Computers & Industrial Engineering 72 (2014) 274–281

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Minimizing total tardiness for scheduling identical parallel machines with family setups

Jeffrey E. Schaller *

Eastern Connecticut State University, Department of Business Administration, 83 Windham ST., Willimantic, CT 06250, United States

ARTICLE INFO

Article history:
Received 22 August 2013
Received in revised form 6 February 2014
Accepted 2 April 2014
Available online 13 April 2014

Keywords:
Scheduling
Batch sizing
Heuristics
Parallel machines
Sequencing

ABSTRACT

This paper presents several procedures for scheduling identical parallel machines with family setups when the objective is to minimize total tardiness. These procedures are tested on several problem sets with varying numbers of families, jobs and machines, varying setup time distributions and various levels of due date tightness and variability. The results show that genetic algorithms are the most effective algorithms for the problem.

1. Introduction

In many operations the existence of changeover times, or setup times, on a machine motivates the grouping of jobs in order to obtain economies of scale. In scheduling, efficiencies that lead to economies of scale are gained by grouping similar jobs together. For example, jobs may belong to families where the jobs in each family tend to be similar in some way, such as their required tooling. As a result of this similarity, a job does not need a setup when following another job from the same family, but a known "family setup time" is required when a job follows a job that is a member of some other family. This is called a family scheduling model. Typically, there are a large number of jobs, but a relatively small number of families.

When a set of jobs is to be scheduled and need to be processed by a type of machine an important consideration is completing each job on or before the customer's due date. To help complete jobs in a timely manner there may be two or more identical machines of a type. These machines are referred to as identical parallel machines and the scheduling of jobs on the machines is referred to as parallel machine scheduling. To address this consideration, this paper seeks to identify methods for assigning and sequencing a set of jobs on identical parallel machines with significant family setup times that will minimize the total tardiness of the jobs. The tardiness of a job is defined as the completion time of the job minus the due date for the job if the job is completed after the due date and the tardiness is equal to zero if the job is completed on or before the due date. When jobs are scheduled in this environment, one approach is to sequence jobs in the same family as a batch on one of the machines (jobs in the same family are sequenced next to each other) to reduce setup time. The batching of jobs could cause some jobs to be processed before they are needed while at the same time delaying other jobs and causing them to be tardy. A second approach is to stop processing jobs in one family so a job in another family can be processed and completed by its due date. This causes an additional setup to be required and this additional setup increases the overall time to complete jobs that could have the effect of causing jobs that are at the end of the sequence on a machine to be very tardy. The trade-off just described causes this sequencing problem to be very challenging and as yet the only research that has been published for sequencing and scheduling in the described environment with an objective to minimize total tardiness was done by Shin and Leon (2004).

Formally, suppose there is a set of n jobs belonging to F setup families to be processed on M identical machines. Let p_j, $f(j)$, S_j, C_j, and d_j represent the processing time, the setup family, the setup time, the completion time, and the due date of job j ($j = 1, \ldots, n$) respectively. The tardiness of job j, T_j is defined as: $T_j = \max\{C_j - d_j, 0\}$, for $j = 1, \ldots, n$. The objective function, Z, can be expressed as: $Z = \sum_{j=1}^{n} T_j$. Also, note that if the job to be sequenced in position j of machine m($m = 1, \ldots, M$) is denoted as $[j]m$ then $C_{[j]m} = C_{[j-1]m} + p_{[j]}$ if $f[j]m = f\ [j-1]m$ and $C_{[j]m} = C_{[j-1]m} + S_{[j]} + p_{[j]}$ if $f\ [j]m \neq f\ [j-1]m$. This notation is summarized in Table 1.

* Tel.: +1 860 465 5226.
E-mail address: schallerj@easternct.edu

http://dx.doi.org/10.1016/j.cie.2014.04.001
0360-8352/

Five papers have addressed the problem of scheduling jobs on parallel machines to minimize total tardiness without family setups using branch-and-bound algorithms. Azizoglu and Kirca (1998) were the first to develop an optimal branch-and-bound algorithm. They developed several dominance properties and a lower bound that includes jobs that are not yet in a partial schedule. Their lower bound is based on a relaxed problem that allows jobs to be simultaneously processed on more than one machine. Yalaoui and Chu (2002) developed a branch-and-bound algorithm that included additional dominance properties and a new lower bound. Shim and Kim (2007) developed a branch-and-bound algorithm that included additional dominance properties and a lower bound that improves the lower bound developed by Azizoglu and Kirca (1998). Schaller (2009) shows how the lower bounds developed by Shim and Kim (2007) can be improved. Tanaka and Araki (2008) incorporated a Lagrangian relaxation into a branch-and-bound procedure for the problem. Sen, Sulek, and Dileepan (2003) include scheduling heuristics for the parallel machine problem to minimize total tardiness in their survey. More recently Biskup, Herrmann, and Gupta (2008) developed an insertion based heuristic for the problem.

Scheduling parallel machines when family setup times exist has been addressed for a variety of objectives such as minimizing flowtime (Azizoglu and Webster) and minimizing tardiness. Shin and Leon (2004) address the problem of minimizing total tardiness on parallel machines when family setup times exist. They developed a two-phase solution procedure. The first phase uses a MULTIFIT method based on the method developed by Coffman, Garey, and Johnson (1978) to develop an initial solution. The initial solution assigns jobs to machines and creates a sequence of jobs on the machines that can be defined in terms of batches of jobs from the same family. The second phase uses a tabu search to create an improved solution. Eom, Shin, Kwun, Shim, and Kim (2002) developed an efficient heuristic for minimizing weighted tardiness for parallel machine scheduling with sequence-dependent family setup times. Shin and Kang (2010) and Kang and Shin (2010) developed heuristic procedures that include total tardiness as an objective for parallel machine scheduling with rework and sequence-dependent setups. Chen and Chen (2008) use bottleneck-based heuristics to minimize the number of tardy jobs in a hybrid flexible flow line with unrelated parallel machines and Behnamian et al. (2009) developed a genetic algorithm for scheduling a hybrid flowshop with sequence-dependent setups and an objective that includes total tardiness.

Schutten (1996) showed that list schedules are dominant to minimize any regular cost function for parallel machine scheduling with sequence dependent setups if each job is assigned to the machine that it will complete the earliest instead of start the earliest. In this paper Schutten's (1996) property is used to develop tabu searches and genetic algorithms based on list schedules. The tabu searches, including the one developed by Shin and Leon (2004), are described in section two. In section three the genetic algorithms are described. A branch-and-bound algorithm to obtain optimal solutions for the problem is described in section four. The procedures were tested on randomly generated problems and the results of these tests are presented in section five. Section six concludes the paper.

Table 1
Summary of notation used throughout the paper.

Indexes	
j	Part index ($j = 1, \ldots, n$)
m	Machine index ($m = 1, \ldots, M$)
f	Setup family index ($f = 1, \ldots, F$)
Parameters	
n	Total number of parts
M	Total number of machines
F	Total number of families
p_j	Processing time of job j
$f(j)$	Setup family job j belongs to
S_j	Setup time for job j
d_j	Due date for job j
Decision variables	
C_j	Completion time of job j
T_j	Tardiness of job j
$[j]m$	Job sequenced in the jth position of machine m
$C_{[j]m}$	Completion time of the job sequenced in the jth position of machine m

2. Tabu searches

In this section four tabu searches are described for the problem. An initial solution is needed to start the tabu search procedures. Procedures for generating initial solutions are described in the next subsection.

2.1. Initial solution

The algorithm to develop an initial solution presented in this section was developed by Shin and Leon (2004). This algorithm uses a MULTIFIT method based on the method developed by Coffman et al. (1978) and is referred to as the GM algorithm. The algorithm schedules jobs within a family in EDD order and attempts to allocate families to machines so total tardiness is minimized. The GM algorithm is described in the appendix.

In this paper a modified version of the GM algorithm is proposed. This version is referred to as GM′. The GM′ algorithm is the same as the GM algorithm except when scheduling the jobs for a family on a machine. In the original GM version jobs belonging to the same batch are sequenced in EDD order. While EDD order would be appropriate in situations where due dates are relatively loose (most jobs are early) it would not be good if due dates are tight (most jobs are tardy). In fact if due dates are very tight and all of the jobs belonging to a batch would be tardy then SPT order would be appropriate. In order to address this issue the revised algorithm initially sequences jobs belonging to a batch in EDD order and then checks a condition developed by Emmons (1969) to identify exchanges of jobs that belong to the same family that could reduce tardiness. If two jobs j and k belong to the same family and are assigned to the same machine m and job k is currently sequenced before job j then the Emmon's condition is checked: if $p_j < p_k$ and $d_j < \max\{d_k, C_k\}$ then the positions of the two jobs on machine m are swapped. The following lemma states that if jobs j and k satisfy the above conditions then job j should precede job k. The following notation is used in the lemma. Let $\sigma(m)$ be a sequence of jobs on machine m in which job k precedes job j and $\sigma'(m)$ be a sequence of jobs on machine m that is the same as the sequence $\sigma(m)$ with the exception that the positions of jobs k and j are swapped. Let C_j and C_k be the completion times of jobs j and k in sequence $\sigma(m)$ and C'_j and C'_k be the completion times of jobs j and k in sequence $\sigma'(m)$.

Lemma. *If two jobs j and k belong to the same family and are to be assigned to the same machine m if $p_j < p_k$ and $d_j < \max\{d_k, C_k\}$ then the job j precedes job k in at least one optimal sequence.*

Proof. Let T_j and T_k be the tardiness of jobs j and k in sequence $\sigma(m)$ and T'_j and T'_k be the tardiness of jobs j and k in sequence $\sigma'(m)$. Let $B_{kj}(m)$ be the set of jobs that are sequenced before job k in $\sigma(m)$, $A_{kj}(m)$ be the set of jobs that are sequenced after job j in $\sigma(m)$, and $Q_{kj}(m)$ be the set of jobs that are sequenced between jobs k and j in $\sigma(m)$. Since jobs j and k belong to the same family, the total tardiness of the jobs in the sets $B_{kj}(m)$ and $A_{kj}(m)$ will

not change as a result of the swap of jobs k and j. Note that $C_j = C'_k$ and since $p_j \leqslant p_k$ then $C'_j \leqslant C_k$. The total tardiness of the jobs in the set $Q_{kj}(m)$ under sequence $\sigma'(m)$ will be less than or equal to the total tardiness of the jobs in the set $Q_{kj}(m)$ under sequence $\sigma(m)$. Therefore to prove this lemma it must be shown that $T'_j + T'_k \leqslant T_j + T_k$. There are two cases to consider: (1) $d_j \geqslant C_k$, (2) $d_j < C_k$. Case (1) $d_j \geqslant C_k$. By the conditions of the lemma $d_j < d_k$. Therefore $d_k \geqslant C_k$ and $T'_j = T_k = 0$. Since $C_j = C'_k$ and $d_j < d_k$, $i_j \geqslant T'_k$ and $T'_j + T'_k \leqslant T_j + T_k$ for this case. Case (2) $d_j < C_k$. $T_j = C'_k - d_j = (C'_k - C_k) + (C_k - d_j)$. Therefore $T_j + T_k = (C'_k - C_k) + (C_k - d_j) + T_k$. Since $T'_k \leqslant (C'_k - C_k) + T_k$ and $C'_j \leqslant C_k \Rightarrow T'_j \leqslant C_k - d_j$ then $T'_j + T'_k \leqslant T_j + T_k$ for this case.// □

Since the swaps identified by using Emmons conditions will not increase tardiness and could decrease tardiness it is hoped improved solutions will result by incorporating this procedure.

2.2. Shin and Leon procedure

Shin and Leon (2004) developed a tabu search procedure that attempts to improve the solution generated by the GM procedure. This procedure is referred to as the SHA procedure in this paper. A given schedule can be considered as sequences of batches of jobs that belong to the same family on each machine. Suppose there are K batches of jobs to be scheduled on M machines. A schedule can be expressed as $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_M\}$, where Π_m is the batch sequence for machine m. The sequence Π_m is expressed as $\{\pi(1, m), \pi(2, m), \ldots, \pi(N_m, m)$, where $\pi(k, m)$ denotes the index of the batch sequenced in position k on machine m and N_m denotes the number of batches scheduled on machine m. The Shin and Leon (2004) tabu search uses an insert move operator to create a neighborhood $N(\Pi_c)$ from a current schedule Π_c. The insert move operator picks a batch $\pi(k, m)$ in Π_c and inserts the batch into a new batch position $\pi(l, m')$ where $k \neq l$. All possible moves are used to create $N(\Pi_c)$. From these moves the best move is selected and the current solution is set equal to the best move and the procedure iterates to generate the next neighborhood. The procedure iterates until there are β non-improving consecutive moves and then stops. The tabu search keeps limited track of the search to prevent the search from staying in a local optimum. To do this a tabu list is used and a move (a move consists of moving a specific batch to a position on a machine) that is on the tabu list cannot be the best move. When determining the total tardiness of the batch sequences jobs within each batch are sequenced in EDD order. Each time the tabu search terminates the batch with the highest total tardiness is split and the tabu search is executed again. The SHA algorithm is as follows.

- Step 0 (generate an initial solution). Execute the GM algorithm to generate an initial solution. Set $a = 0$.
- Step 1 (tabu search).
- Step 2 (split a family batch). Split the family batch that has the highest value of total tardiness into two batches. If the time limit is exceeded then stop otherwise go to step 1.

When a family batch is split it is split into two batches consisting of the same number of jobs if the number of jobs in the original batch is even or one batch has one additional job if the number of jobs in the original batch is odd.

Shin and Leon (2004) fixed the size of the tabu list at 7 and demonstrated that as β increases and the number of times the tabu search is repeated increases the quality of solutions generated improves but the time required by the procedure also increases. Some experimentation was conducted in this research and based on this experimentation the size of the tabu list was set at 7 and β was selected to equal F^*5. The selection β in this research allows for a longer time with non-improving moves as the number of families is increased. Also, to compare this procedure with other procedures proposed in this research a time limit of n seconds is used to stop the procedure.

A second procedure is proposed that is referred to as the SHB procedure. The tabu search in the SHB procedure is essentially the same as the SHA procedure. There are two differences between the SHA and SHB procedures. First the SHB procedure uses the GM′ algorithm to obtain an initial solution (instead of the GM algorithm). The second change is the sequencing of jobs within each family batch each time a solution is developed. In the SHA procedure the jobs within a batch are sequenced in EDD order. In the SHB procedure jobs are first sequenced in EDD order and then the algorithm checks the jobs within each batch to see if Emmon's (1969) condition is violated and if so the positions of the jobs are exchanged. This is done because the sequencing of jobs within a batch also affects the total tardiness of the jobs in the batch and by incorporating Emmon's condition it is hoped solutions with reduced tardiness are generated.

2.3. A Tabu search based on list scheduling

Schutten (1996) showed that list schedules are dominant to minimize any regular cost function for parallel machine scheduling with sequence dependent setups if each job is assigned to the machine that it will complete the earliest instead of start the earliest. Since the family setups considered in this paper is a special kind of sequence dependent setup, Schutten's property applies. In this section a tabu search that works on a list of jobs is described. This algorithm is referred to as the L algorithm in this paper. First the GM′ algorithm is used to obtain an initial solution. This solution is then converted to a list by sorting the jobs in non-decreasing completion time order.

Let δ_c be the current list. The neighborhood of the list $N(\delta_c)$ is created by employing two move operators. The insert move operator takes a job and inserts it into a different position in the list than it was in the current list. The exchange move operator takes two jobs and exchanges their positions to create a different list. All possible moves are used to create $N(\delta_c)$. A schedule is created from a list by assigning jobs to the machine it will complete the earliest (Schutten's, 1996 property). Jobs are assigned in the order that they are on the list. For the sequence of jobs on each machine the algorithm checks that Emmon's (1969) condition is met for jobs belonging to the same family. If two jobs from the same family are assigned to the same machine and they are not sequenced to meet Emmon's condition their positions are exchanged.

This search also uses a tabu list to prevent the search from staying in a local optimum. Based on preliminary experiments the tabu list size was fixed at 7 as was the case with the SHA and SHB procedures. In order to compare this procedure with other procedures proposed in this research a time limit of n seconds is used to stop the procedure.

2.4. A combined approach

The fourth tabu search combines both the list scheduling tabu search and a local search based on batches. This algorithm is referred to as the LB procedure in this paper. The GM′ procedure is used to obtain an initial solution. Then the tabu search procedure starts. In this tabu search procedure two local searches are used: first, the neighborhood of the list from the L procedure is searched, then, a local batch insertion search is used to see if the solution can be improved. The local batch search is referred to as BI. Each batch is removed from the sequence of batches and then inserted into each possible position on each machine while maintaining the

relative order of the other batches. The final position and machine of the batch is the position and machine that results in the lowest total tardiness. This search set the parameters to be the same as the L procedure. The tabu list size was set to 7 and the time limit was set to n seconds.

3. Genetic algorithms

A genetic algorithm is a meta-heuristic search procedure that uses a multiple solution search technique. This approach has been found to quickly generate good solutions for a wide variety of scheduling problems. Vallada and Ruiz (2010) and Singh (2010) have successfully used genetic algorithms for scheduling problems that had an objective of minimizing a function based on the due dates of the individual jobs. In a genetic algorithm an initial population of chromosomes is first created and then successive populations (or generations) of chromosomes are created using some methodology until a stopping condition is met. A chromosome corresponds to a solution for the problem. For this problem solutions can be represented by a list of the jobs. The list of jobs is converted to a solution by starting at the front of the list and scheduling each job on the machine it will be completed earliest (Schutten's, 1996 property). The jth gene in a chromosome corresponds to the job in the jth position of a list.

Two genetic algorithms are proposed in this research. Elements of these genetic algorithms include initialization of the population, a selection mechanism, a crossover operator, mutation operator, generational scheme, local searches, and the stopping condition.

3.1. Initial population

An initial population of (population size) chromosomes (lists) is created. The GM′ algorithm described in Section 2.1 is used to create the first chromosome. The other chromosomes are randomly generated. Each chromosome (list) is created by first generating a random number between 0 and 1 for each job and then sorting the numbers corresponding to each job (lowest to highest) to create the list of jobs (chromosome).

3.2. Selection operator

The genetic algorithms in this research use a selection operator called n-tournament. With this approach a percentage of individuals, based on a parameter called "pressure", are selected and the individual with the lowest total tardiness among these individuals is selected for the mating process.

3.3. Mating operator

Uniform order-based (UOB) crossover is used as the mating operator. The goal of this operator is to generate two good individuals, called offspring, from two progenitors (chromosomes) that were selected using the selection operator described in the previous subsection. The UOB crossover operator tries to copy jobs at each position from the first parent with a probability p_{u1} to a child. The unfilled positions are then filled from the unused jobs in the order they appear in the second parent. The operator then creates a second child by attempting to copy jobs at each position from the second parent with a probability p_{u2} to a child. The unfilled positions are then filled from the unused jobs in the order they appear in the first parent. The value of p_{u1} is set at $T(p_2)/(T(p_1)+T(p_2))$ and p_{u2}, at $T(p_1)/(T(p_1)+T(p_2))$, where $T(p_1)$ and $T(p_2)$ are the total tardiness associated with the sequences of parents 1 and 2, respectively. With this approach better parents tend to have more jobs copied to children.

For example consider the sequences associated with the following two parents:

Parent 1 1 – 2 – 4 – 6 – 5 – 3
Parent 2 4 – 5 – 2 – 3 – 1 – 6
If after the first step child 1 inherits positions 1, 3 and 4 from the first parent so that the partially formed child is (1, _, 4, 6, _, _) then the resulting sequence for the first child is:
1 – 5 – 4 – 6 – 2 – 3.

The genetic algorithm in this research performs the mating operator with the two selected parents with P_U probability.

3.4. Mutation operator

All of the genetic algorithms in this research use a shift mutation operator. With this operator each job in a permutation sequence is extracted with a P_m probability and is inserted in a randomly selected different position in the sequence.

3.5. Local searches

Two local searches are used in the genetic algorithms proposed in this research. The first local search is the job insertion search used in the L and LB tabu searches. This search is referred to as JI. In this search a job is removed from the list and is inserted into all n positions. The final position of the job is the position that results in the lowest total tardiness. This search is carried out in all n jobs of each generated offspring after the mating and mutation operators have been applied with a P_{LS} probability.

The second local search is the batch insertion search used in the LB tabu search. This local search is referred to as BI. Each batch is removed from the sequence of batches and then inserted into each possible position on each machine while maintaining the relative order of the other batches. The final position and machine of the batch is the position and machine that results in the lowest total tardiness. If this search is incorporated in an algorithm it is carried out after the JI local search has been performed.

3.6. Generational scheme

The genetic algorithms proposed in this research have two criteria for accepting generated offspring into the population. (1) An offspring must be better (have a lower total tardiness) than the worst individual in the population. (2) The offspring has a unique list (there are no other individuals in the population with the same list).

3.7. Stopping criteria

The proposed genetic algorithms in this research have two stopping criteria. (1) If a solution is found with zero tardiness the procedure stops. (2) If a time limit is exceeded when a generation is completed the procedure stops.

3.8. Genetic algorithms

Two genetic algorithms are proposed in this research. The first is referred to as GA in this paper and includes all the elements described in Sections 3.1–3.7 with the exception of the BI local search described in Section 3.5. The second genetic algorithm is referred to as GAB and includes the BI local search as well as the other elements described in Sections 3.1–3.7.

Parameters needed for these genetic algorithms are the population size, pressure for selection, probability for applying the uniform-order crossover operator P_U, probability for mutation P_m, the

probability for carrying out local searches, the stopping criteria. Based on preliminary experiments the parameters were set as follows. Population size 30, pressure 30%, P_U 30%, P_m 2%, P_{LS} 15%. In order to compare this procedure with other procedures proposed in this research a time limit of n seconds is used to stop the procedure.

4. Optimal branch-and-bound procedure

In this section a branch and bound procedure for the problem is described. This procedure is based on Azizoglu and Kirca's (1998) algorithm for the problem without family setups. Schutten (1996) showed that list schedules are dominant for this problem so the branch-and-bound algorithm implicitly enumerates permutations (lists or sequences) of jobs. A schedule can be obtained by assigning jobs to the machine they will complete the earliest on. At the conclusion of the procedure a sequence of jobs and a schedule is obtained resulting in the minimum total tardiness for the problem. This procedure is referred to as O.

In the O procedure a list is constructed starting at the beginning and working to the end so a node at the pth level in the branch-and-bound tree corresponds to a partial list (and the associated sub-problem) for the first p jobs in a list. Branching adds a job to the first unassigned position (at the end) in a partial list. From a node at level p, up to $v(v = n - p)$ branches may be generated, one for each job not in the partial list corresponding to the level-p node from which branching occurs.

For each of the nodes generated in the procedure, a lower bound on the sum of tardiness of all jobs is computed. Nodes with lower bounds that are greater than the current incumbent solution are removed from further consideration. If the lower bound is less than the incumbent value and the node represents a complete sequence then the incumbent value is updated to equal the lower bound. A depth-first branching rule is used so the node with the most jobs in the corresponding partial sequence is selected for branching. Ties are broken in favor of the node with the minimum lower bound.

If we have a partial list we can obtain a lower bound on the total tardiness that will result from the completion of the list by creating a schedule for the jobs in the partial list and then computing the total tardiness of these jobs. Azizoglu and Kirca (1998) show how to strengthen this lower bound by including jobs that are not included in the partial list. Let σ be a partial schedule obtained from a partial list. Let σ' be the set of jobs not included in σ. Let $d_{EDD[j]}$ equal the due date of the jth job if the jobs in σ' are sorted in earliest due date order ($d_{EDD[a]} \leqslant d_{EDD[b]}$ if $a < b$). Let $p_{SPT[j]}(\sigma')$ equal the processing time of the job in the jth position of a sequence formed by sorting the jobs in shortest processing time order ($p_{SPT[a]}(\sigma') \leqslant p_{SPT[b]}(\sigma')$ if $a < b$). Let PT(m) be the completion time of the last job scheduled on machine m in the partial schedule σ. Let n' be the number of jobs in σ'. Azizoglu and Kirca (1998) show that without family setups a lower bound on the tardiness of the jobs in σ' is:

$$\sum_{k=1}^{n'} \max\left\{0, \left\{\min_m\{PT(m)\} + \left(\sum_{j=1}^{k} p_{SPT[j]}(\sigma')\right) \Big/ M\right\} - d_{EDD[k]}\right\}$$

In order to extend the above bound when there are family setups let n'_f equal the number of jobs in σ' that belong to family f. For each job in σ' a modified processing time (mp_j) is defined: $mp_j = p_j + \lfloor \frac{Sf(j)}{n'f(j)} \rfloor$. Let $mp_{SPT[j]}(\sigma')$ equal the mp time of the job in the jth position of a sequence formed by sorting the jobs in shortest mp time order ($mp_{SPT[a]}(\sigma') \leqslant mp_{SPTb]}(\sigma')$ if $a < b$). A lower bound on the tardiness of the jobs in σ' can then be computed:

$$\sum_{k=1}^{n'} \max\left\{0, \min_m\{PT(m)\} + \left(\sum_{j=1}^{k} mp_{SPT[j]}(\sigma')\right) \Big/ M - d_{EDD[k]}\right\}.$$

5. Computational tests of the heuristic procedures

5.1. Data and performance measures

The heuristic procedures described in sections two and three were tested on problems of various sizes in terms of the numbers of jobs (n), numbers of families (F), numbers of machines (M), for four sets of distributions of due date range and tightness, and two sets of distributions for family setup times. Each problem set consists of 10 problems. The problems within a problem set have the same number of jobs, the same number of families, the same number of machines, family setup times are drawn from the same distribution, and the due dates for the jobs are generated using the same distribution.

Two classes of problem sets were tested. The problem sets of the first class were small sized problems. The numbers of jobs tested in this problem class were $n = 8$, 10, 12, and 14, the number of families tested were $F = 3$ and 4 and numbers of machines, $M = 2$ and 3. The problem sets of the second class were larger size problems. Two levels of number of jobs were tested for this class of problems: $n = 30$ and 50, two levels of families: $F = 3$ and 6, and three levels of machines: $M = 2$, 6 and 10.

When generating the individual problems the processing times of the jobs were generated using a uniform distribution over the integers 1 and 50. The due dates for the jobs were also randomly generated using a procedure similar to one used by Potts and VanWassenhove for the single-machine problem. The due dates for the jobs were generated using a uniform distribution over the integers AP $(1 - r - R/2)$ and AP $(1 - r + R/2)$, where AP is the total processing time for all n jobs plus the total setup time for all F families divided by the number of machines ($AP = \sum_{f=1}^{F} S_f/M + \sum_{j=1}^{n} p_j/M$), and R and r are two parameters called due date range and tardiness factors. Four sets of these parameters were used for each F, M and n combination:

1. $R = 0.5$ and $r = 0.5$ (set 1).
2. $R = 1.00$ and $r = 0.5$ (set 2).
3. $R = 0.5$ and $r = 0.25$ (set 3).
4. $R = 1.00$ and $r = 0.25$ (set 4).

These four sets of parameters were chosen to see if the tightness and range of due dates would affect the relative performance of the procedures to be tested. Two levels of due date tightness are included: one relatively tight (0.5) and one relatively loose (0.25) and two levels of due date range: one with due dates relatively close to each other (0.5) and one with a relatively larger range (1.0) to create the four sets of parameters above.

The setup times for each family were randomly generated using a uniform distribution. Two setup distributions were used. The first setup distribution (1) generated setups over the integers 1 and 100 and the second (2), over the integers 1 and 50. Each setup distribution was used with each combination of F, M, n, and due date parameters. Each of the tabu search procedures and genetic algorithms were run for each of the problem sets for both the small and larger sized problem classes. The O procedure was run for the problem sets in the small sized problem class. Since the time required by the O procedure increases rapidly as the number of jobs increases it was not practical to use the O procedure for the large sized problem class.

The procedures were coded in Turbo Pascal and were tested on a Dell Inspiron 1525 1.6 GHz Lap Top computer to obtain the objective value of the solution generated for each problem. The measures of performance used to evaluate the procedures for the test of the problems in the small size problem class are the percentage error (% Error) of the total tardiness of the solution generated by

Table 2
Average % error from the optimal total tardiness and number of optimal solutions (in parentheses).

Problem size			Procedure					
n	*F*	*M*	SHA	SHB	L	LB	GA	GAB
8	3	2	1.79 (69)	2.17 (67)	0.38 (74)	0.69 (71)	0.00 (80)	0.00 (80)
8	3	3	0.34 (71)	0.41 (69)	0.11 (79)	0.11 (79)	0.00 (80)	0.00 (80)
8	4	2	4.30 (57)	5.53 (59)	2.60 (69)	1.98 (71)	0.03 (79)	0.00 (80)
8	4	3	1.76 (61)	1.62 (61)	0.20 (77)	0.20 (77)	0.00 (80)	0.00 (80)
10	3	2	3.21 (57)	2.22 (64)	3.90 (62)	1.88 (64)	0.00 (80)	0.00 (80)
10	3	3	1.24 (64)	1.30 (62)	0.11 (77)	0.09 (77)	0.00 (80)	0.00 (80)
10	4	2	5.77 (56)	3.73 (59)	3.70 (60)	2.51 (65)	0.12 (78)	0.00 (80)
10	4	3	2.60 (53)	2.31 (59)	0.87 (68)	0.73 (71)	0.00 (80)	0.00 (80)
12	3	2	5.12 (51)	5.07 (52)	4.61 (53)	3.91 (57)	1.33 (74)	0.04 (79)
12	3	3	4.29 (46)	1.95 (51)	0.72 (72)	0.68 (72)	0.02 (79)	0.03 (78)
12	4	2	6.66 (50)	7.17 (52)	9.02 (48)	6.20 (54)	0.54 (74)	0.07 (79)
12	4	3	5.78 (53)	4.85 (58)	0.77 (67)	0.85 (67)	0.23 (77)	0.04 (78)
14	3	2	11.79 (40)	10.78 (46)	5.19 (58)	5.04 (58)	0.09 (78)	0.00 (80)
14	3	3	4.96 (45)	3.77 (51)	1.29 (70)	1.22 (70)	0.22 (79)	0.00 (80)
14	4	2	10.41 (35)	8.66 (41)	9.38 (50)	6.26 (58)	0.95 (70)	0.05 (79)
14	4	3	6.10 (39)	6.31 (40)	2.82 (60)	2.19 (60)	0.33 (72)	0.26 (76)
Average			4.76 (52.94)	4.23 (55.69)	2.85 (65.25)	2.16 (66.94)	0.24 (77.50)	0.03 (79.31)

Table 3
Average percent vs. best and number of best solutions (in parentheses).

Problem size			Procedure					
n	*F*	*M*	SHA	SHB	L	LB	GA	GAB
30	3	2	23.72 (16)	22.45 (18)	20.44 (21)	15.89 (24)	7.29 (31)	1.20 (64)
30	3	6	11.59 (1)	10.55 (1)	1.30 (45)	1.24 (46)	1.21 (44)	0.89 (59)
30	3	10	3.76 (2)	4.65 (1)	0.44 (55)	0.44 (55)	0.31 (57)	0.06 (73)
30	6	2	26.98 (7)	27.17 (7)	32.77 (3)	26.95 (3)	23.52 (6)	0.68 (70)
30	6	6	9.66 (10)	7.65 (17)	3.08 (37)	3.00 (35)	4.53 (27)	1.92 (42)
30	6	10	5.12 (7)	5.23 (6)	0.76 (50)	0.76 (50)	1.12 (34)	0.59 (51)
50	3	2	27.44 (14)	28.37 (14)	25.42 (17)	26.19 (18)	14.65 (22)	4.55 (54)
50	3	6	35.99 (0)	32.14 (0)	1.36 (37)	2.64 (38)	7.16 (25)	4.83 (35)
50	3	10	20.30 (0)	27.39 (0)	0.93 (36)	0.83 (33)	1.85 (25)	0.92 (45)
50	6	2	26.55 (9)	32.07 (6)	33.84 (3)	27.82 (10)	28.03 (6)	5.07 (54)
50	6	6	15.95 (7)	15.16 (8)	4.36 (47)	3.31 (41)	17.28 (11)	11.21 (26)
50	6	10	24.44 (1)	21.55 (1)	1.00 (47)	1.08 (48)	5.50 (8)	4.34 (26)
Average			19.29 (6.17)	19.53 (6.58)	10.48 (33.17)	9.18 (33.42)	9.37 (24.67)	3.02 (49.92)

each procedure compared to the optimal total tardiness for the problem and the number of optimal solutions generated by each procedure. The percentage error calculates the percentage the optimal total tardiness is less than the total tardiness obtained by using the heuristic procedures. % Error = $[(Z_h - Z_O)/Z_h]^*100$, where Z_O = the optimal total tardiness for the problem, and Z_h = the total tardiness of the solutions generated by the heuristic procedures. The measures of performance used to evaluate the procedures for the test of the problems in the larger size problem class are the percentage deviation compared to the best solution (percent vs. best), and the number of times each procedure generated the best solution. The percentage vs. best = $[(Z_h - Z_B)/Z_h]^*100$, where Z_B = the lowest total tardiness of the solutions and Z_h = the total tardiness of the solutions generated by the heuristic procedures. Since the best solution generated among the heuristics procedures will have a total tardiness that is at least as large as the total tardiness of an optimal solution the percent vs. best measure underestimates the % Error of the solutions generated by heuristics when compared to an optimal solution.

5.2. Results for the small sized problems

Table 2 shows the average percentage an optimal solution's total tardiness is less than the total tardiness for each procedure for each problem size and the number of times each procedure generated an optimal solution for each problem size. The last row of the table shows the averages across all of the problem sizes.

The results show the GAB procedure's average % error was under 0.30% for all of the 16 problem sizes and had the lowest average % error. The GA and LB procedures also performed well on these problems with average % errors of less than 1.40% and 6.35% respectively for all of the problem sizes and averaged less than 2.5%. The L procedure's average % error was less than 10% for all problem sizes but greater than 5% for three problem sizes and the SHA and SHB procedures did not perform as well and had average % errors that frequently exceeded 5% and in some cases exceeded 10%. The results also show that including the batch insertion local search in the list tabu search and the genetic algorithm frequently improved the solutions. The LB procedure had an overall lower average % error than the L procedure and the GAB procedure had an overall lower average % error than the GA procedure. Incorporating Emmon's (1969) dominance condition into Shin and Leon's (2004) procedure also improved the results. The SHB procedure had a lower average % error (4.23) than the SHA procedure (4.76) and found more optimal solutions.

The GAB and GA procedures found the optimal solution for over 95% of the problems, the LB and L procedures, 80% of the problems and the SHA and SHB procedures were only able to find the optimal solutions for less than 70% of the problems.

5.3. Results for the large sized problems

Table 3 shows the average percentage vs. best for each procedure for each problem size and the number of times each procedure

generated the best solution for each problem size. The last row of the table shows the averages across all of the problem sizes.

The results show that the GAB procedure had the lowest average percentage vs. best (3.02%) and generated the most solutions that were best among the heuristic procedures. The GAB procedure was also the only procedure that had an average percentage vs. best that was under 12% for all the problem sizes. The GA and LB procedures also had an average percentage vs. best that was under 10%. The SHA and SHB procedures again performed poorly and had percentage vs. best's over 19%. The results again show that including the batch insertion local search in the list tabu search and the genetic algorithm frequently improved the solutions. The LB procedure had a lower average percentage vs. best than the L procedure on most of the problem sizes and the GAB procedure improved the average percentage vs. best relative to the GA procedure for all 12 problem sizes.

The results show that as problem sizes increase the percentage vs. best generally increases. This is also true for the GAB procedure. Since the best solution among the heuristic procedures will have a total tardiness that is at least as large as the total tardiness of an optimal solution the percentage vs. best underestimates the % error compared to an optimal solution and the number of times the best solution was generated over estimates the number of times a procedure generates a solution that is optimal. Therefore these results show that the quality of solutions generated by the GAB procedure does deteriorate as the problem size is increased. However, the GAB procedure consistently demonstrated that it generates good solutions for both small sized and large sized problems compared to the other heuristic procedures tested and therefore the GAB procedure is recommended for the problem.

6. Conclusions

This paper considered the problem of scheduling on parallel machines with family setup times to minimize total tardiness. Three new tabu searches and two genetic algorithms were proposed for this problem. These tabu searches, genetic algorithms and the tabu search procedure proposed by Shin and Leon (2004) were tested on randomly generated problems. The results of the test showed that there were many instances when the tabu search proposed by Shin and Leon (2004) and a variation of this tabu search did not generate good solutions. The results also showed that a genetic algorithm that also incorporates local searches on jobs and batches of jobs (GAB) consistently generated good solutions. It is therefore recommended that the GAB procedure be used for this problem.

Appendix A. GM algorithm

The following notation is used in the GM algorithm. Let W_f equal the total processing time of family f ($W_f = \sum_{j \in f} p_j$). Let A_j equal the total processing time plus setup time for family f ($A_f = W_f + S_f$). Let TT_f equal the total tardiness of the jobs in family f. Let Q be the set of families that need to be scheduled. The algorithm schedules jobs within a family in EDD order and attempts to allocate families to machines so total tardiness is minimized. The algorithm repeatedly executes a rule referred to as the m-FFD rule for various guidance values for urgency u and capacity q. For a specific value for u and q, the m-FFD rule checks if it is feasible to assign families to the M machines without exceeding the u and q values. Let r_m and y_m be the remaining urgency and machine time on machine m respectively. The algorithm follows.

Step 0 (initialize). List the jobs in each family f according to the earliest due date rule (EDD). Set $r_m = u$ and $y_m = q$ for each machine. Set $m = 0$ and set $B = Q$.

Step 1 (check feasibility). Set $m = m + 1$. If $m > M$ then go to step 4.

Step 2 (m-FFD rule). Go to step 3 if $TT_f > r_m$ for all $f \in B$. Otherwise set $g = \arg\max_{f \in B} \{TT_f | TT_f \leqslant r_m, A_f \leqslant y_m\}$. If $g =$, assign family g to machine m by scheduling the jobs in family g, adding the jobs in family g to the end of the schedule for machine m in EDD order. Set $y_m = y_m - A_g$ and $r_m = r_m - TT_g$. Set $B = B/g$. If B is empty then stop otherwise repeat step 2.

Step 3 (split a family). Set $h = \arg\min_{f \in B} S_f$. If $S_h \geqslant r_m$ or $S_h \geqslant y_m$ then go to step 1 otherwise allocate jobs in family h to machine m and update A_h and TT_h. Go to step 1.

Step 4 (guide next loop) Update the values of u and q using a binary search procedure.

In the procedure above the m-FFD rule is executed repeatedly for different values of u and q. A binary search over the intervals $[L_u, U_u]$ and $[L_q, U_q]$ is conducted where L and U are the lower and upper limits on the values of u and q. Initially L_u is set equal to 0. To set U_u jobs are listed in EDD order and scheduled using the list on the machine they will complete earliest. The total tardiness resulting from this schedule is divided by M and then U_u is set equal to the result. Initially u is set equal to $(1/2)(L_u + U_u)$. If m-FFD finds a feasible schedule then U_u is set to u otherwise L_u is set to $u + 1$. The value for u is then set to $(1/2)(L_u + U_u)$ for the next execution of m-FFD. To set the value for q, L_q is initialized to $\sum_f A_f/M$ and U_q is initialized to $\sum_f A_f$. As the algorithm executes the values for q are adjusted in the same manner described above for u.

References

Azizoglu, M., & Kirca, O. (1998). Tardiness minimization on parallel machines. *International Journal of Production Economics, 55*, 163–168.

Azizoglu, M., & Webster, S. (2003). Scheduling parallel machines to minimize weighted flowtime with family set-up times. *International Journal of Production Research, 41*, 1199–1215.

Behnamian, J., Ghomi, F., & Zandieh, M. (2009). A multi-phase covering Pareto-optimal front method to multi-objective scheduling in a realistic hybrid flowshop using a hybrid metaheuristic. *International Journal of Production Research, 48*, 4949–4976.

Biskup, D., Herrmann, J., & Gupta, J. N. D. (2008). Scheduling identical parallel machines to minimize total tardiness. *International Journal of Production Economics, 115*, 134–142.

Chen, C. L., & Chen, C. L. (2008). Bottle-neck-based heuristics to minimize tardy jobs in a flexible flow line with unrelated parallel machines. *International Journal of Production Research, 46*, 6415–6430.

Coffman, E. G., Garey, M., & Johnson, D. S. (1978). An application of bin-packing to multiprocessor scheduling. *SIAM Journal of Computing, 7*, 1–16.

Emmons, H. (1969). One-machine sequencing to minimize certain functions of job tardiness. *Operations Research, 17*, 701–715.

Eom, D. H., Shin, H. J., Kwun, I. H., Shim, J. K., & Kim, S. S. (2002). Scheduling jobs on parallel machines with sequence-dependent family set-up times. *The International Journal of Advanced Manufacturing Technology, 19*, 926–932.

Kang, H. K., & Shin, H. J. (2010). An adaptive scheduling algorithm for a parallel machine problem with rework processes. *International Journal of Production Research, 48*, 95–115.

Potts, C. N., & VanWassenhove, L. A. (1985). A branch-and-bound algorithm for the total weighted tardiness problem. *Operations Research, 33*, 363–377.

Schaller, J. E. (2009). Note on Shim and Kim's lower bounds for scheduling on identical parallel machines to minimize total tardiness. *European Journal of Operational Research, 197*, 422–426.

Schutten, J. M. J. (1996). List scheduling revisited. *Operations Research Letters, 18*, 167–170.

Sen, T., Sulek, J. M., & Dileepan, P. (2003). Static scheduling research to minimize weighted and unweighted tardiness: A state-of-the-art survey. *International Journal of Production Economics, 83*, 1–12.

Shim, S. O., & Kim, Y. D. (2007). Scheduling on parallel identical machines to minimize total tardiness. *European Journal of Operational Research, 177*, 135–146.

Shin, H. J., & Kang, Y. H. (2010). A rework-based dispatching algorithm for module process in TFT-LCD manufacture. *International Journal of Production Research, 48*, 915–931.

Shin, H. J., & Leon, V. J. (2004). Scheduling with product family set-up times: An application in TFT LCD manufacturing. *International Journal of Production Research, 42*, 4235–4248.

Singh, A. (2010). A hybrid permutation-coded evolutionary algorithm for the early/tardy scheduling problem. *Asia-Pacific Journal of Operational Research, 27*, 713–725.

Tanaka, S., & Araki, M. (2008). A branch-and-bound algorithm with Lagrangian relaxation to minimize total tardiness on identical parallel machines. *International Journal of Production Economics, 113*, 446–458.

Vallada, E., & Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega, 38*, 57–67.

Yalaoui, F., & Chu, C. (2002). Parallel machine scheduling to minimize total tardiness. *International Journal of Production Economics, 76*, 265–279.