the index of the current job being examined.

The implemented heuristic is as follows.

Step 1. Use the Doubling Method to solve a TSP and find a solution to the single machine problem. Call the resultant makespan M. The solution gives the job order to be used in step 0, starting at the first 'real' job. That is, skip job 0, which indicates the start of the sequence.
Step 2. Set an approximate target $t = M/m$.
Step 3. Let $mc = 1$. Schedule job 0 on this machine. Let $j = 1$.
Step 4. If all the jobs have been scheduled, go to step 0.
Step 5. If machine number mc has a schedule length $> t$, place job j on machine number mc and let $j = j + 1$. Go to step 0.
Step 6. If machine number mc has a schedule length $\geq t$, close machine number mc. Let $mc = mc + 1$. Schedule job 0 on this machine. Go to step 0.
Step 7. If the number of machines used is less than that available ($mc < m$), let $t_{new} = t_{old}(mc/m)$. Unschedule all the jobs and go to step 0.
Step 8. If more than m machines were used ($mc > m$), let k be the number of jobs placed on the first m machines. Let $t_{new} = t_{old}(n/k)$. Unschedule all the jobs and go to step 0.
Step 9. If m machines were used, DONE.

4.2. *Heuristic 2: Multiple MULTI-FIT heuristic (MMF)*

The second idea to be explored is an adaptation of the MULTI-FIT algorithm by Coffman *et al.* (1978). While MULTI-FIT does not consider sequence-dependent set-up times, the set-up times can be accounted for by integrating their values into the processing times. The obvious estimates are the minimum, average and maximum set-up time. Following an initial empirical study, it was decided to use the minimum set-up time. MULTI-FIT can then be performed using the modified processing times

$p_i' = p_i + \min_j(s_{ij})$. Once jobs have been assigned to machines, the true processing and set-up times can be used. The TSP can then be used on each machine's sequence to see if a better ordering of those jobs can be found. Otherwise, the modified MULTI-FIT sequence is used.

Let k be the maximum number of iterations. The following describes the heuristic.

```
Step 1. Create the modified processing times.
Step 2. Order the jobs in non-increasing order of the modified processing times.
Step 3. Calculate $CL = \max(1/m \sum p_i, \max_i p_i')$.
Step 4. Calculate $CU = \max(2/m \sum p_i', \max_i p_i')$.
Step 5. $I = 1$.
Step 6. If $I > k$, STOP. Go to step 0.
          Else, $C = (CL + CU)/2$.
Step 7. Perform the First Fit Decreasing heuristic with the modified processing
        times. Let $c$ be the number of machines required.
          If $c \leq m$, let $CU = C$ and $CL = CL$.
          Else, let $CU = CU$ and $CL = C$.
          Let $I = I + 1$, go to step 0.
Step 8. For each machine, solve the TSP with the true processing and set-up times.
        If the makespan on that machine is reduced, use the TSP ordering.
Step 9. Compute the actual makespan.
```

4.3. *Heuristic 3: Multiple insertion heuristic (MI)*

The third idea is a multiple machine adaptation of the Insertion Heuristic for TSP. The Insertion Heuristic for the TSP is one in which set-up times are accounted for by integrating their values into the processing times. As with MMF, we will use the minimum set-up time and the modified processing times are found using the definition $p_i' = p_i + \min_j(s_{ij})$. The Insertion Heuristic can then be performed using these modified processing times. Once jobs have been assigned to machines, the true processing and set-up times can be used.

MI has the following steps:

```
Step 1. Create the modified processing times.
Step 2. Order the jobs in non-increasing order of the modified processing times.
Step 3. Examine every job in the order found in step 0. For each job $i$,
        Step 3(a) insert job $i$ into every position on each machine
        Step 3(b) calculate the true partial makespan for each position of job $i$
        Step 3(c) place job $i$ in the position on the machine with the lowest resultant
                  partial makespan using the actual set-up times.
```

4.4. *Heuristic 4: Genetic algorithm (GA)*

Genetic algorithms (GAs) have been applied to many combinatorial problems (Holland 1975, Davis and Streenstrup 1987). In basic genetic algorithms, a number of problem solutions (chromosomes) are generated, representing a population. They are evaluated based on how well they solve the problem of interest. The quality of each chromosome is measured in some way called the 'fitness' of the chromosome. In each generation, chromosomes can change in random ways, analogous to mutations in the physical world. A new generation is generated out of the old generation through a reproduction scheme that allows 'fitter' chromosomes to reproduce more often but which does not eliminate the chances that 'poor' chromosomes will reproduce as well. Designing GAs requires consideration of five primary components (Davis and Streenstrup 1987):

(1) a chromosomal representation of solutions to the problem;
(2) genetic operators that change the composition of the chromosomes;
(3) a method to initialize a population;
(4) an evaluation function that represents how well the individual solutions function in the environment, called their 'fitness';
(5) the parameters that are required in order to implement the above components, including population size, number of generations that will be allowed, and stopping criteria.

The chromosomal representation of the scheduling information can take many forms and influence the types of genetic operators. One option for the chromosomal representation is an array indexed by the job number where each value in the array corresponds to the machine to which the job is assigned. For example, the chromosome [2, 3, 1, 3, 2, 1] means that machine 1 has jobs 3 and 6, machine 2 has jobs 1 and 5, and machine 3 has jobs 2 and 4. This kind of structure is used by Gu and Chung (1999) for assigning gates to flights in airports. In their application, gates can be repeated but the ordering of flights is not part of the problem, since the flight schedule, even if changing due to delay, is an input and not an output. In our application, however, the job order is relevant. The job order could be found by a TSP for each machine. With this representation, mutations consist of changing the assigned machine for one or more jobs. Each job in each chromosome in the current population is eligible for crossover, and any number of jobs can be selected for crossover. Clearly, these operations could result in no change in the schedule, reducing the effective probabilities of crossover and mutation. Sivrikaya-Serifoglu and Ulusoy (1999) use the job and machine, in that order, as a gene. The jobs are processed on the machines in the order in which they appear in the chromosome. Their example chromosome [4-1, 2-1, 5-2, 1-1, 3-2] means that machine 1 processes jobs 4, 2, and 1 in that order and machine 2 processes jobs 5 and 3. With this representation, a swap mutation involves selecting two jobs in a chromosome in the current population and exchanging them. This may result in a different job ordering on up to two machines. A bit mutation involves selecting a job and randomly assigning a machine to that job. This representation and types of mutation could also result in no change to the schedule.

We choose to use just the machine information in the gene and then generate solutions for m TSPs through the Doubling Method. If we spend a lot of time solving TSPs each iteration, then, and only then, would we include the job order in the gene information. Based on initial experimentation, using TSPs has been found to be sufficient.

The population is initialized randomly, so that every job has an equally likely chance of starting out on any one machine. The fitness function is $(C_{\max})^{-1}$.

The important parameters for the genetic algorithm are as follows:

POPSIZE	the constant number of chromosomes in the population,
PROBMUTATE	the probability that a particular gene undergoes mutation,

PROBCROSS	the probability that a particular chromosome is chosen for crossover,
NUMITERBASE	the value, which when multiplied by n, yields the number of iterations that will define a stopping criterion,
EPSILON	the difference between two successive incumbent solutions that will define a second stopping criterion.

The general structure for the genetic algorithm implemented here follows. Each step is described in more detail in the corresponding figure.

Step 1. Initialize a population of chromosomes. See figure 2.
Step 2. Evaluate the chromosomes in the population to get each chromosome's fitness. See figure 3.
Step 3. Find the new incumbent chromosome and consider stopping. If do not stop, go to step 4. See figure 4.
Step 4. Reproduce to create the next generation, allowing chromosomes with higher fitness values to have higher chances of reproduction. See figure 5.
Step 5. Apply crossover to generation g. See figure 6.
Step 6. Apply mutation to generation g. See figure 7.
Step 7. Go to step 2.

Initialization

```
Set the generation (iteration) number $g$=0.
For $i$=1 to POPSIZE,
    For $j$=1 to $n$,
        Let Generation{$g$}.PopMember{$i$}.gene{$j$} be a randomly generated integer
            between 1 and $m$ inclusive.
Best.fitness=0, OldBest.fitness=0
```

Figure 2. Population initialization.

Chromosome evaluation

```
For $i$=1 to POPSIZE,
    For $k$=1 to $m$,
        Select the set of jobs $J_k$ such that Generation{$g$}.PopMember{$i$}.gene{$j$}=$k$ for
            $j \in J_k$
        Use the Doubling Method for the jobs $J_k$ to find an order for machine $k$
    Find the makespan Cmax for population member $i$
    Let the fitness be Generation{$g$}.PopMember{$i$}.fitness=$(\text{Cmax})^{-1}$
```

Figure 3. Chromosome evaluation.

Find new incumbent, consider stopping

```
NewBest = population member in generation $g$ with the maximum fitness.
If (NewBest.fitness>Best.fitness),
        Copy Best to OldBest, Copy NewBest to Best
        If (Best.fitness – OldBest.fitness < EPSILON),
                The Best schedule is represented by Best:  STOP
If ($g$= NUMITERBASE*$n$),
        The Best schedule is represented by Best:  STOP
Else $g$=$g$+1 and go to step 4.
```

Figure 4. Incumbent and stopping criteria.

Reproduction

```
TotalFit=0
For $i$=1 to POPSIZE,
        TotalFit=TotalFit + Generation{$g$-1}.PopMember{$i$}.fitness
Sort the population members in generation $g$-1 in decreasing order of fitness.  Let ($i$) indicate the
        $i$th element.
Generation{$g$-1}.PopMember{(1)}.ub= Generation{$g$-1}.PopMember{(1)}.fitness / TotalFit
For $i$=2 to POPSIZE,
        Generation{g-1}.PopMember{(i)}.ub= Generation{g-1}.PopMember{(i-1)}.ub +
                Generation{g-1}.PopMember{(i)}.fitness / TotalFit
For $i$=1 to POPSIZE,
        Generate a random real number $s$ between 0 and 1.
        Find the population member ($l$) such that Generation{$g$-1}.PopMember{($l$-1)}.ub <
                $s \leq$ Generation{$g$-1}.PopMember{($l$)}.ub
        Copy Generation{$g$-1}.PopMember{($l$)} to Generation{$g$}.PopMember{$i$}
```

Figure 5. Reproduction.

The Genetic Algorithm has several parameters that need to be tuned. After initial experimentation, the following parameters were found: POPSIZE = 20, PROBMUTATE = 0.5, PROBCROSS = 0.25, NUMITERBASE = 20, EPSILON = 0.000001.

4.5. *Discussion of heuristics*

This problem has the characteristics of both parallel machine scheduling and the travelling salesman problem. The heuristics developed in this section draw their motivation from methods that have proven effective for these two separate problems. The SL and MI procedures derive from the TSP. Both are then modified for parallel

Crossover

```
For $i$=1 to POPSIZE,
    NumToCross=0
    For $j$=1 to $n$,
        Generate a random real number $s$ between 0 and 1.
        If ($s$<PROBCROSS)
            Add $j$ to the set of jobs to be crossed in population member $i$, $C_i$
            NumToCross=NumToCross+1
    If NumToCross is odd,
        Generate a random real number $s$ between 0 and 1.
        If $s$<0.5,
            Generate a random integer $s$ between 1 and NumToCross inclusive.
            Remove job ($s$) from $C_i$
            NumToCross=NumToCross-1
        Else, while NumToCross is odd
            Generate a random integer $s$ between 1 and $n$ inclusive until $s \notin C_i$
            Add job $s$ to $C_i$
            NumToCross=NumToCross+1
    CrossPairs=NumToCross/2
    For $j$=1 to CrossPairs,
        Generate two different random integers $s$ and $t$ between 1 and NumToCross
            inclusive.
        Swap Generation{$g$}.PopMember{$i$}.gene{($s$)} and
            Generation{$g$}.PopMember{$i$}.gene{($t$)}
        Remove jobs ($s$) and ($t$) from $C_i$
        NumToCross=NumToCross-2
```

Figure 6. Crossover.

Mutation

```
For $i$=1 to POPSIZE,
    For $j$=1 to $n$,
        Generate a random real number $s$ between 0 and 1.
        If ($s$<PROBMUTATE)
            Let Generation{$g$}.PopMember{$i$}.gene{$j$} be a randomly generated integer
                between 1 and $m$ inclusive.
```

Figure 7. Mutation.

machines. MMF is adapted from parallel processor solution procedures but with an added step to address sequence-dependent set-ups. The genetic algorithm attempts to consider equally, both aspects of the problem.

5. Experimental design

An experiment was performed to evaluate the heuristics for problem $P|r_i, s_{ij}|C\max$. The experimental runs contain on the order of 100 jobs to be scheduled on up to 10 machines. Processing times and sequence-dependent set-up times are chosen randomly from a uniform distribution as rational values with two decimal places. We first describe the experimental set-up.

5.1. *Ready times*

Each value r_i is either the zero vector (all jobs available at time 0) or each element is drawn from a uniform distribution with bounds a and b. Our intent is to control the standard deviation σ_r relative to the mean. In setting ready times we want to emulate the arrival of jobs. The lower bound is set to zero. The upper bound should be the expected makespan of the (for now non-existent) preceding stage—the expected completion time for the last job through that stage. This would, of course, depend on the structure of the preceding stage. Assume that the preceding stage has essentially the same structure as the current stage. The preceding stage's makespan can be roughly estimated by examining the average number of jobs per machine (n_m), the mean of the processing time (μ_p) and the mean of the set-up time (μ_s). Every job must be processed and set-up. The makespan can be estimated by $n_m(\mu_p + \mu_s)$. Jobs arrive to the current stage as soon as the first job is done, which happens approximately at time $\mu_p + \mu_s$. Thus, jobs arrive at the current stage over a time period with length $n_m(\mu_p + \mu_s) - (\mu_p + \mu_s) = (n_m - 1)(\mu_p + \mu_s)$. Note that one job must arrive at time 0. Since the length is an estimate only, it is not necessary to force a job to arrive at time $(n_m - 1)(\mu_p + \mu_s)$. The ready times are thus generated from a uniform distribution with lower bound zero and upper bound $(n_m - 1)(\mu_p + \mu_s)$, transformed so that one job arrives at time 0.

5.2. *Set-up times*

Each value $s_{ij}, i \neq j$ is drawn from a uniform distribution with standard deviation σ_s and the values s_{ii} are set to a large value. While set-up times always satisfy the triangle inequality, they may either be symmetric or asymmetric. Set-up times that satisfy the triangle inequality are desired so that the direct path between two jobs is always no longer than any non-direct path between two jobs. Moreover, the SL heuristic requires the triangle inequality. If a matrix is generated with two values lower than half the largest value in the matrix, then the triangle inequality may not hold for that matrix. To prevent this, the restriction that the lower bound must be at least half the upper bound has been introduced. Let a be the lower bound and b be the upper bound. Then, for uniformly distributed set-ups, $\mu_s = a + b/2$ and $\sigma_s^2 = (b-a)^2/12$. Solving these two equations yields $a = \mu_s - \sqrt{3}\sigma_s$ and $b = \mu_s + \sqrt{3}\sigma_t$. If the triangle inequality is to hold, then $2a \geq b$ and so $\sigma_s \leq (1/3\sqrt{3})\mu_z = \sqrt{3}\mu_s/9$. Thus, only the mean must be specified. We select $\mu_s = 450$ so that σ_s is rational.

5.3. *Processing times*

Each value p_i is drawn from a uniform distribution with standard deviation σ_s. However, the range of the processing times must correspond to the set-up times in some way. Following Morris and Tersine (1990), the mean of the processing times can take on one of two values: $\mu_p = \mu_s$ or $\mu_p = 10\mu_s$. The range of processing times is set at either $[0.94\mu_p, 1.06\mu_p]$ or $[0.4\mu_p, 1.6\mu_p]$.

5.4. *Place-holder jobs*

Two jobs must be added for implementation reasons. Jobs 0 and $n+1$ are the place holder jobs. These jobs are basically markers for the beginning and ending of the sequence of jobs. However, they also indicate the initial machine set-up and required ending state. If the start and end states of the machines are known, then a job sequence can be found including these details. If the required start and end states are not known, then the place holder jobs can provide substitute information. With the addition of these two jobs, there are $n+2$ jobs, numbered from 0 to $n+1$. Their ready times are zero and their processing times are zero. Their set-up times are found using the equations $s_{0i} = \max_{\forall j \neq i}\{s_{ji}\}$ and $s_{i,n+1} = s_{0i}$. The maximum function was chosen so that the makespan would not be under estimated due to the lack of knowledge regarding the required initial and ending machine states.

5.5. *Problem data characterization*

Problem data can now be characterized by seven factors: ready times, range of processing times, mean of processing times, variability of set-up times, set-up times structure, the number of machines and average number of jobs per machine. Each of these factors is tested at two levels: low and high. The meanings of these levels are shown in table 1.

Since each factor has two levels, there are $2^7 = 128$ possible experimental designs. Ten data sets for each of these cases have been generated.

Factor	Low	High
Ready times	$r = 0$	$r \sim \text{Unif}(0, (n_m - 1)(\mu_p + \mu_s))$
Range of processing times	$p \sim \text{Unif}(0.94\mu_p, 1.06\mu_p)$	$p \sim \text{Unif}(0.4\mu_p, 1.6\mu_p)$
Mean of processing times	$\mu_p = \mu_s$	$\mu_p = 10\mu_s$
Set-up times structure	Symmetric	Asymmetric
Std dev of set-up times $S \sim \text{Unif}(\mu_s - \sqrt{3}\sigma_s, \mu_s + \sqrt{3}\sigma_s)$ $\mu_s = 450$	$\sigma_s = \frac{1}{2}\frac{1.5\mu_s}{9}$	$\sigma_s = \frac{1.5\mu_s}{9}$
Number of machines	2	10
Ave. number of jobs per machine $= n_m$	3	10

Table 1. Factor levels.

6. Identifying bounds

In this section, we shall characterize problem solutions. Lower bounds are developed as well as a technique for finding optimal solutions for certain problems. We conclude this section with a discussion of the error introduced by using asymmetric set-up time matrices.

6.1. *Lower bounds*

A lower bound on the makespan can be found by looking at the minimum pre-emptive schedule makespan (McNaughton 1959). We have n jobs that need to be scheduled, each of which must be processed and set-up. The minimum set-up time for each job is the lower bound of the set-up time distribution, which is $LB_S = \mu_s - \sqrt{3}\sigma_s$. The minimum processing time is the lower bound of the processing time distribution LB_P. Thus, a lower bound on the makespan is $LB^{(1)}_{C_{\max}} = n(LB_S + LB_P)/m$.

Especially in cases with a high Range of Processing Times, the general pre-emptive lower bound can be very poor. We use a pre-emptive type lower bound for each of the individual data sets using the actual data. Clearly, a job cannot begin until it is ready. Another lower bound on the makespan is the largest ready time plus its processing time. We shall use the larger of the pre-emptive type data-dependent lower bound and the largest ready time plus processing time.

A data dependent lower bound for each data set is

$$LB^{(2)}_{C_{\max}} = \max\left(\frac{1}{m}\left\{\sum_{i=1}^{n}\left[p_i + \min_{j\in\{1,\ldots,n\}}(s_{ij})\right]\right\}, \max_i\left\{r_i + \min_{j\in(1,\ldots,n\}}(s_{ij}) + p_i\right\}\right).$$

6.2. *Scheduling computational complexity*

The optimal solution can be found for the case of $m = 2$. Consider the case with six jobs. The first machine could conceivably have any number from 1 to 5 jobs assigned to it. Since machines are identical and indistinguishable, assigning one job to machine 1 and five jobs to machine 2 is equivalent to assigning five jobs to machine 1 and one job to machine 2. For this reason, machine 1 can have one, two or three jobs and machine 2 can have five, four or three jobs. Examine these configurations of assignments:

(1) machine 1 has one job, machine 2 has five jobs in the TSP order;
(2) machine 1 has two jobs in the TSP order, machine 2 has four jobs in the TSP order;
(3) machine 1 has three jobs in the TSP order, machine 2 has three jobs in the TSP order.

Finding the best makespan over all the configuration types will yield the best schedule for two machines and six jobs to minimize makespan. The same argument follows for the cases with two machines and 20 jobs. However, the number of different configurations is much larger.

The number of configurations in the six-jobs case is small enough to enable enumeration to find the optimal solution, while the problem with 20 jobs is much too large to enable finding the optimal solution. Consider the cases where only the

10,10 configurations need to be examined. First, there are $\binom{20}{10}$ ways to choose the ten jobs to go on the first machine. Since there is no difference between the machines, this can be reduced to $\binom{20}{10}\frac{1}{2}$. Then, the TSP solution for each of the machine assignments must be found. Since there are 10! ways to order each machine, there are $\binom{20}{10}\frac{1}{2}10!10! = \frac{20!}{2} \approx 1.216 \times 10^{18}$ possibilities. Since the jobs could also be distributed in the 9,11 configuration, this configuration would need to be examined as well. This continues for all the required configurations. Thus, even if we could use data based bounds to reduce the number of configurations to be tested, enumeration would still not be feasible.

If there are more than two machines, it is much more difficult to find the optimal makespan. Consider 30 jobs and 10 machines, under the assumption that three jobs will be on each machine in the optimal solution. The first machine could have $\binom{30}{3}$ different sets of three jobs assigned to it. The second machine could have $\binom{27}{3}$ different sets of three jobs assigned to it and so on. The TSP solution for each of these machines must then be found. This results in

$$\binom{30}{3}\binom{27}{3}\binom{24}{3}\cdots\binom{6}{3}\binom{3}{3}(3!)^{10} = 30!$$

possibilities. Moreover, the assumption that the jobs would be evenly distributed among machines is flawed. The jobs could be distributed among the machines in many possible configurations, each of which would have to be considered. Due to this explosion of scenarios that would need to be examined, the optimal solution has only been found for the two-machine, average three jobs per machine cases.

6.3. *Symmetric matrix ceiling*

In our intended application, set-up time matrices could be asymmetric. Portions of the heuristics assume symmetric matrices. However, if the set-up time matrix is asymmetric, it can be replaced by a symmetric matrix with a known error. Consider the following definitions. Let M^* be the length of the TSP tour of the asymmetric set-up time matrix. Let M_s^* be the length of the TSP tour of the asymmetric matrix's symmetric replacement M, called the *Symmetric Matrix Ceiling*. We define $M = (M_{ij})$ where $M_{ij} = \max(s_{ij}, s_{ji})$ and $E_{ij} = M_{ij} - s_{ij}$.

Theorem:

$$M_S^* \leq M^* + \min\left(\sum_i \max_j\{E_{ij}\}, \sum_j \max_i\{E_{ij}\}\right).$$

Proof

Replace s_{ij} and s_{ji} by M_{ij}. Each value E_{ij} is either 0 or positive. Consider a feasible TSP solution to M. This solution contains exactly one element from each row and column. Either element (i, j) has $M_{ij} = s_{ij}$ or $M_{ij} = s_{ij} + E_{ij}$. Let $X_{ij} = 1$ if element (i, j) is in the TSP solution and $X_{ij} = 0$ otherwise. A feasible TSP solution on M then has value

$$M_S^* = \sum_i \sum_j X_{ij} M_{ij} = \sum_i \sum_j X_{ij} \max(s_{ij}, s_{ji})$$

Note that

$$\sum_i \sum_j X_{ij} \max(s_{ij}, s_{ji}) \leq M^* + \sum_i \max_j \{E_{ij}\}$$

and

$$\sum_i \sum_j X_{ij} \max(s_{ij}, s_{ji}) \leq M^* + \sum_j \max_i \{E_{ij}\}.$$

7. Experimental results

7.1. *No ready times*

7.1.1. *Comparison to data dependent lower bounds*

An ANOVA was performed to compare the four heuristics for the no ready time case. The heuristic used was found to be significant at the 0.001 level (F statistic = 836.382 with three degrees of freedom, 2304 error degrees of freedom). MI proved to be the best. Several tables with additional data have been compiled to support this conclusion. The average values for the makespan divided by the data dependent lower bounds for each heuristic are summarized in figure 8. Each data point is the average over all cases with the shown levels for Range of Processing Times, Standard Deviation of Set-up Times, Number of Machines and Average Number of Jobs. Note that for many cases, GA outperforms MI even though the ANOVA indicated that MI is better overall. In general, GA performs well in cases with the low level for Number of Machines and poorly in cases with the high level for Number of Machines. Figure 9 shows the number of times each heuristic had the lowest makespan. Note that the sums of the bars for one case may be more than 40 because the heuristics could have found the same solution. However, each bar can range between 0 and 40. We see that, while GA could be considered competitive in the 6 and 20 job cases, MI dominates in the 30 and 100 job cases. Out of 640 trails, MI has the lowest makespan of the four heuristics 446 times while GA has the lowest makespan only 207 times.

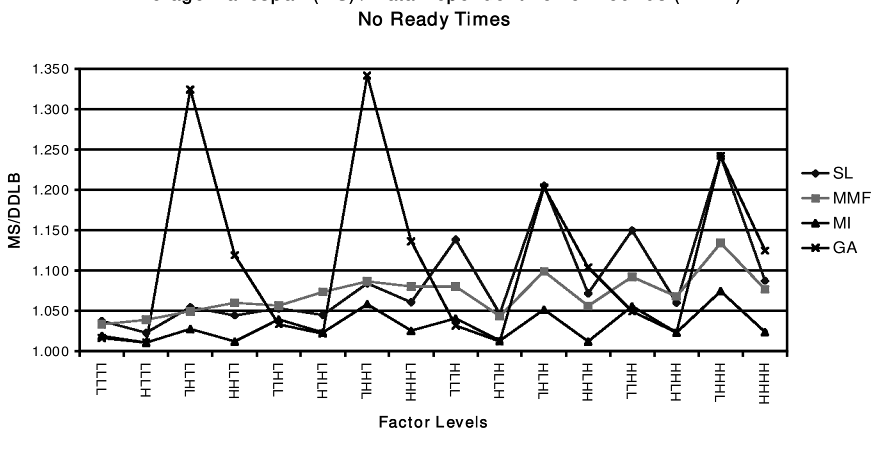

Figure 8. Average makespan divided by data dependent lower bounds—no ready times.

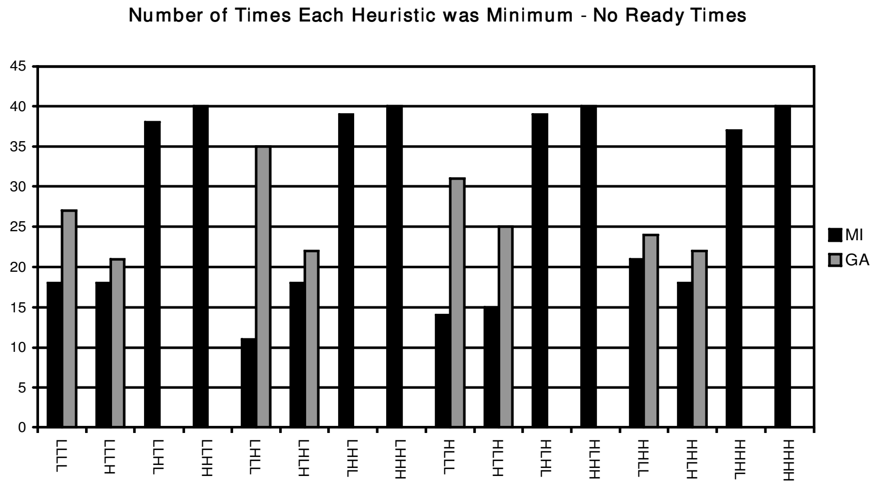

Figure 9. Number of times achieve minimum vlaue—no ready times.

7.1.2. *Comparison to optimal*

The ratio between the makespan and the optimal solution for cases with two machines and six jobs has been summarized in figure 10. Each data point is the average over all cases with the shown levels for Range of Processing Times, Mean of Processing Times, Set-up Times Structure and Standard Deviation of Set-up Times; the Number of Machines and Average Number of Jobs factors are both held at the low level. GA has the lowest ratio in all but one case. The number of times each heuristic achieved the optimal makespan, for cases with two machines and three jobs, has been summarized in figure 11. MI achieves the optimal makespan 37 times in the 160 data sets examined. GA achieves the optimal makespan 78 times. Moreover, SL only achieves the optimal makespan six times and MMF only achieves the optimal makespan twice. The average and maximum ratio of achieved makespan

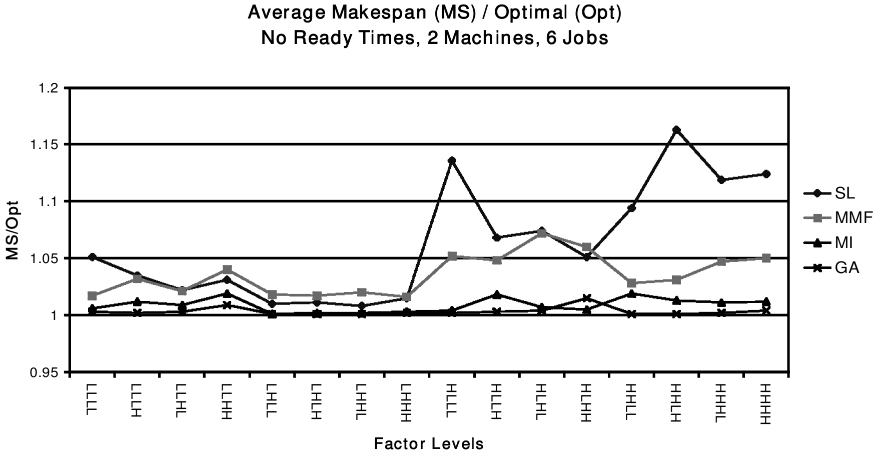

Figure 10. Average makespan divided by optimal—no ready times, two machines, six jobs.

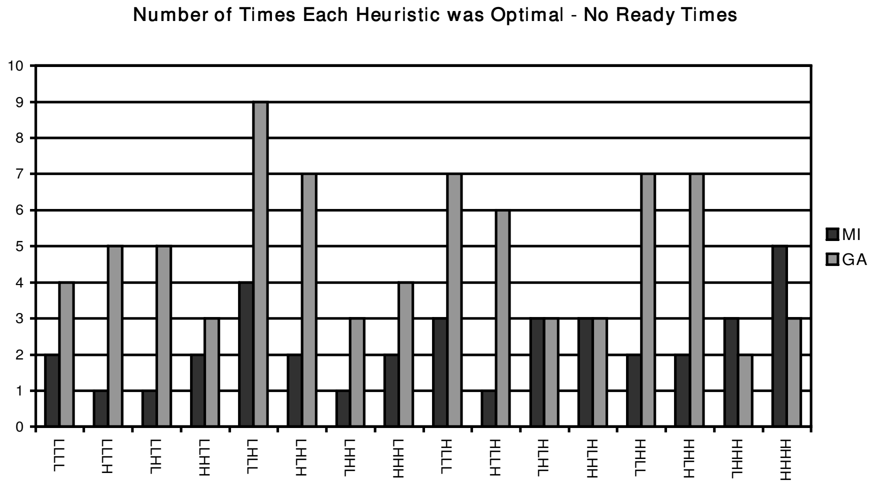

Figure 11. Number of times achieve optimal value—no ready times, twomahcines, six jobs.

to optimal makespan for cases with two machines and three jobs has been summarized in table 2. GA has the lowest ratio at 1.047, followed by MI at 1.079. In comparison to the optimal solutions, GA appears preferable.

However, the added element of running time must be considered also. Each heuristic was written in C, compiled with Microsoft Visual C++ and run on a PC with a Pentium II processor and 64 MB RAM. The running times were found using the clock( ) function and the averages over the data files with 6, 20, 30 and 100 jobs have been tabulated in table 3. If the average time was less than 0.01 seconds, a dash has been used. Again, MI performed at least as well as the other heuristics in all cases on average. GA performed horribly on this measure, taking 100 to 1000 times longer than the next slowest alternative. In the 30 and 100 job cases, these long running times cannot be considered in light of the quality of the solution, because the solutions were not very good.

Heuristic	Average	Maximum
SL	1.063	1.621
MMF	1.036	1.152
MI	1.009	1.079
GA	1.003	1.047

Table 2. Makespan optimal—no ready times.

Heuristic	LL	LH	HL	HH
SL	—	—	—	1.73
MMF	—	—	—	39.12
MI	0.01	—	—	49.75
GA	0.95	0.09	0.08	1206.32

Table 3. Running times—no ready times.

SL has a fundamental flaw. It ignores the modified set-up times when the sequence is sliced. In addition, the discrete time aspect of the problem is ignored when setting candidate makespans. Thus, a job that would increase makespan by a small may be forced to a new machine causing problems later on. Moreover, despite the iterative attempts to eliminate the problem, there is nothing in SL that guarantees one machine will not have all the jobs in the schedule.

Using many different measures, MI has been found to be best for the types of problems with no ready times. Note that MI explicitly considers both the *m* machines and set-up time interactions when placing jobs. While GA is attractive in some instances, its overall performance, especially in light of its running time, is inferior to MI.

7.2. *Ready times*

7.2.1. *Comparison with data dependent lower bounds*

An ANOVA was performed to compare the two heuristics for the ready time case. Once again, the heuristic used was found to be significant at the 0.001 level (F statistic = 1139.089 with one degree of freedom, 1278 error degrees of freedom) and MI to be the best. Several tables have been compiled to support this conclusion. The average value for the makespan divided by the data dependent lower bounds for each heuristic has been summarized in figure 12. Each data point is the average over all cases with the shown levels for Range of Processing Times, Standard Deviation of Set-up Times, Number of Machines and Average Number of Jobs. MI outperforms GA in every data point. Figure 13 shows the number of times each heuristic had the lowest makespan. We see that while GA finds lower schedules in some cases, this only occurs in cases with six jobs. Out of 640 trails, MI has the lowest makespan 610 times while GA has the lowest makespan only 30 times.

In comparison with the data dependent lower bounds, MI is preferable.

7.2.2. *Comparison with optimal*

The ratio between the makespan and the optimal solution for cases with two machines and six jobs has been summarized in figure 14. MI has the lowest ratio in all cases. The number of times each heuristic achieved the optimal makespan, for cases with two machines and six jobs, has been summarized in figure 15. MI achieves

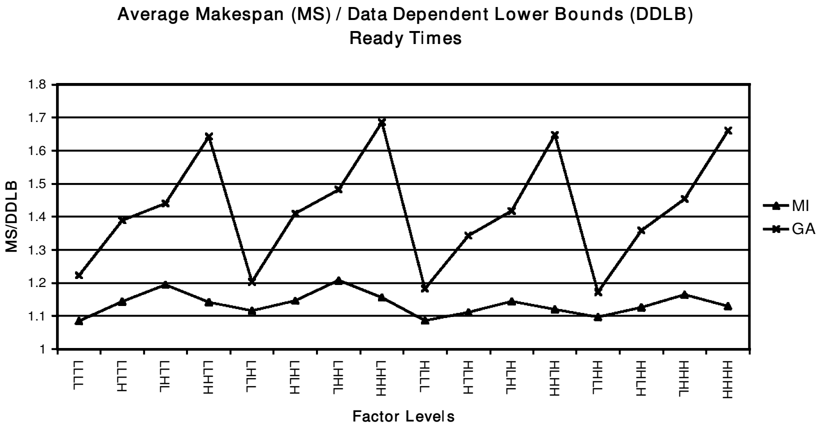

Figure 12. Number of times achieve minimum value—ready times.

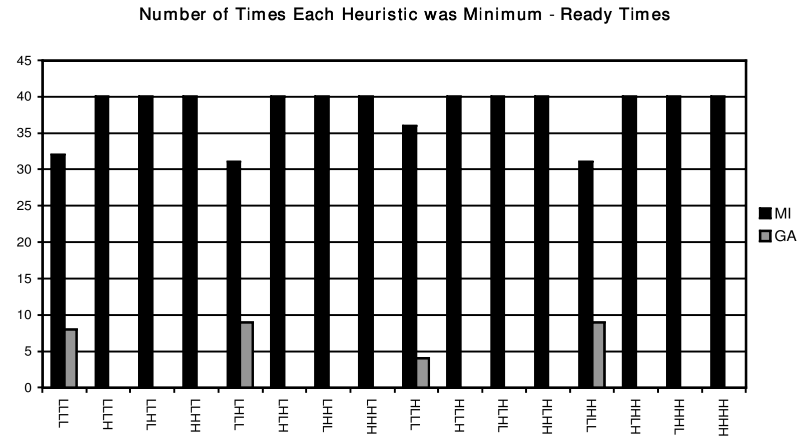

Figure 13. Number of times achieve minimum value—ready times.

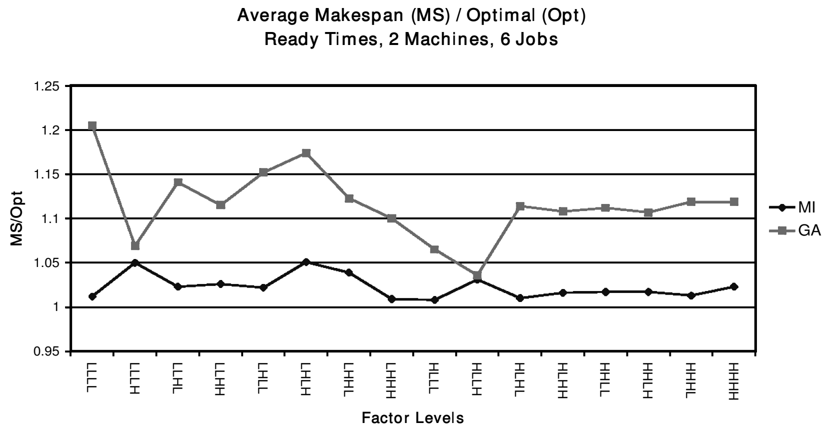

Figure 14. Average makespan divided by optimal—ready times, two machines, six jobs.

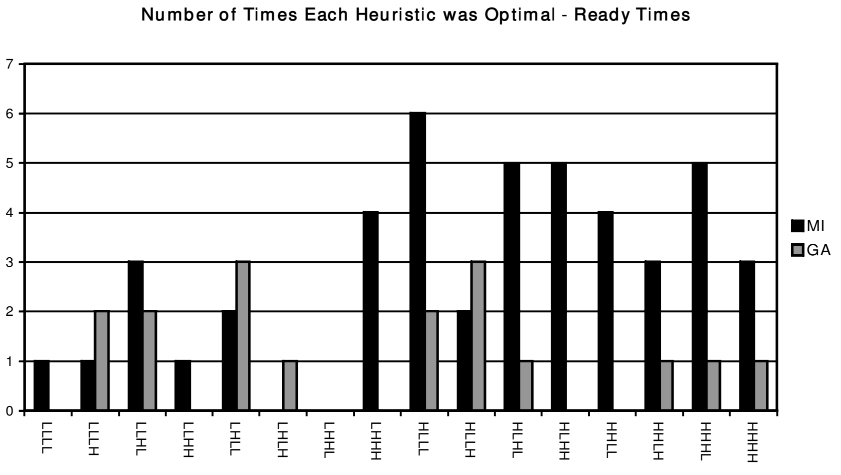

Figure 15. Number of times achieve optimal value—ready times, two machines, six jobs.

Heuristic	Average	Maximum
MI	1.023	1.232
GA	1.116	1.408

Table 4. Makespan/optimal—ready times.

the optimal makespan 45 times in the 160 data sets examined. GA achieves the optimal makespan 17 times. The average and maximum ratio of achieved makespan to optimal makespan for cases with two machines and three jobs has been summarized in table 4. GA has the higher ratio at 1.408 compared to MI at 1.232. In comparison to the optimal solutions, the MI is preferable.

Using many different measures, MI has been found to be best for the types of problems with ready times.

8. Conclusions

We have examined the problem of scheduling sequence-dependent jobs on identical parallel processors. Ready times were allowed. This is seen as a building block to a multistage environment and replicates actual job availability in a dynamic operation. Using many different measures, MI has been found to be best for the types of problems discussed here. While GA is attractive in some instances, its overall performance, especially in light of its running time, is inferior to MI. Some of the worst performances of GA result from the parameter values. The amount of time required to find appropriate parameter values and achieve good solutions, even in the six-job case, is so large as to limit perhaps the value of refining the GA, as implemented here, further. However, other variants of genetic algorithms, such as the Random Keys Genetic Algorithm by Bean (1994) may respond better to the problem. Further work in this area may also incorporate due dates.

Acknowledgement

M. E. Kurz's work was supported by the Engineering Research Program of the Office of Basic Energy Sciences at the Department of Energy.

References

Babel, L., Kellerer, H. and Kotov, V., 1998, The k-partitioning problem. *Mathematical Methods of Operations Research*, **47**, 59–82.

Baker, K., 1974, *Introduction to Sequencing and Scheduling* (New York: Wiley).

Bartal, Y., Fiat, A., Karloff, H. and Vohra, R., 1995, New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, **51**, 359–366.

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

Bitran, G. R. and Gilbert, S. M., 1990, Sequencing production on parallel machines with two magnitudes of sequence dependent set-up cost. *Journal of Manufacturing and Operations Management*, **3**, 24–52.

Blocher, J. D. and Chand, S., 1991, Scheduling of parallel processors: a posterior bound on LPT sequencing and a two-step algorithm. *Naval Research Logistics*, **38**, 273–287.

Coffman, Jr., E. G. and Sethi, R., 1976, A generalized bound on LPT sequencing. *RAIRO-Informatique*, **10**, 17–25.

Coffman, Jr., E. G., Garey, M. R. and Johnson, D. S., 1978, An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, **7**, 1–17.

Corrêa, R. C., Ferreira, A. and Rebreyend, P., 1999, Scheduling multiprocessor tasks with genetic algorithms. *IEEE Transactions on Parallel and Distributed Systems*, **10**, 825–837.

Davis, L. and Streenstrup, M., 1987, Genetic algorithms and simulated annealing: an overview. In *Genetic Algorithms and Simulated Annealing*, edited by L. Davis (London: Pitman), pp. 1–11.

Friesen, D. K., 1984, Tighter bounds for the MULTIFIT processor scheduling algorithm. *SIAM Journal on Computing*, **13**, 170–181.

Graham, R. L., 1969, Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, **17**, 416–429.

Gu, Y. and Chung, C. A., 1999, Genetic algorithm approach to aircraft gate reassignment problem. *Journal of Transportation Engineering*, **125**, 384–389.

Guignard, M., 1993, Solving makespan minimization problems with Lagrangean decomposition. *Discrete Applied Mathematics*, **42**, 17–29.

Holland, J. H., 1975, *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: The University of Michigan Press).

Hou, E., Ansari, N. and Ren, H., 1994, A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, **5**, 113–120.

Hu, T. C., 1961, Parallel sequencing and assembly line problems. *Operations Research*, **9**, 841–848.

Johnson, D. S. and Papadimitriou, C. H., 1985, Computational complexity. In *The Traveling Salesman Problem*, edited by E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy, and D. B. Shmoys (Chichester, UK: Wiley), pp. 37–85.

Lee, C.-Y. and Massey, J. D., 1988, Multiprocessor scheduling: combining LPT and MULTIFIT. *Discrete Applied Mathematics*, **20**, 233–242.

McNaughton, R., 1959, Scheduling with deadlines and loss functions. *Management Science*, **6**, 1–12.

Min, L. and Cheng, W., 1998, Identical parallel machine scheduling problem for minimizing the makespan using genetic algorithm combined by simulated annealing. *Chinese Journal of Electronics*, **7**, 317–321.

Monma, C. L. and Potts, C. N., 1989, On the complexity of scheduling with batch setup times. *Operations Research*, **37**, 798–814.

Monma, C. L. and Potts, C. N., 1993, Analysis of heuristics for preemptive parallel machine scheduling with batch setup times. *Operations Research*, **41**, 981–993.

Morris, J. S. and Tersine, R. J., 1990, A simulation analysis of factors influencing the attractiveness of group technology cellular layouts. *Management Science*, **36**, 1567–1578.

Muntz, R. R. and Coffman, E. G., 1969, Optimal preemptive scheduling on two-processor systems. *IEEE Transactions on Computers*, **18**, 1014–1020.

Muntz, R. R. and Coffman, E. G., 1970, Preemptive scheduling of real-time tasks on multiprocessor systems. *Journal of the ACM*, **17**, 324–338.

Ovacik, I. M. and Uzsoy, R., 1993, Worst-case error bounds for parallel machine scheduling problems with bounded sequence-dependent setup times. *Operations Research Letters*, **14**, 251–256.

Papadimitriou, C. H. and Steiglitz, K., 1998, *Combinatorial Optimization: Algorithms and Complexity* (Mineola, New York: Dover).

Parker, R. G., Deane, R. H. and Holmes, R. A., 1976, On the use of a vehicle routing algorithm for the parallel processor problem with sequence dependent changeover costs. *AIIE Transactions*, **9**, 155–160.

Punnen, A. P. and Aneja, Y. P., 1995, Minmax combinatorial optimization. *European Journal of Operational Research*, **81**, 634–643.

Rajgopal, J. and Bidanda, B., 1991, On scheduling parallel machines with two setup classes. *International Journal of Production Research*, **29**, 2443–2458.

Reinelt, G., 1994, *The Traveling Salesman: Combinatorial Solutions for TSP Applications.* (Berlin: Springer-Verlag).

Sahni, S. K., 1976, Algorithms for scheduling independent tasks. *Journal of the ACM*, **23**, 116–127.

Sivrikaya-Serifoglu, F. and Ulusoy, G., 1999, Parallel machine scheduling with earliness and tardiness penalties. *Computers and Operations Research*, **26**, 773–787.

Tang, C. S., 1990, Scheduling batches on parallel machines with major and minor set-ups. *European Journal of Operational Research*, **46**, 28–37.

Zhang, G., 1997, A simple semi on-line algorithm for $P_2||C_{\max}$ with a buffer. *Information Processing Letters*, **61**, 145–148.