Single Machine Group Scheduling with Setups to Minimize Total Tardiness

Samarn Chantaravarapan

Production Modeling Corporation (PMC)
Dearborn, MI 48126
schantar@pmcorp.com

Jatinder N.D. Gupta
(Corresponding Author)

College of Administrative Sciences
University of Alabama in Huntsville
Huntsville, AL 35899
Telephone: 256-824-6593
FAX: 256-824-2929
guptaj@uah.edu

June 10, 2004

ABSTRACT

This paper considers the single machine scheduling problem with independent family (group) setup times where jobs in each family are processed together. A sequence independent setup is required to process a job from a different family. The objective is to minimize total tardiness. A mixed integer programming model capable of solving small sized problems is described. Several heuristic algorithms are proposed and empirically evaluated as to their effectiveness in finding optimal schedules. These results show that several heuristic algorithms generate solutions that are quite close to the optimal solutions.

Keywords: *Single machine scheduling, family setups, minimizing tardiness, heuristic algorithms, empirical results.*

1. INTRODUCTION

Group scheduling problems, where all jobs of the same family must be scheduled together, have attracted numerous researchers due to their frequent real-life occurrence. Many manufacturers have implemented the concept of group technology (GT) in order to reduce setup costs, lead times, work-in-process inventory costs, and material handling costs. Group technology consists of dividing the total set of jobs into several subsets, called families, where a family is a subset of jobs that have similar requirements in terms of tooling and setups. Since different job families require different tooling, a setup is often necessary when a job from a new family is to be produced. Since jobs are assigned to families based on tooling and setup requirements, there is usually a negligible or minor setup to change from one part to another within the same family. Because there is a major setup change between part families, there is an advantage to processing parts belonging to the same family as a group. This is a key difference between the group scheduling problem and the traditional scheduling problem. In order to address this issue of major setups between families, an optimum job sequence within each family as well as an optimal family sequence need to be developed to optimize a given performance measure.

With current emphasis on customer service and meeting the promised delivery dates, major thrust in scheduling research is being directed towards improving performance with respect to due dates. Therefore, a widely used performance measure in scheduling research is total tardiness where the tardiness of a job is defined as the time delay in delivering a job to the customer. Under the tardiness criterion, there is no benefit gained from completing jobs early, and a delay penalty incurs when a job is tardy.

Reviews of recent research in solving scheduling problems with family setups by Monma and Potts (1989), Potts and van Wassenhove (1992), Potts and Kovalyov (2000), and Webster and Baker (1995) show that majority of studies in group scheduling problems are involved with total (weighted) flow-time and maximum tardiness. For example, Cheng, Gordon, and Kovalyov (1996) presented a polynomial time algorithm with the aim of minimizing the maximum cost of a schedule subject to minimum total weighted flow-time. To the best of our knowledge, however, there is no published work available on the single-machine group scheduling problem to minimize total tardiness. Therefore, this paper considers the single machine group scheduling with sequence independent family setup times

and it develops a mixed integer programming model and several heuristic algorithms to find schedules with minimum total tardiness.

The rest of the paper is organized as follows: section 2 describes the problem, comments on its complexity and develops a mixed integer programming model of the problem. Several heuristic algorithms, which include extensions and modifications of existing algorithms for the single machine total tardiness problem, are proposed and discussed in section 3. Results of computational tests to evaluate the performance of these heuristic algorithms are reported in section 4. Finally, section 5 discusses the main results and describes some fruitful directions for future research.

2. PROBLEM FORMULATION AND COMPLEXITY

The single machine group scheduling problem with family setup times can be stated as follows: A given number of families, denoted by f, and the number of jobs in each family, represented by n_i, for family $i = 1, \ldots, f$ is to be processed, without interruption or preemption on a single machine. The processing time and the due date of the j^{th} job from family i are defined by p_{ij} and d_{ij}, respectively. Furthermore, if a job follows the preceding job from the same family, then there is no setup time between them; otherwise, the family setup time s_i is required before the next process. Note that s_i is sequence independent. That is, the family setup time depends on the following family only. Additionally, it is assumed that there is a setup prior to the first job in any sequence. All jobs are available at time zero, and machine idle time and job preemption are prohibited. A machine processes at most one job at a time and it cannot process any job while a setup proceeds. All jobs in each family must be scheduled together. In other words, there are only f setups in any feasible sequence. Moreover, if a job is finished before or on its due date, there is no tardiness. Otherwise, the tardiness incurred is given by $T_{ij} = C_{ij} - d_{ij}$, where T_{ij} and C_{ij} represent tardiness and completion time of job j from family i, respectively. Thus, tardiness can be defined by $T_{ij} = \max \{0, C_{ij} - d_{ij}\}$. The total number of jobs is $n = n_1 + n_2 + \ldots + n_f$.

With the above definitions, the single machine group scheduling problem considered here is one of finding a schedule of n jobs such that the total tardiness of all jobs is minimum. When each family contains only one job, the traditional single machine total tardiness problem is known to be binary (in ordinary sense) NP-hard (Du and Leung, 1991). Therefore, the group scheduling problem considered in this paper is least binary NP-hard. The question as to its NP-hardness status in the unary

sense remains open. In view of the NP-hard nature of the problem, a mixed integer programming model of the problem is proposed here. To do so, we first define the notations used in the formulation.

Notations

X_{ijk} $= \begin{cases} 1, & \text{if job } j \text{ from family } i \text{ is placed in position } k. \\ 0, & \text{otherwise.} \end{cases}$

$Y_{i,k}$ $= \begin{cases} 1, & \text{if } s_i \text{ is needed before a job at position } k. \\ 0, & \text{otherwise.} \end{cases}$

f = number of families.

n_i = number of jobs in family i.

n = total number of jobs = $n_1 + n_2 + \ldots + n_f$.

C_k = completion time of the job at position k.

d_{ij} = due date of the j^{th} job in family i.

p_{ij} = processing time of the j^{th} job in family i.

T_k = tardiness of the job at position k.

s_i = family setup time of family i.

Formulation

Objective Function

$$\text{Min} \quad Z = \sum_{k=1}^{n} T_k$$

Subject to

$$\sum_{i=1}^{f} \sum_{j=1}^{n_i} X_{ijk} = 1 \qquad k = 1, 2, ..., n \qquad (1)$$

$$\sum_{k=1}^{n} X_{ijk} = 1 \qquad j = 1, 2, ..., n_i, i = 1, 2, ..., f \qquad (2)$$

$$\sum_{j=1}^{n_i} X_{ijk} = Y_{i,1} \qquad i = 1, 2, ..., f \qquad (3)$$

$$\sum_{j=1}^{n_i} X_{ijk} + \sum_{j=1}^{n_p} \sum_{\substack{p \in \{1,\ldots f\} \\ -\{i\}}} X_{pj(k-1)} - Y_{i,k} \le 1 \qquad k = 2, 3, ..., n, i = 1,2,..., f \qquad (4)$$

$$C_1 = \sum_{i=1}^{f} s_i Y_{i,1} + \sum_{i=1}^{f} \sum_{j=1}^{n_i} p_{ij} . X_{ij1} \qquad (5)$$

$$C_k = C_{k-1} + \sum_{i=1}^{f} s_i Y_{i,k} + \sum_{i=1}^{f} \sum_{j=1}^{n_i} p_{ij} . X_{ijk} \qquad k = 2, 3, ..., n \qquad (6)$$

$$C_k - \sum_{i=1}^{f}\sum_{j=1}^{n_i} d_{ij}.X_{ijk} \le T_k \qquad k = 1, 2, ..., n \qquad (7)$$

$$\sum_{k=1}^{n_i} Y_{i,k} = 1 \qquad i = 1,2,..., f \qquad (8)$$

$$X_{ijk} = 0,1 \qquad j = 1, 2, ..., n_i, \text{i} = 1, 2, ..., f, k = 1, 2, ..., n$$

$$Y_{i,k} = 0,1 \qquad k = 1, 2, ..., n, i = 1, 2, ..., f$$

$$C_k,\ T_k \ge 0 \qquad k = 1, 2, ..., n$$

The objective of the model is to minimize the total tardiness of the problem. Constraints (1) and (2) state that each position can be occupied by only one job and each job can be processed only once. Constraint (3) controls the setup time of the first position of the sequence, leading to the completion time of the first position in (5). Constraint (4) checks whether or not the preceding job and the following job are from the same family. If so, there is no setup time between them. Otherwise, a family setup time of the job in position *k* exists. Constraint (6) calculates the completion time from the 2nd position to the last position of the sequence. Constraint (7) determines the tardiness values for all positions, while constraint (8) ensures that each family has only one setup. Note that X_{ijk} and $Y_{i,k}$ are binary (0-1) integer variables. In general, there are $n^2 + (f+2)\, n$ variables, with $n^2 + f\, n$ binary integer variables, and $(4+f)\, n + f$ constraints in the proposed model.

The integer programming model was coded and executed under the GAMS/CPLEX environment. The GAMS/CPLEX solver is widely accepted to be one of the fastest solvers available today. However, the computational time for a medium/large size problem was noted to be excessive. Therefore, the solver speed is accelerated by attaching a heuristic solution as an upper bound of a problem. With this upper bounding technique, the solver would overlook some iterations with total tardiness values worse than its upper bound value. However, even with this technique, the computational time was still huge. Table 1 shows the summary of the number of optimal solutions found on small size problems. The values in the "Optimal" column indicate the number of problems that could be solved optimally. The "Feasible" column shows the number of problems for which feasible solutions were found but failed to reach the optimal solutions within a pre-set time limit (10 minutes) on a Pentium II 400Mhz PC. The values in the "Unfound" column indicate that CPLEX did not find a feasible solution within the 10 minute time limit. As shown

in Table 1, the number of "Unfound" problems increased when the number of families and number of jobs increased. This result supports our conclusion that increasing both the number of families and jobs results consumes more computational time.

[Insert table 1 about here]

3. HEURISTIC ALGORTIHMS

The proposed heuristics may be classified into two groups: *sequence construction* and *sequence improving* heuristics. For each group, we need heuristics to sequence the jobs within each family and also sequence the families. The PSK/NBR heuristics (Panwalkar, Smith, and Koulamas, 1993; Holsenback and Russell, 1992) were modified and adapted and combined to find a job sequence inside each family. Given the sequence of jobs in each family, a GT heuristic was developed and implemented to construct an initial sequence of families. The initial sequences were further improved by the SWAP and simulated annealing heuristics. The details of each of these heuristics are presented next.

The PSK/NBR Heuristic

For each family, all jobs may be sequenced using the NBR and PSK heuristics. The NBR heuristic (Holsenback and Russell, 1992) and the PSK heuristic (Panwalkar, Smith, and Koulamas, 1993) described in Appendix A are well recognized to be quite effective in solving the single-family single-machine total tardiness problems. Nevertheless, a preliminary experiment was performed to investigate the performance of these heuristics. The results showed that neither of the two heuristics outperforms the other. The PSK heuristic may perform better than the NBR heuristic in one problem, while it may perform worse than the NBR heuristic in another problem. Therefore, we used both heuristics, and then selected the better (lower) value. Since both heuristics assume that the job assignment starts from time zero, we modified these heuristics to accept an initial starting time C. Table 2 summarizes the experimental results of the composite PSK/NBR heuristic, compared

to the optimal solutions obtained from GAMS/CPLEX solver. This comparison also includes the C time as the additional parameter representing different start time distribution. The sample size for each problem size was 160. Note that the "No. Opt. Solutions" column shows the number of data sets for which the PSK/NBR heuristic could find the optimal solutions.

As can be seen from Table 2, the results indicate that the PSK/NBR heuristic provides near-optimal solutions. The maximum relative deviation was less than 15%, while average relative deviation was less than 3%. Furthermore, the performance of both heuristics improved noticeably, as an initial start time was kept sufficiently large. Thus, it would be reasonable to use this *combined* PSK/NBR heuristic to sequence the jobs inside each family.

[Insert table 2 about here]

The GT Heuristic

As mentioned above, the sole purpose of the proposed GT heuristic is to generate an *initial* sequence for use in a sequence improving heuristic. Let U and S be a set of unscheduled families and a set of scheduled jobs, respectively. In the sequence construction phase, the GT heuristic generates an initial sequence using the following steps.

1. Set $U = \{1, 2, \ldots, f\}$, $i = f$, and $S = \phi$.
2. Considering each family in set U, determine the sequence of all jobs in a batch for the i^{th} family position. Note that the starting time for family k can be determined by $t = \sum_{U} s_j + \sum_{U-k} p$, where p is the total processing times of family in set U. The near-optimal sequence inside a batch is obtained by the PSK/NBR heuristic starting from time t. Calculate the total tardiness at family position i for each family.
3. Select the family with the *lowest* tardiness for family position i, and remove that family from set U. In case of ties, select a family arbitrarily. Put the

family batch in front of existing jobs in *S*. Set $i = i - 1$. If $i = 1$, enter Step 4. Otherwise, return to Step 2.

4. Determine the sequence of jobs of the last family in *U* by the PSK/NBR heuristic with t = setup time of the last family, then put them in front of the existing jobs in set *S*. Calculate the total tardiness of the final sequence in *S*, and then terminate.

SWAP (SW) Heuristic

We now turn to sequence improving heuristics. The proposed SW improvement phase consists of two switching strategies: adjacent pairwise interchange and randomized interchange. The first interchange, *S1*, swaps two consecutive families, say families at family positions i and $i+1$, from the first family position of the sequence to the last family position, and then a new solution is obtained by applying the PSK/NBR heuristic in each family. If the new solution is not better than the current solution, those families are switched back to the original positions, and then the next pair of adjacent families is considered. However, if the solution is improved after switching, the initial sequence is updated and the process starts from the family at family position $i-1$ to examine solution improvement when switching a new family at position i and a family at position $i-1$. The process continues until the families at positions $f-1$ and f are considered.

Randomized interchange is implemented as the second switching strategy (*S2*). This strategy randomly chooses any two families and switches the family positions, and then a new solution is obtained by applying the PSK/NBR heuristic in each family. If the solution is not improved, the families are switched back to their original positions. This randomized process helps prevent the solution getting stuck into a local optimum resulting from swapping only two consecutive families in the first switching strategy. The number of randomized switchings is determined by the complexity of the first switching strategy, which is f^2. After the randomized switching strategy is done, the updated solution is compared with the initial solution before the first switching strategy. If the solutions are not identical, then the updated solution is retained and the improving process is restarted from adjacent pairwise interchange. Otherwise, the search terminates.

Simulated Annealing

The SW heuristic described above may be considered one of the local search approaches. One common technique used in such approaches is that a solution is only updated whenever an improvement exists. Thus, it may get trapped at a local optimum. To overcome this problem, we propose to use simulated annealing (SA) algorithms which are similar to the random descent method in that the neighborhood is sampled at random. The difference is that, in simulated annealing algorithm, an inferior solution may be accepted with some probability. This approach, developed by Kirkpatrick, Gelatt, and Vecchi (1983), has been used widely in solving combinatorial optimization problems.

Simulated annealing allows for *uphill moves* (accepting an inferior solution) with some probability, in an attempt to decrease the chance of becoming stuck in a local optimum. In general, a simulated annealing algorithm starts by first defining the simulated annealing parameter values. These parameters are the initial temperature (T_{max}), and final temperature (T_0), temperature decay rate (r), and iterations (n). The initial temperature is a parameter which acts like an iteration/time counter for the algorithm. The temperature is successively reduced by means of a temperature decay rate, r. When the temperature reaches final temperature (T_0), the procedure is said to be 'frozen' and is terminated. At every temperature iterations are carried out in search for better solutions.

In our proposed algorithm, within any iteration, an insertion method was used. SA would select any two families randomly and interchange the families in those positions. Here, all jobs in each family are moved to a new family position, and the PSK/NBR heuristic is applied to re-sort the jobs inside each family. The purpose of this procedure is to introduce different incumbent family sequences for the insertion process. In the insertion process, a family is selected at random and is inserted at every family position to find the best solution. Once the best family sequence is founded, a new total tardiness value is calculated. Furthermore, simulated annealing randomizes the search procedure to allow for the occasional acceptance of an inferior solution, in an attempt to reduce the probability of the search becoming trapped in a locally optimal solution. SA uses the Boltzmann probability function to determine whether to accept a change that would lead to a worsening of the objective function value. The Boltzmann probability function is defined by its probability mass function:

$$p(c) = \frac{1}{1 + e^{\Delta H / T}}$$

where T = current temperature, $\Delta H = H` - H$, $H`$ = total tardiness value *before* the insertion process, and H = total tardiness value *after* the insertion process.

The steps of the simulated annealing algorithm used here are as follows:

1. Set the control parameters: T_{max}, T_0, r, and n.
2. Set current temperature (T_{cur}) = T_{max} and calculate the total tardiness of current sequence, H.
3. While ($T_{cur} > T_0$)
   - 3.1. Perform the following procedures n times.
     - 3.1.1. Select a random number between 1 to f.
     - 3.1.2. Insert the selected family into every other family positions and choose the family sequence with the lowest total tardiness value, H'.
     - 3.1.3. Set $\Delta H = H' - H$.
     - 3.1.4. If $\Delta H \leq 0$ (downhill move), accept a new sequence. Set $H = H'$.
     - 3.1.5. If $\Delta H > 0$ (uphill move), then
       - 3.1.5.1. Calculate the probability of accepting the new sequence using Boltzmann probability mass function, pc.
       - 3.1.5.2. Select a random number between 0 and 1, say h. if $pc > h$, accept the new sequence and set $H = H'$. Otherwise, reject the new sequence and use the previous sequence for the next iteration.
     - 3.1.6. Return to 3.1.1.
   - 3.2. Set $T_{cur} = r \, . T_{cur}$.
   - 3.3. Return to 3.
4. Terminate.

The above simulated annealing algorithm terminates when $T_{cur} \leq T_0$. We propose to use an additional simulated annealing algorithm to improve the solution. Before simulated annealing terminates, the current solution is compared with the initial solution from Step 2. If a difference exists, then we go back to Step 2, update the initial solution in Step 2 with the current solution, and restart the whole algorithm again. The algorithm keeps iterating until the there is no improvement in solution. In this paper, the first and second simulated annealing algorithms are called the SA-NL

(SA without return) and SA-L heuristics (SA with return), respectively. Further, the parameter values used in this study for SA heuristics were: T_{max}= 1000; T_0 = 0.1; r = 0.90, and $n = f$ x f.

4. COMPUTATIONAL RESULTS

We now report on the evaluation of the effectiveness of the proposed heuristic algorithms in solving the single-machine group scheduling problem to minimize total tardiness.

Experiment Settings

The parameter settings, such as number of jobs, number of families, the due date range parameter and setup type, were determined following the existing literature and experimental justification. The processing times in all experiments are randomly assigned from a uniform distribution of integers from 1 to 100. Additionally, in practice, when a due date is assigned to a job, the possibly earliest due date of that job should be at least equal to the required processing time plus its family setup time. As a result, due dates are drawn from the integral range of $[s_q + p_{qj}, max(s_q + p_{qj}, rP)]$, where $P = \sum_{i=1}^{f} S_i + \sum_{i=1}^{f} \sum_{j=1}^{n_i} p_{ij}$.

We note that r in the above expression is the due date tightness parameter. The smaller the value of r, the higher the number of tardy jobs is due to the due date range. Four values starting from 0.25 to 1.00 with increment of 0.25 were used in this study. Furthermore, a family designation of a job was assigned from a uniform distribution with a range of [1, f]. Thus, the total number of jobs in each family would not be equal. Moreover, to detect the effect of setup time on the performance of an algorithm, three different types of setup time ranges were assigned with discrete uniform distribution as follows: type 1: [1, 20]; type 2: [1,100], and type 3: [101, 150]. Setup time distributions in types 1 and 3 generated small setup time and large setup time values, respectively, while setup time type 2 had the same distribution as the processing times.

All heuristics and simulated annealing algorithms were coded in C++ and were executed on a Pentium II 400Mhz PC. The experiments are categorized into two main sections. In the first section, small problem sizes are considered. Fifteen problem sizes considered in this part are 10 x 2, 10 x 4, 10 x 6, 10 x 8, 10 x 10, 15 x 2,

15 x 4, 15 x 6, 15 x 8, 15 x 10, 20 x 2, 20 x 4, 20 x 6, 20 x 8, and 20 x 10. Note that a problem size of 10 x 2 represents a problem with a total of 10 jobs and 2 families, and so on. The heuristic solutions are compared with the optimal solutions obtained from GAMS/CPLEX. The second section is involved with the large problem sizes. Twenty five problem sizes represent the combinations of 5 job sizes (20, 40, 60, 80, and 100) and 5 family sizes (2, 4, 6, 8, and 10). Each problem size consists of 12 combinations of 3 setup types and 4 due date range parameters. Sample size for each combination is 5. Therefore, the total number of data sets used for experiments of small problem sizes and large problem sizes are 900 and 1,500 data sets, respectively.

Results for Small Size Problems

In this section, small problem size experiments were performed. The solutions from heuristics were compared with the optimal solutions from GAMS/CPLEX solvers. Note that the GAMS/CPLEX solver was run for 10 minutes for each data set. Therefore, a solution from the solver could be infeasible solution, feasible solution, or optimal solution. *Infeasible solution* indicates that the solver could not find a feasible solution within time limit (10 minutes). However, if the solver can find a feasible solution within the time limit, but fails to find an optimal solution, then the solution is a feasible solution. An optimal solution is a solution when the solver optimally solved the problem within the time limit. In the experiment, we only consider the data sets with feasible and optimal solutions. Let H_H be the total tardiness of schedule found by using heuristic H and $H_{Optimal}$ be the optimal value of total tardiness found by using MILP model described earlier. Then, the performance heuristic H, measured by the relative deviation percentage, is defined as:

$$deviation(\%) = \frac{H_H - H_{Optimal}}{H_{Optimal}} x100$$

To compare the effectiveness of job scheduling and family scheduling heuristics, we created another heuristic, called "Perm.". This heuristic enumerates all possible family sequences, and then uses the PSK/NBR heuristic to obtain job sequences inside each family. In the following discussion, a schedule obtained by this heuristic is called a "permutation schedule".

Table 3 summarizes the results of Permutation (Perm.), GT, SW, SA-NL, and SA-L heuristics, compared to the optimal solutions from GAMS/CPLEX. The results show that the performances of SA algorithms are slightly better than the SW heuristics. The negative values in Table 3 indicate that the heuristics provide better

solutions than the GAMS/CPLEX solver does because the solver sometime provides only feasible solutions. These experiments indicated that the permutation approach obtains near-optimal/optimal solutions in several cases. The difference in results between optimal solutions and permutation solutions is caused by the job sequence inside each family. As stated earlier, the PSK/NBR heuristic demonstrated good performance in providing near-optimal solutions for job sequence. Therefore, this approach can be used to provide a benchmark for comparison purpose in our large problem size experiments. We also note that this approach is not appropriate for problems with more than 10 families, as the computational time required to enumerate all family sequences would increase dramatically.

[Insert table 3 about here]

During the experiments, we noticed that, in most cases, the heuristics could obtain the optimal family sequences. Thus, the difference in solution could be caused by the job sequence inside each family. Therefore, to improve the solutions, after applying the heuristics, we would re-sort job sequences inside each family with another simulated annealing approach. We tested the performance of the SA, by comparing to the optimal solutions obtained from Szwarc, Grosso, and Della Croce (2001). This SA procedure for job sequence is similar to the SA-NL algorithm we proposed above, except that we consider jobs, instead of families. The parameter values used in this SA algorithm were as follows: T_{max}= 10000; T_0 = 0.1; r = 0.995, and n = 100. The results in table 4 show that the SA algorithm performs slightly better than the PSK/NBR heuristic. Therefore, it would be advisable to use this SA algorithm to improve the job sequences inside each family.

[Insert table 4 about here]

According to Table 5, the results of heuristics with and without applying SA for job sequences are not different at all. One possible reason is that the PSK/NBR heuristic already provides optimal solutions, thus SA for job sequence would not show much improvement in solutions. This conclusion is supported by the results from Table 2. For problems with 20 jobs, out of 160 problems, 117 problems were

solved for optimal solutions by the PSK/NBR heuristic. Nevertheless, we hypothesize that the results may be different when the problem size increases.

[Insert table 5 about here]

Large Problem Size Experiments

In the case of large problem size, we compared the results of heuristics with the permutation solutions. Let H_H and H_{perm} be the total tardiness of schedule found by using heuristic H and H_{perm} respectively. Then, the performance of heuristic H, measured by the relative deviation percentage, is defined as:

$$deviation(\%) = \frac{H_H - H_{perm}}{H_{perm}} x100$$

According to tables 6-9, applying SA to job sequences after applying the SW heuristic and both simulated annealing algorithms (SW-SA, SA-NL-SA and SA-L-SA) hardly show improvement in solutions at all. Thus, we will only consider the results from the SW, SA-NL and SA-L heuristics. Table 6 shows that the GT heuristic provides good solutions with average relative deviations less than 10%. The SW and simulated annealing heuristics perform quite well, as the average deviations are less than 1%. When the number of jobs increases, the performance of the SW heuristic imporves, while the performance of the simulated annealing algorithms (SA-NL and SA-L) decreases. However, Table 7 shows different results. Grouped by the number of families, the SW heuristic appears to perform worse when the number of families increases, while the other two SA heuristics seem to perform consistently.

[Insert table 6 about here]

[Insert table 7 about here]

[Insert table 8 about here]

[Insert table 9 about here]

The results in Table 8 show that due date tightness parameters appear to have no effect on the GT heuristic's performance, while the remaining heuristics seem to worsen as the due date parameter value increases. In addition, the SW heuristic appears to outperform both SA heuristics when the due date tightness value is small. However, when the due date tightness value is large, the SA-L heuristic performs better than the SW heuristic. Furthermore, Table 9 shows that the SW heuristic outperforms the SA-L heuristic when the family setup times are large.

Next, a comparison among the SW, SA-NL, SA-L heuristics is presented. Table 10 summarizes the basic statistical results including the 95% confidence intervals for the differences between any two heuristics. The confidence intervals show that the SA-NL heuristic is worsen than the SA-L heuristic, while the SW and SA-L heuristics cannot completely outperform each other. Therefore, the SW heuristic is recommended to solve problems with a few families, tighter due dates, or large family setup times. In other cases, the SA-L heuristic would seem to be a better approach. Nevertheless, all heuristics, except the GT heuristic, provide acceptable performance with average relative deviation less than 1%.

[Insert table 10 about here]

5. CONCLUSIONS

The purpose of this research was to develop and evaluate methodologies for solving the single machine group scheduling problem to minimize total tardiness. Since the complexity of this problem is at least binary NP-Hard, this paper developed and implemented effective heuristics that would provide good solutions within a reasonable time. Computational results show that the proposed heuristics can provide good solutions with the average relative deviation less than 1 %. It is shown that the combined PSK/NBR heuristic delivers good performance in obtaining the near-optimal job sequence inside a family. Furthermore, two promising heuristics, the SWAP and SA-L heuristics, may be used in a variety of situations. The SW heuristic is recommended for problems with a few families, tighter due dates, or large family setup times. In other cases, the SA-L heuristic seems to be a better approach. Our suggestion is to use both heuristics, and select the lower solution value.

Several issues are worthy of future investigation. First, recent literature provides fast and effective optimization algorithms for traditional single machine total tardiness problems. Replacing the PSK/NBR heuristic with these algorithms would seem to be a good strategy in improving the solutions. Second, designing and evaluating other meta-heuristics, such as tabu search and genetic algorithm, for improving the proposed solution could be beneficial. Finally, extension of our proposed approaches to more complex machine environments and other optimality criteria, like the total weighted tardiness, would be worthwhile future research projects.

APPENDIX A

The PSK Heuristic (Panwalkar, Smith, and Koulamas, 1993)

The PSK heuristic makes n passes from left to right and in the k^{th} pass, a job is selected and is put in the k^{th} position. The procedure of PSK heuristic starts with sorting all jobs with the SPT rule (ties are broken in an EDD order) and putting all jobs in set U. Set $C = 0$ and $S = \phi$, where C is sum of processing time of jobs in S, and S is set of scheduled jobs. Next the procedure of the PSK heuristic is presented as follows:

1. If U contains only 1 job, schedule it in the last position in S and go to Step 9. Otherwise, label the leftmost job in U as the active job J_i.
2. If $C + p_i \geq d_i$, then go to Step 8.
3. Select the next job on the right in U as job J_j.
4. If $d_i \leq C + p_j$, then go to Step 8.
5. If $d_i \leq d_j$, go to Step 7.
6. Job J_j now becomes the active job J_i. If this is the last job in U, go to Step 8; ELSE return to Step 2.
7. If J_j is the last job in U, go to Step 8; ELSE return to Step 3.
8. Remove job J_i from U and put it in the last position in S, then update the value of C by p_i, and return to Step 1.
9. Calculate total tardiness for the sequence and terminate.

The NBR Heuristic (Holsenback, and Russell, 1992)

The NBR heuristic is implemented based on the concept of Net Benefit of Reallocation (NBR), which is defined as:

$$NBR_j = BR_j - CR_j$$

where $CR_j = \max\left[0, \sum_{i=j+1}^{k} p_i - S_j\right]$, $BR_j = \sum_{i=j+1}^{k} \min(p_j, T_j)$, and $S_j = \max(d_j - C_j, 0)$

Note that CR_j is the Cost of Reallocating of a job from position j to position k. Furthermore, once the job at position j is moved, the jobs after position j are moved forward, resulting in less tardiness values of all jobs between position j and k. It is called Benefit of gained from Reallocating (BR). Furthermore, the initial sequence for NBR is implemented using modified due date, which is $d_k$mod = max (d_k, p_k). The NBR procedures are presented as follows:

Step 1

Label all jobs. This is necessary because some due dates may have to be modified.

Adjust due dates as necessary according to the modification rule.

Order the set by non-decreasing due dates and in the case of equality, by non-decreasing of processing times. Renumber the jobs so that this sequence is (J_1,..., J_N)

Calculate the slack or tardiness of each job in the sequence.

Step 2

Beginning with the job in the last position, J_N, and incrementally proceeding toward J_1, identify the first job J_k with $T_k > p_k$. If none exists, proceed to Step 7.

Step 3

Beginning with J_{k-1} and continuing forward J_1, identify the first preceding job, J_j, with property $p_j > p_k$. If none exists, proceed to Step 6.

Beginning with J_{j-1} and continuing forward J_1, identify the first preceding job, J_{j-m}, with property $p_{j-m} > p_k$. Continue in this manner until J_1 has been considered.

Step 4

Compute the NBR_j for each job, J_i, identified in Step 3, considering that it will be relocated to a position immediately following J_k.

Relocate that job, Ji, yielding the greatest NBR_i, subject to the constraint $NBR_i > 0$. If no $NBR_i > 0$, proceed to Step 6. If two or more jobs yield the same NBR_i, relocating that job with the greatest processing time. This selection method is purely arbitrary and is chosen to reduce the number of computations necessary for a final ordering.

Step 5

Renumber the current sequences as (J_1, ..., J_N) and recompute the tardiness and slack.

Step 6

Begin with the job in position k –1 and repeat Steps 2-4, substituting k-1 for k.

Continue in this fashion until the job in position 1 has been considered.

Step 7

Final ordering is complete.

Applying the original due dates if any were modified and compute the tardiness of the schedule.

REFERENCES

1. Cheng, T. C. E., Gordon, V. S. and Kovalyov, M. Y. (1996) "Single Machine Scheduling with Batch Deliveries," *European Journal of Operational Research*, 94, 277-283.

2. Du, J. and Leung, J. Y. T. (1990) "Minimizing Total Tardiness on One Machine is NP-Hard," *Mathematics of Operations Research*, 15(3), 483-495.

3. Holsenback, J. E. and Russell, R. M. (1992) "A Heuristic Algorithm for Sequencing on One Machine to Minimize Total Tardiness," *Journal of the Operational Research Society*, 43(1), 53-62.

4. Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983) "Optimization by Simulated Annealing", *Science*, 20/4598, 671-680

5. Monma, C. L. and Potts, C. N., On the complexity of scheduling with batch setups. *Operations Research*, 1989, 37, 798-804.

6. Panwalkar, S. S., Smith, M. L. and Koulamas, C. P. (1993) "A Heuristic for the Single Machine Tardiness Problem," *European Journal of Operational Research*, 70(3), 304-310.

7. Potts, C. N., and Kovalyove, M. Y., Scheduling with batching: a review. *European Journal of Operational Research*, 120(3), 228-249.

8. Potts, C. N., and van Wassenhove, L. N., Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of Operational Research Society*, 1992, 43, 395-406.

9. Szwarc, W., Grosso, A. and Della Croce, F. (2001) "Algorithmic paradoxes of the single-machine total tardiness problem," *Journal of Scheduling*, 4(2), 93-104.

10. Webster, S. and Baker, K. R. (1995) "Scheduling Groups of Jobs on a Single Machine," *Operations Research*, 43(4), 692-703.

Table 1. The number of optimal/feasible/unfound solutions from GAMS/CPLEX: sequence independent setup case.

Problem Size	Total	Optimal	Feasible	Unfound
10 x 2	60	60	-	-
10 x 4	60	55	5	-
10 x 6	60	59	1	-
10 x 8	60	60	-	-
10 x 10	60	58	2	-
15 x 2	60	42	7	11
15 x 4	60	46	4	10
15 x 6	60	41	15	4
15 x 8	60	38	17	5
15 x 10	60	28	24	8
20 x 2	60	31	12	17
20 x 4	60	20	13	17
20 x 6	60	9	38	13
20 x 8	60	1	41	18
20 x 10	60	0	38	22

Table 2. Statistical summary for the PSK/NBR heuristic

# Jobs	Initial C time dist.	No. Opt. Solutions	Mean	Standard Deviation	Max
10	0	40	0.055	0.239	1.303
	[1,25]	40	0.161	0.860	5.363
	[26,50]	40	0.000	0.000	0.000
	[51,100]	40	0.000	0.000	0.000
20	0	30	0.197	0.442	1.738
	[1,25]	29	0.148	0.488	2.481
	[26,50]	29	0.071	0.179	0.664
	[51,100]	29	0.025	0.107	0.559
40	0	20	1.221	0.817	3.397
	[1,25]	21	2.194	2.449	9.454
	[26,50]	21	1.858	1.549	5.944
	[51,100]	21	2.056	2.363	11.008
60	0	20	2.227	2.096	7.794
	[1,25]	19	1.952	2.024	7.828
	[26,50]	19	2.059	1.567	6.070
	[51,100]	19	1.589	1.050	3.941
80	0	13	1.895	1.871	6.276
	[1,25]	13	1.509	0.950	3.470
	[26,50]	13	1.387	1.156	3.244
	[51,100]	13	1.734	1.657	4.837
100	0	10	1.994	1.486	4.386
	[1,25]	11	2.016	1.488	5.441
	[26,50]	10	1.655	1.181	3.490
	[51,100]	10	1.920	0.963	2.911

Table 3. Average relative deviation of proposed heuristics for small size problems

(grouped by number of jobs and number of families)

# Jobs	# Fam.	Perm.	GT	SW	SA-NL	SA-L
10	2	0.000	0.814	0.000	0.000	0.000
	4	0.000	9.021	0.216	0.254	0.061
	6	-0.618	12.956	0.330	-0.399	-0.493
	8	1.480	15.798	1.782	1.983	1.905
	10	-0.329	18.542	0.110	-0.120	-0.137
15	2	0.000	1.391	0.000	0.000	0.000
	4	-1.248	5.143	-1.038	-0.764	-0.767
	6	-0.695	9.617	-0.308	-0.587	-0.556
	8	-3.578	10.736	-2.938	-3.261	-3.198
	10	-6.469	10.697	-6.143	-6.411	-6.398
20	2	-1.163	0.230	-1.163	-1.163	-1.163
	4	-5.780	0.805	-5.291	-5.643	-5.635
	6	-13.998	-5.263	-13.845	-13.947	-13.953
	8	-18.459	-8.074	-17.942	-18.233	-18.224
	10	-24.234	-12.244	-23.892	-23.908	-23.967

Table 4. SA performance compared to the optimal solutions

		Relative Deviation (%)			
# Jobs	No. Opt. Solutions	Mean	Std. Dev.	Maximum	95% Confidence Interval
10	75	0.057	0.287	2.267	[0.000, 0.121]
20	53	0.258	0.541	2.904	[0.138, 0.379]
40	38	0.328	0.601	2.885	[0.195, 0.462]
60	16	0.681	1.064	6.532	[0.444, 0.917]
80	7	0.665	0.700	2.602	[0.509, 0.821]
100	4	0.608	0.766	3.633	[0.437, 0.778]

Table 5. Performance comparison of SA for job sequences
(Average relative deviation)

# Jobs	Fam.	SW	SW-SA	SA-NL	SA-NL-SA	SA-L	SA-L-SA
10	2	0.000	0.000	0.000	0.000	0.000	0.000
	4	0.216	0.209	0.254	0.254	0.061	0.061
	6	0.330	0.231	-0.399	-0.399	-0.493	-0.493
	8	1.782	1.594	1.983	1.983	1.905	1.905
	10	0.110	-0.324	-0.120	-0.120	-0.137	-0.137
15	2	0.000	0.000	0.000	0.000	0.000	0.000
	4	-1.038	-1.038	-0.764	-0.764	-0.767	-0.767
	6	-0.308	-0.391	-0.587	-0.587	-0.556	-0.556
	8	-2.938	-3.257	-3.261	-3.261	-3.198	-3.198
	10	-6.143	-6.496	-6.411	-6.411	-6.398	-6.398
20	2	-1.163	-1.163	-1.163	-1.163	-1.163	-1.163
	4	-5.291	-5.291	-5.643	-5.643	-5.635	-5.636
	6	-13.845	-13.930	-13.947	-13.947	-13.953	-13.953
	8	-17.942	-18.177	-18.233	-18.233	-18.224	-18.224
	10	-23.892	-24.173	-23.908	-23.908	-23.967	-23.967

Table 6. Average relative deviation of proposed heuristics
grouped by number of jobs (large problem size)

# Jobs	GT	SW	SW-SA	SA-NL	SA-NL-SA	SA-L	SA-L-SA
20	9.494	0.373	0.216	0.221	0.221	0.211	0.210
40	7.434	0.389	0.240	0.238	0.238	0.152	0.151
60	6.754	0.356	0.240	0.327	0.326	0.232	0.231
80	6.122	0.252	0.145	0.531	0.529	0.383	0.380
100	6.154	0.187	0.100	0.569	0.568	0.418	0.416

Table 7. Average relative deviation of proposed heuristics grouped by number of families (large problem size)

# Families	GT	SW	SW-SA	SA-NL	SA-NL-SA	SA-L	SA-L-SA
2	0.784	0.000	0.000	0.000	-0.005	0.000	-0.005
4	5.960	0.284	0.269	0.386	0.386	0.352	0.351
6	8.083	0.286	0.165	0.466	0.466	0.354	0.354
8	9.735	0.564	0.341	0.512	0.512	0.391	0.391
10	11.397	0.423	0.166	0.523	0.523	0.297	0.297

Table 8. Average relative deviation of proposed heuristics grouped by due date tightness values (large problem size)

R	GT	SW	SW-SA	SA-NL	SA-NL-SA	SA-L	SA-L-SA
0.25	6.242	0.099	0.034	0.230	0.228	0.144	0.142
0.50	7.830	0.259	0.130	0.347	0.345	0.256	0.254
0.75	7.626	0.398	0.267	0.375	0.374	0.328	0.328
1.00	7.069	0.490	0.322	0.557	0.557	0.388	0.388

Table 9. Average relative deviation of proposed heuristics grouped by family setup time type (large problem size)

Setup	GT	SW	SW-SA	SA-NL	SA-NL-SA	SA-L	SA-L-SA
[10,20]	9.770	0.306	0.165	0.353	0.352	0.273	0.272
[1, 100]	7.606	0.394	0.248	0.380	0.379	0.276	0.275
[101, 150]	4.198	0.234	0.152	0.399	0.397	0.288	0.286

Table 10. Statistical summary for the heuristic comparisons

Heuristic Comparison	$\bar{d}$	Std. Dev.	d_{max}	95% Confidence Interval for d
d = (SW) – (SA-L)	0.063	1.650	18.303	[-0.007, 0.134]
d = (SW) – (SA-NL)	-0.014	1.693	18.313	[-0.087, 0.058]
d = (SA-NL) – (SA-L)	0.077	0.961	11.551	[0.036, 0.119]