International Journal of Operations Research

International Journal of Operations Research Vol. 3, No. 3, 1–11 (2006)

A Tabu Search Algorithm to Minimize the Makespan for the Unrelated Parallel Machines Scheduling Problem with Setup Times

Magdy Helal[1], Ghaith Rabadi[2,*], and Ameer Al-Salem[3]

[1]Industrial Engineering & Management System Department, University of Central Florida, Orlando, Florida 32816, USA

[2]Department of Engineering Management and Systems Engineering, Old Dominion University, 241 Kaufman Hall, Norfolk, Virginia 23529, USA

[3]Department of Mechanical Engineering, University of Qatar, Doha, Qatar, PO Box: 2713

Received June 2005; *Revised October 2005*; *Accepted August 2006*

Abstract—In this paper we propose a tabu search implementation to solve the unrelated parallel machines scheduling problem with sequence- and machine- dependent setup times to minimize the schedule's makespan. The problem is NP-hard and finding an optimal solution efficiently is unlikely. Therefore, heuristic techniques are more appropriate to find near-optimal solutions. The proposed tabu search algorithm uses two phases of perturbation schemes: the intra-machine perturbation, which optimizes the sequence of jobs on the machines, and the inter-machine perturbation, which balances the assignment of the jobs to the machines. We compare the proposed algorithm to an existing one that addressed the same problem. The computational results show that the proposed tabu search procedure generally outperforms the existing heuristic for small- and large-sized problems.

Keywords—Tabu search, Scheduling, Unrelated parallel machines, Setup times

1. INTRODUCTION

The problem addressed in this paper is the non-preemptive scheduling of N available jobs on M parallel, unrelated machines where the jobs' processing times are machine dependent and there is no relationship between machine speeds. Free machines are capable of processing any job without preemption where it may take for example processing time of p_{j1} to process job j on machine 1, and p_{j2} to process job j on machine 2 with $p_{j1} > p_{j2}$; while it may take p_{i1} to process job i on machine 1, and p_{i2} to process job i on machine 2 with $p_{i1} < p_{i2}$. In other words, a speed factor among machines is not identified. The objective is to minimize the makespan or the maximum completion time C_{max}. Sequence-dependent setup times S_{ijk} are considered. They are separated from the processing times where S_{ijk} is the amount of setup time needed if job j is scheduled after job i on machine k. Sequence-dependent setup times add tremendous amount of computational complexity as they significantly increase the number of possible permutations.

This problem is NP-hard as it is a generalization of the identical parallel machine scheduling problem (PMSP), which is NP-hard even for two machines and makespan as scheduling criterion (Garey and Johnson, 1979). In the case of identical machines, the processing time of a job is the same regardless of which machine processes it. For the unrelated PMSP case machines do not necessarily have the same capacities and/or capabilities, and the same job may have different processing and setup times on the different machines.

Having resources in parallel is common in real life situations to obtain adequate capacity. Setup activities are often required when switching between jobs. Applications are common in painting and plastic industries where thorough cleaning is required between operations. Similar situations are also common in textile, glass, chemical, and paper manufacturing industries, as well as some service industries (Franca et al., 1996; Radhakrishnan and Ventura, 2000; Kurz and Askin, 2001; Randhawa and Kuo, 2001).

Research efforts to solve the PMSPs have dealt with many variations of the problem; identical and unrelated machines with and without setup times. Other characteristics that were studied included preemptive vs. non-preemptive problems, distinct due vs. common due date for, and the static vs. dynamic cases (Franca et al., 1996; Randhawa and Kuo, 1997; Azizoglu and Kirca, 1999; Sivrikaya-Serifoglu and Ulusoy, 1999; Radhakrishnan and Ventura, 2000; Kurz and Askin, 2001; Yalaoui and Chu, 2002). Yet the majority of the published work has concentrated on the case of the identical parallel machines. Further, most of the reviewed literature used the minimization of either makespan or maximum tardiness as scheduling objective. Comprehensive reviews and state-of-the-art surveys can be found in Graves (1981), Cheng and Sin (1990), Lawler et al. (1993), and more recently in Mokotoff (2001). Additionally, Allahverdi et al.

* Corresponding author's email: grabadi@odu.edu

(1999) presented a survey on scheduling problems involving setup times.

Attempts to generate optimum solution for the PMSP were conducted by Liaw et al. (2003) and Lancia (2000) who developed branch-and-bound algorithms to find optimal solutions without considering setup times. The objective functions were the total weighted tardiness and makespan respectively. In Section 2 of this paper we present a Mixed Integer Programming (MIP) formulation to find optimal solutions for the problem at hand.

Being NP-hard, various heuristic procedures were proposed in literature for the different variations of the PMSP. Horn (1973) and Bruno et al. (1974) examined the problem for minimizing the total completion time heuristically. Hariri and Potts (1991) introduced a two-phase heuristic approach in which they applied linear programming to generate a partial schedule and then used the earliest completion time heuristic to schedule the remaining jobs to minimize the makespan. They did not, however, consider setup times. Azizoglu and Kirca (1999) studied the problem considering the total weighted completion time. Bank and Werner (2001) compared several constructive and generative algorithms considering situations with release dates to minimize the weighted sum of earliness and tardiness. Weng et al. (2001) considered sequence-dependent setup times where they presented and tested several heuristics for problems of up to 120 jobs and 12 machines. Their objective was to minimize the weighted mean completion time and they only considered machine-independent setup times.

Meta-heuristic techniques have also been applied to the unrelated PMSP. Glass et al. (1994) compared genetic algorithms (GA), simulated annealing (SA), and tabu search (TS) to minimize makespan without setup times. They noticeably reported poor performance of GA. Srivastava (1997) presented a TS formulation for the same problem addressed in this paper without setup times and reported that TS can provide good quality solutions for practical size problems within reasonable computational times. Kim et al. (2002, 2003) developed SA implementations to minimize the total tardiness, considering only sequence-dependent setup times. Ghirardi and Potts (2005) applied a method called the Recovering Beam Search to minimize the makespan on unrelated parallel machines without setups.

In the current paper we propose a TS based heuristic algorithm for solving the PMSP with sequence- and machine- dependent setup times. Our review identified no TS implementations for this problem before ours. We compare the proposed TS heuristic with the Partitioning Heuristic (PH) algorithm proposed by Al-Salem (2004) with respect to makespan for different problem sizes. We also compare the two techniques to the optimal solution for small problem sizes as obtained by an optimal Integer Program. Throughout this paper, small problems refer to instances of up to 9 jobs and 4 machines, while large problems refer to instances of up to 120 jobs and 12 machines.

Up to our knowledge, only Al-Salem (2004) has addressed the same problem we are addressing in this paper where he introduced the PH, which performed well compared to a lower bound that he proposed in the same paper. The TS implementation in the current paper was first proposed by Helal and Hosni (2003). They showed that the algorithm is effective in improving a random initial solution and reaching a balanced assignment of jobs to machines. Improvements of average of 60% in initial makespan were achieved for problems of up to five machines and 40 jobs. However they did not evaluate the performance of the algorithm with respect to any other technique. There are still no other TS formulations that addressed the current problem.

2. OPTIMAL SOLUTION FORMULATION

A Mixed Integer Program (MIP) is formulated to find optimal solutions for the problem at hand. This formulation is used to evaluate the TS performance for small-sized problems. Similar formulation was introduced by Guinet (1991) who addressed the same problem but with the total completion time as an objective in one section and the total tardiness as another objective in another section of the paper. Our objective in this MIP is to minimize the makespan.

$$\text{Minimize } C_{max} \tag{1}$$

subject to

$$\sum_{\substack{i=0\\ i\neq j}}^{N}\sum_{k=1}^{M} x_{i,j,k} = 1, \forall j = 1,...,N \tag{2}$$

$$\sum_{\substack{i=0\\ i\neq h}}^{N} x_{i,h,k} - \sum_{\substack{j=0\\ j\neq h}}^{N} x_{h,j,k} = 0, \forall h = 1,...,N, \forall k = 1,...,M \tag{3}$$

$$C_j \geq C_i + \sum_{k=1}^{M} x_{i,j,k}\left(S_{i,j,k} + p_{j,k}\right) + V\left(\sum_{k=1}^{M} x_{i,j,k} - 1\right)$$
$$\forall i = 0,...,N, \forall j = 1,...,M \tag{4}$$

$$\sum_{j=0}^{N} x_{0,j,k} = 1, \forall k = 1,...,M \tag{5}$$

$$C_j \leq C_{max}, \forall j = 1,...,N \tag{6}$$

$$x_{i,j,k} \in \{0,1\}, \forall i = 0,...,N, \forall j = 0,...,n, \forall k = 1,...,M \tag{7}$$

$$C_0 = 0 \tag{8}$$

$$C_j \geq 0, \forall j = 1,...,N \tag{9}$$

where

- C_{max}: Maximum completion time (makespan)
- C_j: Completion time of job j
- $p_{j,k}$: Processing time of job j on machine k
- $S_{i,j,k}$: Sequence-dependent setup time to process job j after job i on machine k
- $S_{0,j,k}$: Setup time to process job j first on machine k
- $x_{i,j,k}$: 1 if job j is processed directly after job i on machine k and 0 otherwise
- $x_{0,j,k}$: 1 if job j is the first job to be processed on machine k and 0 otherwise
- $x_{j,0,k}$: 1 if job j is the last job to be processed on

machine k and 0 otherwise

V: A large positive number

M: Number of machines

N: Number of Jobs

Objective (1) is to minimize the makespan. Constraints (2) ensure that each job is scheduled only once and processed by one machine. Constraints (3) make sure that each job must neither be preceded nor succeeded by more than one job. Constraints (4) are used to calculate completion times and to ensure that no job can precede and succeed the same job. This is guaranteed by using the large positive number (in theory $V = \infty$) where if job j is scheduled right after job i, then $\sum_{k=1}^{M} x_{i,j,k} = 1$, and so $V\left(\sum_{k=1}^{M} x_{i,j,k} - 1\right) = 0$. This way, $C_j = C_i + S_{i,j,k} + p_{j,k}$. On the other hand, if job j is *not* scheduled right after job i, then $\sum_{k=1}^{M} x_{i,j,k} = 0$ and so $V\left(\sum_{k=1}^{M} x_{i,j,k} - 1\right) = -V$, and therefore, this constraint becomes redundant and does not impact the MIP. Constraints (4) are needed because we do not know which job will precede the other when including sequence-dependent setup times. Constraints (5) ensure that no more than one job can be scheduled first at each machine. In fact, constraints (5) is missing from the original formulation proposed by Guinet (1991) and if not included, infeasible schedules may result because without these constraints more than one job may end up scheduled first (or last) on the same machine, which can be easily demonstrated with a numerical example. Note that there is no need for another set of constraints to guarantee that only one job is scheduled last on each machine because this is guaranteed by constraints (5) in conjunction with (3). Constraints (6) define the makespan C_{max} as a variable that must be larger than any other job's completion time. Constraints (7) specify that the decision variable x is binary over all domains. Constraints (8) state that the completion time for the dummy job 0 is zero and constraints (9) ensure that completion times are non-negative. Optimal solutions for the problem can be obtained by solving the above formulation using a MIP solver.

3. THE PARTITIONING HEURISTIC ALGORITHM

Al-Salem (2004) developed the Partitioning Heuristic (PH) algorithm to minimize makespan on unrelated parallel machines with machine-dependent and sequence-dependent setup times. The PH algorithm applies three heuristics: constructive, improvement, and a traveling salesman problem (TSP)-like heuristic sequentially to minimize the makespan. The first heuristic is applied to initially assign jobs to machines. The second is to improve the solution obtained by the constructive heuristic. The third deals with each machine as a TSP and determines the sequence of jobs on each machine.

The constructive heuristic of the PH algorithm develops an initial assignment of jobs to machines. The processing time plus the average setup times for each job on each machine is computed. Then for each job, the ratio of the minimum machine processing time plus the average machine setup times to the second shortest minimum machine processing time plus the average machine setup times is computed. The job is assigned to the machine with shortest processing time plus average setup times if the value of its ratio is small (e.g., < 0.7). The job is considered pending if the value of its ratio is large (e.g., > 0.7). Pending jobs are then assigned in a decreasing order of the average processing times plus average setup times to machines that result in the lowest partial estimated makespan. The value of the estimated makespan $\hat{C}_{max}$ is obtained similar to the method used by Lee et al. (1997) to estimate the value of the makespan for the single machine scheduling problem with sequence-dependent setup times. Therefore $\hat{C}_{max} = (\theta \bar{s} + \bar{p})n$ where θ is a coefficient that takes into account the effect of the setup times on the makespan. The average time to process a job including its setup time is assumed to be $\hat{C}_{max} = \theta \bar{s} + \bar{p}$ with $\theta \leq 1$. The constructive heuristic of the PH algorithm can be described in pseudo code as follows:

For j = 1 to n, Do:

Calculate $PS_{jk} = P_{jk} + \dfrac{\sum_{i=1}^{n} S_{ijk}}{n}$, k = 1, ..., m

End Do

For j = 1 to n, Do:

Calculate $\rho_{jk^*} = \dfrac{First \min(PS_{jk})}{Second \min(PS_{jk})}$, k = 1, ..., m

If $\rho_{jk^*} \leq \alpha$ assign job j to machine k^*.

Calculate the estimated $\hat{C}_{k^*} = (\theta \bar{s} + \bar{p})n$

Else, job j is considered pending.

End Do

Order the pending jobs in a decreasing value order of

$$\frac{1}{m}\sum_{k=1}^{m}\left(P_{jk} + \frac{\sum_{i=1}^{n} S_{ijk}}{n}\right)$$

Following the pending jobs order, Do:

Find $\min_{k}(\bar{C}_K)$. Let the corresponding k be k^{**}

Assign job j to machine k^{**} and Update $\hat{C}_{k^{**}}$

End Do

To improve the estimated value of the makespan $\hat{C}_{max}$, the improvement heuristic is applied to the schedule generated by the constructive heuristic. In this application, the composite exchange heuristic developed by Hariri and Potts (1991) is modified to account for the sequence-dependent setup times. The modified composite exchange heuristic consists of two phases. In Phase 1, a job is

removed from the machine that generates the largest $\hat{C}_{max}$ and is inserted into a machine that generates the lowest $\hat{C}_{max}$. In Phase 2, two jobs are exchanged, one from the machine that produces the largest $\hat{C}_{max}$ and one from the machine that produces the lowest $\hat{C}_{max}$. All jobs and all possible assignment are considered. Phase 1 is applied by first using the constructive schedule as an input. Then using the resulting improved schedule as an input to Phase 2 and further reduction in the $\hat{C}_{max}$ is attempted. The procedure continues by repeatedly applying Phases 1 and 2 until no further reduction in makespan is possible. At this point, the sequence of jobs on each machine and the actual value of the makespan are unknown. To determine the sequence and the value of the makespan on each machine, the following procedure is executed:

1. Use the nearest neighbor heuristic (NNH) to find the initial sequence and the value of the makespan on each machine.
2. Among all available machines, select the one that produces the largest makespan and the one that produces the second largest makespan.
3. Solve the related TSP using the adjacent pairwise interchange heuristic (API) for the machine that produces the largest makespan.
4. If solution is improved and less than the second largest makespan, go to 2; otherwise stop.

Using Quicksort with the PH stages that require sorting and finding minimum and maximum values, it can be shown that the complexity of the PH is $O(n^2 \log n)$ because the complexity of the Quicksort procedure is $O(n \log n)$. For greater detail on the PH heuristic, see Al-Salem (2004).

4. THE TABU SEARCH APPROACH

TS is a meta-heuristic that has its origins in the combinatorial optimization procedures applied to some non-linear problems in the late 1970s. Principles of the technique in a broader sense are laid out in Glover (1989, 1990). Unlike other methodologies such as SA and GA, TS is a deterministic technique that does not utilize random numbers. This has been shown to be advantageous in Helal et al. (2000). A typical TS implementation starts with an initial solution that is to be perturbed by a set of alterations (moves) to move from this initial solution (or current solution after starting up) to neighbor solutions, searching for better results. The components of the TS methodology work to guide this search process and ensure its effectiveness.

The three basic components of TS are the tabu list (*T*), the long-term memory (*LTM*), and the aspiration level function. *T* is comprised of a list of recent moves that are not allowed (called tabu moves) at the current iteration because such moves would take the search process back to a previously tested solution. Prohibiting recently visited solutions avoids local optima and cycling, which saves computational time. *LTM* records features of the best trial solutions generated during a particular period of the search process. Features that are common are considered regional attributes of a good solution. The TS method then seeks new solutions that exhibit these features (search intensification). Alternatively, *LTM* may be used to guide the process to avoid those attributes and investigate regions that contrast with those examined so far (search diversification). In either use, *LTM* accumulates the experience gained during the search process to enhance the effectiveness of the search during subsequent search phases. The aspiration level function adds flexibility to choosing good moves by allowing the tabu status of a move to be overridden if this aspiration level is fulfilled. The most commonly used aspiration level is to allow a tabu move if an overall improvement can be achieved.

Definitions and implementation of these components are application-based. The tabu list *T* gives the technique its name and the speed of convergence of the heuristic is a function of *T*'s size. If it is too long, its use becomes time consuming while if it is too short, it becomes ineffective in escaping local optima. A tabu list size between five and twelve is recommended by the developer of TS (Glover, 1989) and by Zolfaghari and Liang (2002). Glover also recommended seven as the preferred tabu list size. Yet an appropriate size *T* should be determined by experimenting different sizes. We show the results of our experiments later in this paper.

4.1 The proposed TS implementation

The proposed TS implementation starts with an initial solution and works to improve that solution through the application of two perturbation schemes: intra-machine perturbation and inter-machine perturbation. A heuristic to develop the initial solution is proposed as described in the following subsection. Our implementation of the TS components is described in the sections that follow.

There are *N* jobs to be assigned to *M* parallel machines and sequenced such that the maximum completion time (C_{max}) among all machines is minimized. Processing times and setup times for each job are presented in a matrix format. For N jobs an $N \times N$ matrix is used where each entry a_{ij} ($i = j = 1, 2, \ldots, N$) represents the sum of the setup time and processing time for job *j* if it is scheduled immediately after job *i*. If $i = j$ it means that the job is in the first position on the machines. A matrix for each machine is used since the setup times are not only sequence-dependent, but also machine-dependent. The solution of the problem should describe how the *N* jobs are divided among the *M* machines such that each machine *k* ($k = 1, 2, \ldots, M$) is assigned n_k jobs where $\sum_k n_k = N$. Makespan for the solution is the maximum makespan among all *M* machines.

4.1.1 The initial solution

A complete schedule consists of *M* sequences of jobs for the *M* parallel machines. The following algorithm is

proposed to construct an initial complete solution. This initial solution algorithm is based on the well-known shortest processing time (*SPT*) dispatching rule as follows:

1. For each of the *M* machines:
   a. For each of the *N* jobs:
      i. Compute the SPT_{avg} index by averaging the sum of the job's processing times as if it is scheduled after each of the other jobs or the first on the machine.
2. Construct a complete schedule by assigning jobs, one at a time, to the machines based on ascending order of the SPT_{avg} index. Assign the job to the machine on which it has the smallest SPT_{avg} value. In case of a tie assign the job to the least utilized machine so far.
3. Identify the sequence of jobs on each machine and its makespan. The makespan for the complete solution is the maximum makespan across all machines.

4.1.2 Perturbation schemes

Two perturbation schemes are used in the search for better schedules. The first scheme is the intra-machine perturbation in which jobs on each machine are rearranged in order to find the shortest makespan for that machine. This perturbation scheme is meant to optimize the sequence of jobs on the machines. The second scheme is the inter-machines perturbation in which a job is taken from the busiest machine and inserted onto one of the other machines. This is meant to balance the assignment of jobs to machines. The busiest machine is the one that has the maximum makespan so far. We remove a job from busiest machine such that makespan is reduced the most after the job is removed. The search process iterates between these two perturbation schemes as described in the algorithm listing below.

4.1.3 The tabu list (*T*)

T will contain the best trial sequences selected during the perturbation actions. There will be a list for each machine with a size of 9. Glover (1989) who developed the generic technique of TS recommended *T* of 5 to 12 elements. In our experiments we found that for smaller problems, sizes of 8 and 9 led to the best makespan in 13 out of 18 problem sizes tested as shown in Table 1 where the shaded cells in the table indicate the best makespans. A problem size with no shaded cell indicates that the best makespan was obtained with a *T* size different than 8 or 9. Computational times were less than one second for all sizes and there was no trend that could be identified due to changing the size of *T*. Each *T* size was tested with 15 instances of each of the 18 problem sizes resulting 270 problems for each *T* size, a total of 18 × 15 × 8 = 2160 instances. In all instances, the processing and setup times were balanced where both times were drawn from Uniform [50, 100] as will be described in the Computational Experience section.

Similar experiments were conducted for larger problems with 15 instances (with balanced processing and setup times) for each problem size and each *T* size between 5 and 12, resulting 960 instances. *T* sizes of 9 and 10 gave the best TS performance for all cases. As shown in Table 2 size 9 led to the best makespan in most cases with a maximum time of about 6.7 minutes per problem.

Table 1. Results for tabu list (*T*) sizes of 8 and 9 for small problems

Problem Size	Makespan		CPU Time (seconds)	
(*N* / *M*)	*T* = 8	*T* = 9	*T* = 8	*T* = 9
6 / 2	395.27	397.20	0.15	0.44
6 / 4	251.27	249.07	0.02	0.01
7 / 2	494.73	502.00	0.20	0.21
7 / 4	264.27	259.53	0.26	0.02
8 / 2	521.20	522.07	0.08	0.26
8 / 4	270.47	268.93	0.03	0.07
8 / 6	240.27	235.47	0.29	0.03
9 / 2	607.33	614.53	0.29	0.31
9 / 4	346.47	347.73	0.86	0.93
9 / 6	249.27	244.33	0.05	0.06
10 / 2	645.33	649.60	0.34	0.37
10 / 4	360.33	363.27	0.98	0.95
10 / 6	259.13	254.67	0.08	0.06
10 / 8	232.00	230.07	0.08	0.09
11 / 2	722.53	724.47	0.44	0.44
11 / 4	276.30	375.80	0.99	0.96
11 / 6	273.80	265.87	0.04	0.04
11 / 8	235.20	232.87	0.12	0.11

Table 2. Results for tabu list sizes of 9 and 10 for large sized problems

Problem Size	Makespan		CPU Time (seconds)	
(*N* / *M*)	*T* = 9	*T* = 10	*T* = 9	*T* = 10
100 / 2	6165.27	6175.67	377.91	401.85
100 / 6	1934.87	1936.13	119.74	152.64
100 / 10	1142.40	1146.60	197.63	212.49
60 / 6	1174.40	1184.27	85.16	89.60
60 / 10	688.20	688.96	38.89	36.38
80 / 4	2366.80	2367.47	185.93	201.31
80 / 10	920.10	918.00	263.35	210.02
80 / 12	773.13	777.47	66.15	66.42

The nature of elements stored in *T* has a significant impact on the efficiency of the TS algorithm. In the current implementation, *T* records the best trial sequences found during the search process. Upon each perturbation, the resulting sequence is appended to *T* and the oldest sequence is pushed off of it. The elements of *T* could have been implemented as moves that are performed during perturbations. However, in the perturbation schemes described above, a job can be moved from, say, machine A to machine B resulting in a certain sequence, and after some iterations the same job may need to be moved back from B to A, which would result in a different sequence. If *T* contains the moves themselves, then a move from B back to A will be a tabu because it is the reverse of a recent move, although it may result in an improvement (or a step towards improvement) due to the existence of sequence-dependent setup times on each machine. When the sequences themselves are stored in *T*, the search

process has more flexibility to perform moves as long as they do not return to a previously tested solution, even if they are the reverses of each other. This way of implementation is more meaningful to pursue an optimal (or near-optimal) solution. During perturbations, T is checked to ensure that a resulting trial sequence is not on the list (not tabu sequence). If it is tabu, then the move that led to it is a tabu move and is canceled and the current trial sequence for the machine being worked is recovered.

4.1.4 The long-term-memory (*LTM*)

LTM consists of M matrices each of size $N \times N$. An element a_{ij} ($i = j = 1, 2, \ldots, N$) of the LTM_k matrix of machine k ($k = 1, 2, \ldots, M$) represents the number of times job i came after job j on machine k. Each time a new trial solution is identified or *LTM* matrices are used to create a new starting solution, the appropriate elements in the matrices are incremented by one. Thus *LTM* for a machine is a frequency matrix that records the number of times each job happened to be after each of the other jobs during the best trial solutions. The diagonal elements of the matrix are the number of times that each job happened to be in the first position on the machine.

To use the *LTM* matrices, the search process is divided into a number of phases. After every phase the frequency information in *LTM* is used to construct a complete solution to use it as a re-starting solution in the following phase. In the proposed implementation, *LTM* is used to generate a new re-starting solution based on the maximum entries in the matrix (search intensification). This is done by letting each job to be after the job it used to succeed for the largest number of times in the best trial solutions during the previous search phases. As such, the jobs are kept in the relative positions that previously resulted in good sequences, which is likely to result in a good quality re-starting solution for the remaining search phases. It should be noted that the incumbent solution (the best complete solution found so far) is maintained all the time and is only updated when an iteration results in a better solution.

When running the TS algorithm, we used *LTM* three times to generate three new re-starting solutions. Considering the first run to collect the frequency data for the *LTM* for the first time, the proposed algorithm has four search phases in total. The first phase is a complete run of the algorithm that starts with an initial solution generated by the SPT_{ave} index algorithm. Each phase after that is another complete run of the algorithm that starts with an *LTM*-based starting solution. The number of phases was determined by testing different values. Tables 3 and 4 compare the use of 4, 5, and 6 search phases for the small and large problems respectively. The number of instances solved for small problems (Table 3) is 2160 (as explained in section 4.1.3) for each of the three phase, resulting a total of $3 \times 2160 = 6480$ instances. For large problems the number of instances solved is 960 (see section 4.1.3) for a total of 2880 instances for all three phases. In both tables, Δ (4 to 6) indicates the change in makespan or computational time (CPU) if 6 phases are used instead of 4. At T size of 9 there was insignificant improvement in the solution quality (makespan) and significant increase in CPU (up to 25%) when the number of phases is increased from 4 to 6 as can be seen in Table 3. Similar results were obtained for the larger problems (Table 4). Consequently, the number of search phases was fixed to 4 in this implementation.

Table 3. The effect of more search phases on makespan and computational time – small problems

Problem Size N / M	Makespan				CPU Time (seconds)			
	4 Phases	5 Phases	6 Phases	Δ (4 to 6)	4 Phases	5 Phases	6 Phases	Δ (4 to 6)
6 / 2	396.40	396.40	396.40	0.00%	0.150	0.160	0.170	11.88%
6 / 4	251.73	251.73	251.73	0.00%	0.012	0.013	0.014	8.56%
7 / 2	495.07	495.07	495.07	0.00%	0.210	0.230	0.246	14.47%
7 / 4	265.07	263.07	263.07	0.75%	0.018	0.019	0.024	25.67%
8 / 2	522.60	522.60	521.87	0.14%	0.262	0.273	0.291	9.74%
8 / 4	271.27	271.20	270.80	0.17%	0.068	0.075	0.083	18.76%
8 / 6	242.07	241.93	240.67	0.58%	0.031	0.036	0.036	15.73%
9 / 2	603.80	603.80	603.80	0.00%	0.309	0.320	0.344	9.97%
9 / 4	346.40	346.40	346.20	0.06%	0.927	0.963	1.036	10.53%
9 / 6	249.53	249.53	249.53	0.00%	0.060	0.061	0.065	6.06%
10 / 2	645.33	645.20	645.20	0.02%	0.368	0.381	0.424	13.25%
10 / 4	361.60	361.60	361.60	0.00%	0.985	1.075	1.153	14.61%
10 / 6	259.47	259.00	259.00	0.18%	0.060	0.069	0.076	20.73%
10 / 8	232.00	231.40	231.40	0.26%	0.087	0.091	0.092	5.47%
11 / 2	721.27	721.27	721.27	0.00%	0.442	0.465	0.503	12.21%
11 / 4	374.33	374.33	373.47	0.23%	1.062	1.123	1.198	11.39%
11 / 6	274.53	273.00	272.60	0.70%	0.042	0.050	0.054	23.06%
11 / 8	235.60	235.60	235.60	0.00%	0.105	0.109	0.113	6.56%

Table 4. The effect of more search phases on makespan and computational time – larger problems

Problem Size N / M	Makespan				CPU Time (seconds)			
	4 Phases	5 Phases	6 Phases	Δ (4 to 6)	4 Phases	5 Phases	6 Phases	Δ (4 to 6)
100 / 2	6180.87	6180.87	6180.87	0.00%	133.22	143.06	155.01	14.06%
100 / 6	1940.60	1940.60	1940.60	0.00%	222.05	233.43	245.51	9.56%
100 / 10	1153.27	1152.40	1152.40	0.08%	379.36	390.83	412.90	8.12%
60 / 6	692.73	692.73	692.73	0.00%	87.71	91.77	94.55	7.23%
80 / 4	778.47	777.93	777.93	0.07%	223.81	234.76	244.67	8.53%
80 / 10	1179.27	1178.27	1178.00	0.11%	38.90	41.45	43.42	10.42%
80 / 12	920.80	920.80	920.80	0.00%	188.53	196.79	199.91	5.69%

4.1.5 The aspiration level function

We use the commonly used aspiration level function by which the tabu status of a move is overridden if an overall improvement is possible by that move. At any iteration, if the makespan of a trial solution resulting from a tabu move is better than current incumbent makespan then the tabu status of the move is ignored and the move is executed. The aspiration level function is checked only if a candidate move is found to be tabu.

4.1.6 Stopping criteria and iteration counters

The TS algorithm is run in four search phases, in each phase there are two iteration counters: the inter-machine perturbations counter and the overall iteration counter (referred to simply as the iteration counter). The inter-machine perturbations counter counts the number of iterations performed in the inter-machine perturbation step where the maximum number of this inter-machine perturbation = Max{25, $\lfloor N/(LTM\_restart + 1) \rfloor$}, where $\lfloor x \rfloor$ is floor of x. The minimum number of iterations is thus 25.

Additional iterations resulted in longer computational time with no significant improvement. It should be noted that the inter-machine perturbation counter is reset each time the incumbent solution is updated. So the algorithm switches to the intra-machine perturbation if no improvements are obtained in a number of iterations equal to Max{25, $\lfloor N/(LTM\_restart + 1) \rfloor$}. If improvements were obtained the counter is reset to zero and the algorithm continuous in this perturbation scheme.

On the other hand the iteration counter keeps track of the overall number of iterations. In each iteration, the intra-machine perturbations are performed for every machine and then the algorithm switches to perform the inter-machine perturbations. The iteration is over if the inter-machine perturbation iterations are performed without a change in the current solution. The number of overall iterations varies with the number of machines, M and it is equal to $\lfloor M^2/(LTM\_restart$ counter+1)$\rfloor$, where *LTM_restart* is the number of search phases. The proposed TS algorithm is described below. The following parameters are used in the description.

- Ω^0 and Ω^* : A complete solution where Ω^0 is the current solution and Ω^* is the incumbent solution.
- C^0_{max} and C^*_{max}: Makespan for Ω^0 and Ω^* respectively.
- T_k: Tabu list for machine k where k = 1, 2, …, M.
- LTM_k: Long term memory for machine k where k = 1, 2, …, M.
- n_k: number of jobs on machine k where k = 1, 2, …, M.
- *LTM_restart*: number of times *LTM* is utilized to develop new starting solutions.

I. Initialization & Reinitialization

Step 1. Let T_k and LTM_k be zero matrices. Set *LTM_restart* = 3, *LTM_restart* counter = 0, and the iteration counter = 0. Create an initial complete solution using the SPT_{arg} algorithm. Let this solution be the current solution Ω^0 and its makespan be $C^0_{\max}$. Let $\Omega^* = \Omega^0$ and $C^*_{max} = C^0_{max}$. Go to *Step* 3.

Step 2. Use the LTM_k to construct a complete new starting solution. Increment the *LTM_restart* counter by one. Let the newly generated solution be the current solution Ω^0 and its makespan C^0_{max}. If $C^0_{max} < C^*_{max}$, then let $\Omega^* = \Omega^0$ and $C^0_{max} = C^*_{max}$.

II. Perturbation

Step 3. Intra-machine Perturbation: ∀ machine k, perturb the sequence on k by reinserting each job in the sequence, one at a time, into each of the other $n_k - 1$ positions. While doing so, keep the relative positions of other jobs on the machine intact. Let the resulting complete schedule after perturbing all machines become the current solution Ω^0 and its makespan become C^0_{max}. If $C^0_{max} < C^*_{max}$ then let $\Omega^* = \Omega^0$ and $C^*_{max} = C^0_{max}$. ∀ k, increment LTM_k by one to reflect the new Ω^0. Update T_k with Ω^0.

Step 4. Inter-machine perturbation: Set the inter- achine iteration counter equal to 0. For the current solution Ω^0 take each job on the busiest machine (the machine with the largest makespan) one at a time, and try inserting it into all possible

positions on the other $M - 1$ machines. At any one time, only the busiest machine and one other machine should be altered by this insertion action. Identify the best resulting solution and let it replace the current Ω^0 and its makespan replace the current C^0_{max} . Increment the inter-machine iteration counter by one. Update LTM_k and T_k with the new Ω^0. If $C^0_{max} < C^*_{max}$, then let $\Omega^* = \Omega^0$ and $C^*_{max} = C^0_{max}$ and restart the current inter-machine perturbation step, Else if no improvement in Ω^* is achieved in this step and the inter-machine iteration counter is equal to the Max{25, $\lfloor N/(LTM\_restart + 1 \rfloor$} Then continue to *Step* 5.

III. Stopping criteria

Step 5. If the iteration counter $\leq$ (M^2 / (*LTM_restart* counter + 1)) then go to *Step* 3, Else if *LTM_restart* counter $\leq$ 3 Then go to *Step* 2, Else record the final solution Ω^* and its C^*_{max} and STOP.

5. COMPUTATIONAL ANALYSIS

Ideally, it will be best to compare the performance of the proposed TS to benchmark problems. However, such problems do not exist for the problem addressed in this paper. Therefore, two sets of experiments were conducted where the TS performance was compared to optimal solutions for small problems (up to 9 jobs and 4 machines), and to the PH algorithm for larger problems (up to 120 jobs and 12 machines). In the first set, solutions from both TS and BH algorithms were also compared to the optimal solutions, which were obtained using OPL (Optimization Programming Language) to solve the MIP presented in section 2. OPL Studio 3.7 is a modeling environment that utilizes OPL as a modeling language and CPLEX 7.0 as a solver (see Van Hentenryck (2001) for more details). In the second experiment, large problem instances were generated and solved using both TS and the PH algorithms. Optimal solutions were unattainable for large instances.

The processing and setup times were randomly drawn from two uniform distributions: U[50, 100] and U[125, 175]. Uniform distribution is widely used for this purpose as many real world instances match such data settings. It has high variance, which allows testing the algorithms under unfavorable conditions (Weng et al., 2001). The selection of the uniform distribution bounds for processing and setup times determines the level of dominance. That is, when processing and setup times are balanced (denoted B), they are both drawn from U[50, 100]. When processing times are dominant (denoted by P), the processing and setup times are drawn from U[125, 175] and U[50, 100] respectively. When the setup times are dominant (denoted as S), the processing and setup times are drawn from [50, 100] and [125, 175] respectively. Both algorithms were implemented in C++ and the experiments were run on a Pentium IV- 1.7 GHz personal computer. Data sets for both small and large problems as well as solutions using TS and the PH algorithm are available at SchedulingResearch (2005).

5.1 Analysis of performance for small problems

For the small problems the percent deviation δ from the optimum is calculated using (10) where C is the average makespan.

$$\delta = \frac{C_{heuristic} - C_{Optimal}}{C_{Optimal}} \times 100 \qquad (10)$$

Optimal solutions are available for up to nine jobs on two machines and eight jobs on four machines. It became prohibitively time consuming to solve for larger problems, and therefore, the MIP solver was interrupted after running hours without reaching a solution. For example, for two-machine problems, the computational time on average was 12 seconds only when the number of jobs was 6 and grew exponentially to take an average of 5 hours when the number of jobs was increased to 9. Similarly, for 4 machines, the average time was 48 seconds when the number of jobs was 6, and increased to 1.5 hours when the number of jobs was increased to 8. Table 5 shows the average percent deviation of both algorithms from the optimal solution for the balanced (B), processing-time-dominated (P), and setup-time-dominated (S) problems. The number of random instances generated and solved for each configuration was 15 resulting in a total of $3 \times 7 \times 15 = 315$.

One can see that TS outperforms the PH algorithm in most cases. For the 2-machine problems, TS was better in all cases and for the 4-machine problems it was better for seven out of nine configurations. For the 2-machine problems, TS was between 0.2 to 1.0% away from the optimal while the PH algorithm was 1.5 to 6% away. For the 4-machine problems, the two algorithms were close; TS was about 1 to 5% away from the optimum compared to 2 to 4.5 % for PH. It can also be observed that both algorithms tend to perform relatively better for problems with dominant processing or setup times than for the balanced problems. But there are no identifiable trends that can be observed for any of the three cases as the number of jobs or number of machines increases, possibly due to the small sizes of the problems.

In summary, TS is better than PH for the 2-machine problems but the two algorithms approach each other for the 4-machine problems. It is worth noticing that the PH algorithm maintains its level of performance for the 2- and 4-machine cases. On the other hand, TS seems to be more sensitive to increasing the number of machines, as it was more inferior to itself when the number of machines increased from 2 to 4.

Table 5. Percent deviation of TS and PH algorithm from optimal solution

	$M = 2$								$M = 4$					
	N								N					
	6		7		8		9		6		7		8	
	TS	PH	TS	PH	TS	PH	TS	PH	TS	PH	TS	PH	TS	PH
B	0.42	3.70	0.84	4.90	0.99	3.30	0.88	5.69	2.76	3.30	5.15	4.22	2.51	2.79
P	0.39	1.69	0.44	3.19	0.21	2.67	0.42	2.57	1.14	2.35	2.93	3.44	0.88	2.12
S	0.29	1.97	0.54	2.16	0.54	2.67	0.85	2.85	2.52	2.86	2.74	3.41	2.62	2.31

When both algorithms are compared to each other, with TS being the baseline, the impact of increasing the number of machines becomes clear (see Table 6). Note that optimal solutions are not available for many cases. The negative values in the table indicate that the PH algorithm is better than TS. As the number of machines increases, the PH algorithm performance tends to be closer to the performance of the TS. However this trend does not continue beyond 6 machines indicating that the TS may be more robust and better for larger problems.

5.2 Analysis of performance for large problems

The TS results are used here as the reference value and C_{TS} (Makespan by TS) replaces $C_{Optimal}$ to calculate δ in (10). The average relative difference between both algorithms are shown in Table 7 based on solving 15 instances per machine-job configuration for balanced, dominant processing times, and dominant setup times (a total of 15 × 6 × 6 × 3 = 1620 instances). The negative values for some of the 2- and 4-machine problems indicate that the PH performs better than the TS. This happened in instances with large number of jobs on the two and four machines; however, TS was always better with instances with more than four machines. In terms of performance frequencies, TS was better than PH in 95 out of 108 cases. In terms of percent deviation, TS outperformed PH by up to 8% in some cases.

The difference between both algorithms is clearer for the balanced problems. Smaller differences are observable when either the processing times or the setup times are dominant. In addition, for the same number of machines, the difference between TS and PH decreases as the number of jobs increases. This is clearly observable for 2, 4, 5 and 8 machines. It is also noted that the difference is larger for the balanced problems than for problems with dominant processing or setup times and this difference is even bigger for the larger number of machines. Recall that the larger the deviation, the better the TS performs over the PH algorithm.

Table 6. Percent deviation of the PH algorithm from the TS

	$M = 2$						$M = 4$					
	N						N					
	6	7	8	9	10	11	6	7	8	9	10	11
B	3.28	4.06	2.29	4.77	4.34	2.83	0.59	-0.74	0.41	4.04	2.49	2.85
P	1.30	2.75	2.46	2.14	2.39	1.94	1.22	0.54	1.24	2.65	1.79	0.85
S	1.68	1.63	2.12	1.99	2.30	1.72	0.38	0.72	-0.24	3.28	3.32	1.47
	$M = 6$						$M = 8$					
	N						N					
	n/a	n/a	8	9	10	11	n/a	n/a	n/a	n/a	10	11
B	--	--	0.59	-0.14	-0.16	-3.88	--	--	--	--	2.12	3.00
P	--	--	-1.22	-2.68	-2.59	-3.80	--	--	--	--	1.8	1.58
S	--	--	1.09	1.09	-0.81	-0.75	--	--	--	--	2.16	2.38

Table 7. Percent deviation of the PH algorithm relative to TS for large problems

	$M = 2$						$M = 4$					
	N						N					
	20	40	60	80	100	120	20	40	60	80	100	120
B	2.49	1.40	-0.08	-0.31	-0.85	-1.49	4.54	3.24	1.83	1.13	0.89	0.65
P	1.82	0.95	0.06	-0.30	-0.88	-0.98	3.32	2.11	0.83	0.72	-0.03	-0.11
S	1.67	1.17	0.13	-0.31	-0.69	-1.18	3.53	1.96	0.77	0.29	-0.49	0.00
	$M = 6$						$M = 8$					
	N						N					
	20	40	60	80	100	120	20	40	60	80	100	120
B	6.92	5.49	4.51	3.68	1.69	1.76	7.09	6.26	6.54	4.65	4.54	2.21
P	3.85	2.13	1.31	2.03	0.82	0.57	3.46	3.78	3.14	1.75	1.87	0.44
S	3.60	2.58	1.95	2.29	0.71	0.90	4.47	3.89	2.22	1.69	1.46	1.65
	$M = 10$						$M = 12$					
	N						N					
	20	40	60	80	100	120	20	40	60	80	100	120
B	0.47	7.35	7.01	6.92	4.77	5.59	2.05	8.38	8.74	6.59	7.24	6.82
P	-1.42	4.36	3.73	2.03	0.90	1.01	1.56	4.47	2.98	3.17	3.35	0.91
S	-0.42	5.40	3.47	2.03	1.65	1.54	0.61	4.41	3.41	3.59	3.33	0.88

When the number of jobs is fixed and the number of machines is increased, the percent deviation between the two algorithms increases. For the 2-machine problems both algorithms can be equal or sometimes the PH algorithm is better. But for more machines TS is always better and the difference reaches 8% for the largest problems. And like before, TS seems to perform better for the balanced problems than for problems with dominant processing or setup times. As for computational times, TS takes in general more time (see Tables 1 through 4) than the PH algorithm, which took less than one minute to reach its solution for the largest instances.

In conclusion, TS outperformed the PH algorithm in most cases; yet it seems to be unstable for the smaller problems. As more machines are considered, TS shows observable advantage and robustness over the PH algorithm. These conclusions are true for the balanced problems as well as for the problems with dominant processing or setup times, but it is more obvious in the balanced problems.

6. POTENTIAL TABU SEARCH IMPROVEMENTS

TS offers the flexibility of redefining all of its three components to best fit the application. This includes the selection of the tabu list elements, the nature of information included in the long term memory, and the definition of the aspiration level. Since TS is not a randomized technique, its performance is only a function of the way its components are implemented. The difference in the proposed TS algorithm performance for small and large problems indicates the need to consider redefining the TS components. Possible directions for further improvement in the proposed implementation include:

- The use of a variable size tabu list T to fit the different problem sizes. The list size can significantly impact the effectiveness and efficiency of the algorithm.
- The long term memory *LTM*, was defined as a frequency matrix. It could be worth investigating including information about the relative improvements occurring during the search process with the frequency information in *LTM* as a potential approach to improve the TS performance.
- Multidimensional aspiration criteria can be investigated in addition to the overall improvement in the objective function.

Additional perturbation schemes may also be investigated.

7. CONCLUSIONS AND FURTHER EXTENSIONS

A tabu search algorithm (TS) was proposed in this paper to solve the unrelated parallel machine scheduling problem with machine- and sequence-dependent setup times. The problem is known to be NP-hard, yet few attempts have been made to develop heuristic techniques to solve it. The TS performance was compared to the Partitioning Heuristic (PH), an existing algorithm introduced by Al-Salem (2004), and to the optimal solution for small problems. The results favored the proposed TS algorithm for large and small problems; yet, TS showed slightly less robustness for the smaller problems. The TS also outperformed the PH in terms of the number of times it did better than PH.

Based on the computational analysis, directions for future modifications to improve the performance of the proposed TS were suggested. Those included the use of a variable size tabu list, more search-based information in the long term memory matrices, the use of multi-dimensional aspiration criteria, and investigating other perturbation schemes for the search process.

REFERENCES

1. Al-Salem, A. (2004). Scheduling to minimize makespan on unrelated Parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar*, 17: 177-187.
2. Allahverdi, A., Gupta, J., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27: 219-39.
3. Azizoglu, M. and Kirka, O. (1999). Scheduling jobs on unrelated parallel machines to minimize regular total cost functions. *IIE Transactions*, 31: 153-159.
4. Bank, J. and Werner, F. (2001). Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and Computer Modelling*, 33: 363-383.
5. Bruno, L.G., Coffman, E.G., and Sethi, R. (1974). Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17: 382-387.
6. Cheng, T. and Sin, C. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operation Research*, 47: 271-292.
7. Franca, P.M., Gendreau, M., Laporte, G., and Muller, F.M. (1996). A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, 43: 79-89.
8. Garey, M.R. and Johnson. D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman and Company, USA.
9. Ghirardi, M. and Potts, C.N. (2005). Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165(2): 457-467.
10. Glass, C.A., Potts, C.N., and Shade, P. (1994). Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modeling*, 20(2): 41-52.
11. Glover, F. (1989). Tabu search - Part I. *ORSA Journal of Computing*, 1: 190-206.
12. Glover, F. (1990). Tabu search - Part II. *ORSA Journal of Computing*, 2: 4-32.
13. Graves, S.C. (1981). A review of production scheduling. *Operation Research*, 29: 646-675.

14. Guinet, A. (1991). Textile production systems: A succession of non-identical parallel processor shops. *Journal of the Operational Research Society*, 42(8): 655-671.
15. Hariri, A.M.A. and Potts, C.N. (1991). Heuristics for scheduling unrelated parallel machines. *Computers and Operations Research*, 18(3): 323-331.
16. Helal, M., Badr, M., and Huzayyin, A. (2000). An investigation of the group scheduling heuristics in a flow-line cell. *Current Advances in Mechanical Design & Production. Proceedings of 7th Cairo University International MDP Conference*, Cairo, Egypt, pp. 361-373.
17. Helal, M. and Hosni, Y. (2003). A tabu search approach for the non-identical parallel-machines scheduling problem with sequence-dependent setup times. *Proceedings of the IERC2003*, *IIE Annual Research Conference*, Portland, Oregon.
18. Horn, W. (1973). Minimizing average flow time with parallel machines. *Operations Research*, 21: 846-847.
19. Kim, D.W., Kim, K.H., Jang, W., and Chen, F.F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer Integrated Manufacturing*, 18: 223-231.
20. Kim, D.W., Na, D.G., and Chen, F.F. (2003). Unrelated parallel machine scheduling with setup times and total weighted tardiness objective. *Robotics and Computer Integrated Manufacturing*, 19: 173-181.
21. Kurz, M.E. and Askin, R.G. (2001). Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, 39: 3747-3769.
22. Lancia, G. (2000). Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan. *European Journal of Operational Research*, 120: 277-288.
23. Lawler, E.L., Lenstra, J.K., Rinnooy-Kan, A.H., and Shmoys, D.B. (1993). Sequencing and scheduling: Algorithms and complexity. *Handbooks in Operations Research and Management Science 4*: *Logistics of Production and Inventory*, North Holland, Amsterdam, pp. 445-524.
24. Lee, H., Bhaskaran, K., and Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, 100: 464-474.
25. Liaw, C.F., Lin, Y.K., Chen, C.Y., and Chen, M. (2003). Scheduling unrelated parallel machines to minimize total weighted tardiness. *Computers & Operations Research*, 30: 1777-1789.
26. Mokotoff, E. (2001). Parallel machine scheduling problems: A survey. *Asia-Pacific Journal of Operational research*, 18: 193-242.
27. Radhakrishnan, S. and Ventura, J.A. (2000). Simulated annealing for parallel machine scheduling with earliness/tardiness penalties and sequence-dependent set-up times. *International Journal of Production Research*, 38: 2233-2252.
28. Randhawa, S.U. and Kuo, C.H. (1997). Evaluating scheduling heuristics for non-identical parallel processors. *International Journal of Production Research*, 35: 969-981.
29. SchedulingResearch (2005), http://www.SchedulingResearch.com, a web site that includes benchmark problem data sets and solutions for scheduling problems.
30. Sivrikaya-Serifoglu, F. and Ulusoy, G. (1999). Parallel machine scheduling with earliness and tardiness penalties. *Computers and Operations Research*, 26: 773-787.
31. Srivastava, B. (1997). An effective heuristic for minimizing makespan on unrelated parallel machines. *Journal of the Operational Research Society*, 49: 886-894.
32. Van Hentenryck, P. (2001). *ILOG OPL Studio 3.5 Language Manual*, ILOG, France.
33. Weng, M., Lu, J., and Ren, H. (2001). Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International Journal of Production Economics*, 70: 215-226.
34. Yalaoui, F. and Chu, C. (2002). Parallel machine scheduling to minimize total tardiness. *International Journal of Production Economics*, 76: 265-279.
35. Zolfaghari, S. and Liang, M. (2002). Comparative study of simulated annealing, genetic algorithms and tabu search for solving binary and comprehensive machine-grouping problems. *International Journal of Production Research*, 40: 2141-2158.