"LOCAL SEARCH HEURISTICS FOR SINGLE MACHINE TARDINESS SEQUENCING"

by
H.A.J. CRAUWELS*
C.N. POTTS**
and
L. VAN WASSENHOVE***
94/52/TM

* KIHDN, Sint-Katelijine-Waver, Belgium.

** Faculty of Mathematical Studies, University of Southampton, UK.

*** Professor of Operations Management and Operations Research at INSEAD, Boulevard de Constance, 77305 Fontainebleau Cedex, France.

Printed at INSEAD, Fontainebleau, France

Local Search Heuristics for Single Machine Tardiness Sequencing

H. A. J. Crauwels
KIHDN, Sint-Katelijne-Waver, Belgium

C. N. Potts
Faculty of Mathematical Studies, University of Southampton, U.K.

L. N. Van Wassenhove
INSEAD, Fontainebleau, France

March, 1994

Abstract

This paper presents several local search heuristics for the problem of scheduling a single machine to minimize total weighted tardiness. A genetic algorithm is developed which employs a new binary encoding scheme to represent solutions and uses a heuristic method to convert the representation into a sequence. This algorithm is compared to descent, simulated annealing and tabu search methods on a large set of test problems. The computational results indicate that tabu search performs better than the other two neighbourhood search methods. However, the genetic algorithm generally produces superior solutions and it also appears to be a robust heuristic.

Subject classifications: Production/scheduling, Deterministic single machine, Heuristic, Local search.

1 Introduction

The single machine tardiness problem may be stated as follows. Each of n jobs (numbered $1,\ldots,n$) is to be processed without interruption on a single machine that can handle only one job at a time. Job i ($i = 1,\ldots,n$) becomes available for processing at time zero, requires an integer processing time p_i, has a positive weight w_i and a due date d_i. For a given processing order of the jobs, the (earliest) completion time C_i and the tardiness $T_i = \max\{C_i - d_i, 0\}$ of job i ($i = 1,\ldots,n$) can be computed. In the total *weighted* tardiness problem, the objective is to find a processing order of the jobs that minimizes $\sum_{i=1}^{n} w_i T_i$. When all job weights are equal, minimizing $\sum_{i=1}^{n} T_i$ is called the total tardiness problem.

The total weighted tardiness problem is (unary) NP-hard (Lawler [6] and Lenstra *et al.* [7]), whereas the total tardiness problem is (binary) NP-hard (Du and Leung [2]). There are well-known dynamic programming and branch and bound algorithms

for both the total tardiness and the total weighted tardiness problem. Abdul-Razaq *et al.* [1] discuss several of these algorithms for the total weighted tardiness problem, and a survey of algorithms for the total tardiness problem is given by Potts and Van Wassenhove [12].

Each of the available enumerative algorithms may have heavy requirements for both computation time and core storage, especially when the number of jobs is large. Consequently, available computer resources are not always adequate to obtain exact solutions. Therefore, the quest for good and robust heuristics is of great practical relevance.

As indicated by Potts and Van Wassenhove [13], simple dispatching rules (EDD, SWPT, COVERT or AU) do not consistently provide good quality solutions. Even the more sophisticated Wilkerson-Irwin heuristic [14] performs rather poorly on more difficult problems. In recent years, much attention and many papers have been devoted to a number of *local search* heuristics for 'solving' combinatorial optimization problems. Neighbourhood search forms one category of local search heuristics. Starting from an arbitrary solution, these methods construct a sequence of solutions by repeatedly moving from the current solution to a solution in an appropriately defined neighbourhood. The most common method of this type is descent in which the transition from neighbour to neighbour continues until a better solution can no longer be found. The resulting solution is a local optimum. The main weakness of descent is that the local optimum may deviate significantly from the true (global) optimum. Simulated annealing and tabu search are local search strategies designed to overcome this weakness. They allow neighbourhood moves which result in a deterioration of the objective function value and can therefore potentially escape from a local optimum. Genetic algorithms form another category of local search methods. In contrast to neighbourhood search, a genetic algorithm works with a population of solutions. By allowing the solutions to interact, 'children' are produced that hopefully retain the good characteristics of their parents. An introductory overview of these methods and some applications are given by Goldberg [5] and Pirlot [9].

Potts and Van Wassenhove [13] propose descent and simulated annealing methods for the total weighted tardiness problem. They also test these methods on the total tardiness problem. In this paper, we present a tabu search procedure in Section 2, and a genetic algorithm is developed in Section 3: both methods are applicable to the total weighted and total tardiness problems. Computational results for the four methods on the test data sets of Potts and Van Wassenhove [13] are reported in Section 4. Lastly, Section 5 summarizes the main conclusions.

2 Tabu search

Tabu search is a neighbourhood search strategy designed to allow escape from local optima. Let N be some suitably chosen neighbourhood structure, and let x_c be the current solution. Tabu search executes a move from x_c to x, where x is the best solution in $N(x_c)$ (or some subset of $N(x_c)$), even if x is no better than x_c in terms of objective function value. If, as is usually the case, the neighbourhood is symmetric, i.e., if x_c belongs to the neighbourhood $N(x)$ of x whenever $x \in N(x_c)$, there is a danger of

cycling when, at the next step, we explore $N(x)$. The possibility exists that x_c could be the best solution in $N(x)$, in which case the procedure would immediately reset x_c to be the current solution, and thereafter oscillate between x and x_c.

To avoid cycling, a *tabu list* is created to prohibit certain moves. After a move is executed, one (or sometimes several) of its attributes or characteristics is stored in this list, and moves which would reverse this attribute are *tabu*. The list is organized cyclically so that a move remains tabu only during a prespecified number of iterations. Tabu moves are never allowed unless an *aspiration* criterion is used which allows the tabu status of a move to be overridden.

A tutorial description of tabu search is given by Glover [4]. To develop a tabu search based solution method for our sequencing problem, there are a number of key elements to consider: choosing the neighbourhood structure, selecting an attribute to store on the tabu list, and specifying which moves are forbidden after consideration of any aspiration criteria. There are other optional features such as devices to diversify the search. The aim of diversification is to drive the search into unexplored regions of the solution space.

2.1 Choice of a suitable neighbourhood

Potts and Van Wassenhove [13] use an interchange neighbourhood in their simulated annealing algorithm. Two positions u and v ($1 \leq u < v \leq n$) in the current sequence of jobs are selected, and a new sequence is generated by interchanging the jobs in positions u and v. In our tabu search method, sequences are generated for several (u,v) pairs, and the objective function is evaluated for each of these neighbours. A move to the best non-tabu neighbour is then executed.

The (u,v) pairs can be selected at random, or with a systematic and repeated scanning of all job positions, e.g. $(u,v) = (1,2),(1,3),\ldots,(n-1,n)$. The neighbourhood size is $n(n-1)/2$, so there is a large computational investment to find the best neighbour. Our initial experiments show that it is preferable to perform more iterations with a restricted neighbourhood. One method of restricting the neighbourhood is to consider only adjacent job interchanges in which $v = u + 1$. However, the moves under this approach are too limited to search different regions of the solution space. As an alternative, we create a small subset containing n general interchange neighbours which are selected by generating (u,v) pairs at random. However, this random neighbourhood generation is stopped if a non-tabu neighbour is found which improves the current objective function value, and the corresponding move is executed.

2.2 The tabu list and aspiration levels

After each move in which jobs in positions u and v are interchanged, the job in position v of the new sequence is stored on the tabu list together with the corresponding objective function value of the new sequence. Any neighbour that is created by interchanging a job which appears on the tabu list is prohibited unless an aspiration level criterion is satisfied. This criterion is satisfied if the objective function value of the sequence created by the interchange is strictly smaller than the stored value corresponding to a

job on the tabu list that is interchanged during the creation of this new solution.

2.3 Intensification and diversification.

According to Glover [3], a good heuristic search strategy is to alternate between phases involving intensification and diversification. The tabu list is a short-term memory device which helps to diversify the search. However, to strengthen the inherent intensifying and diversifying components already present in tabu search, we incorporate two additional devices. Firstly, as in the simulated annealing heuristic of Potts and Van Wassenhove [13], a descent algorithm is applied to find a local optimum whenever an increase of the objective function is followed immediately by a decrease. The descent algorithm uses the adjacent interchange neighbourhood. Because the tabu list is ignored during the descent procedure, the result is an intensified search to a local optimum. However, to avoid disruption to the tabu search algorithm, the search continues from the starting sequence that is input into descent, rather than from the local optimum sequence that is output.

To motivate the use of our second device, suppose that during a number of successive iterations the best move does not change the objective function value because the two jobs which are interchanged are both early. A stronger diversifying move is then necessary to drive the search into a new region of the solution space. This is achieved by imposing some additional restrictions on which (u, v) pairs are allowed. We select the best move from three candidates; u is randomly chosen from the first $n/4$ positions in the sequence and v is randomly selected from the last $n/4$ positions in each candidate.

3 A genetic algorithm

Genetic algorithms, which are motivated by natural selection and evolution, form another category of local search methods. Key components of a genetic algorithm include a 'chromosomal' representation of solutions, a mechanism to generate an initial population, a measure of solution fitness (based on the objective function), and genetic operators that combine and alter current (parent) solutions to form new (child) solutions. An introductory account of the theory of genetic algorithms, as well as the main developments and applications, is given by Goldberg [5].

3.1 Solution encoding

The success of a genetic algorithm relies on an effective chromosomal encoding of solutions. A chromosome is a string of characters from a finite alphabet, where the alphabet often comprises binary digits. An obvious method for many scheduling problems is to encode the solutions as sequences. However, disadvantages of such an encoding include the difficulty in applying the genetic operators and the possibility that child solutions do not inherit important parental features. Therefore, we propose a binary encoding of solutions which is described below.

We assume that the jobs are renumbered in shortest weighted processing time (SWPT) order so that $p_1/w_1 \leq \ldots \leq p_n/w_n$, where $d_i \leq d_{i+1}$ whenever $p_i/w_i =$

p_{i+1}/w_{i+1} $(i = 1, \ldots n-1)$. A chromosome consists of a string of n binary digits, where the elements correspond to jobs $1, \ldots, n$ respectively. If the element corresponding to any job i is set to 1, then i is considered to be early; otherwise, this element is 0 and job i is regarded as being late. The set of early jobs, as defined by a such a chromosome, cannot necessarily be scheduled so that each job is completed by its due date. In Section 3.3, we present a heuristic procedure that constructs a sequence for which the actual early and late jobs match the chromosome as closely as possible.

3.2 Initial population

A population consists of M chromosomes. At the start of the algorithm, an initial population must be generated. One possible approach is to use a completely random assignment, where the probability of setting each element to 1 is equal to 0.5 for all chromosomes. However, we use a more sophisticated method which is described below.

In the first chromosome of the initial population, all elements are set to 1. This chromosome is required to find a quick solution to problems where all the jobs can be sequenced on time (in order of non-decreasing due dates, or EDD order). The $M-1$ remaining chromosomes are generated as follows. The number of elements set to 1 (corresponding to early jobs) is related to the average tardiness factor TF which is defined by $\mathrm{TF} = 1 - d_{\mathbf{avg}}/P$, where $d_{\mathbf{avg}} = \sum_{i=1}^{n} d_i/n$ is the average due date and $P = \sum_{i=1}^{n} p_i$ is the total processing time. Clearly, a relatively small value of TF arises when due dates are larger, and therefore there are more early jobs in an optimal solution. The value of p_i/w_i also influences the setting of elements corresponding to job i: the likelihood that job i is early in an optimal solution increases if p_i/w_i is small. Thus, chromosomes $2, \ldots, M$ are randomly generated, where the probability P_i of setting the element corresponding to job i $(i = 1, \ldots, n)$ to 1 is defined as

$$P_i = 1 - \max\{0.01, \min\{0.99, n\mathrm{TF}(p_i/w_i)/\sum_{j=1}^{n} p_j/w_j\}\}$$

Our method of generating an initial population allows the possibility that some of these M chromosomes are replaced. Firstly, for each chromosome, a sequence of jobs is constructed using the method described below in Section 3.3, and its total weighted tardiness is evaluated. Also, the average total weighted tardiness for these M chromosomes is computed. Then, $M/2$ additional chromosomes are created, a subset of which may replace the 'bad' chromosomes from the original M. In the first of these chromosomes, all elements are set to 0 (all jobs are considered to be late), while the other $M/2-1$ additional chromosomes are randomly generated using a probability $P_i = 1 - \mathrm{TF}$ of setting each element to 1. The following procedure is used to test whether an original chromosome with the largest total weighted tardiness should be replaced by an additional chromosome. The condition for replacement is that the total weighted tardiness for the additional chromosome should be less than the average total weighted tardiness that is computed for the original M chromosomes, and also less than the total weighted tardiness of the chromosome that it replaces. This test for replacement is applied for each of the $M/2$ additional chromosomes.

3.3 Chromosome evaluation

A method is needed to convert a chromosome into a solution which is defined by a sequence of jobs. Having found a sequence, the objective function value can be computed and converted into a fitness value for the chromosome. Based on the values of the elements in the chromosome under consideration, we define sets of early and late jobs. Firstly, the early set is checked for feasibility and, where necessary, jobs are transferred from the early to the late set. Having obtained a feasible early set, we use a heuristic method to construct a schedule. Our heuristic first schedules the early jobs as late as possible in EDD order, and subsequently attempts to insert jobs of the late set between the early jobs. Finally, any remaining jobs are appended to the resulting partial sequence in SWPT order. We now give precise details of our heuristic.

Decoding heuristic

Step 1. Construct a list E of early jobs in EDD order and let L be a list of late jobs. Let $E = (\pi(1), \ldots, \pi(n_e))$, where n_e is the number of jobs in E. Set $i = 0$, $j = 1$ and $t = 0$.

Step 2. If $t + p_{\pi(j)} \leq d_{\pi(j)}$, set $i = i+1$, $\sigma(i) = \pi(j)$ and $t = t + p_{\pi(j)}$. If $t + p_{\pi(j)} > d_{\pi(j)}$, remove job $\pi(j)$ from E and append it to L. Set $j = j + 1$. If $j < n_e$, repeat Step 2. If $j = n_e$, set $n_e = i$.

Step 3. Reorder jobs of the list L of late jobs in SWPT order. Compute latest start times for the jobs of E using $s_{\sigma(n_e)} = d_{\sigma(n_e)} - p_{\sigma(n_e)}$, and $s_{\sigma(i)} = \min\{s_{\sigma(i+1)}, d_{\sigma(i)}\} - p_{\sigma(i)}$ for $i = n_e - 1, n_e - 2, \ldots, 1$. Compute $R = 2n \max_{i=1,\ldots,n}\{p_i/w_i\} / \sum_{i=1}^{n} p_i/w_i$. Set $t = 0$ (t denotes the completion time of the current partial sequence), and set e to be the first job in E.

Step 4. If $s_e = t$, go to Step 8. Otherwise, set i to be the first job in L and, if possible, find the first job j for which $t + p_j > d_j$ (the job is late) and $p_j/w_j \leq R p_i/w_i$ (the ratios p_i/w_i and p_j/w_j are relatively close). If j cannot be found, go to Step 8. If $t + p_j \leq s_e$, go to Step 5 (we search to find whether there is a better job than j in L to be scheduled before job e); otherwise, if $t + p_j > s_e$, go to Step 6 (we search to find whether there is a job of L that can be scheduled before job e without making it late).

Step 5. Search for the first job k that follows job j in L for which $t + p_k > d_k$ (the job is late), $t + p_k \leq s_e$ (the job can be scheduled before job e) and one of the two following conditions is satisfied:

(a) $t + p_j + p_k \leq s_e$ (both j and k can be scheduled before job e) and $V_k > V_j$, where $V_j = w_j(t + p_j - d_j)$ and $V_k = w_k(t + p_k - d_k)$;

(b) $t + p_j + p_k > s_e$ (jobs j and k cannot both be scheduled before e) and $V_{kej} \leq V_{jek}$, where $V_{jek} = w_j(t + p_j - d_j) + w_k(t + p_j + p_e + p_k - d_k)$ and $V_{kej} = w_k(t + p_k - d_k) + w_j(t + p_k + p_e + p_j - d_j)$.

If k is found, set $h = k$ (job k is scheduled next) and go to Step 7; otherwise, set $h = j$ (job j is scheduled next) and go to Step 7.

Step 6. Search for the first job l that follows job j in L for which $t + p_l > d_l$ (the job is late), $t + p_l \leq s_e$ (the job can be scheduled before job e) and $V_{lej} < V_{ejl}$, where $V_{lej} = w_l(t + p_l - d_l) + w_j(t + p_l + p_e + p_j - d_j)$ and $V_{ejl} = w_j(t + p_e + p_j - d_j) + w_l(t + p_e + p_j + p_l - d_l)$. If l is found, set $h = l$ (job l is scheduled next) and go Step 7; otherwise (job e is scheduled next), go to Step 8.

Step 7 Add job h to the partial sequence, delete h from L, set $t = t + p_h$ and go to Step 4.

Step 8. Add the early job e to the partial sequence, delete e from E, and set $t = t + p_e$. If E is not empty, set e to be the first job in E and go to Step 4. Otherwise, add to the partial sequence in SWPT order any remaining jobs in L.

Step 9. The resulting sequence is taken as the initial solution for a descent algorithm which uses the adjacent interchange neighbourhood. During this search, a neighbour is accepted if it decreases the objective value, or if it leaves the value unaltered and the two interchanged jobs become sequenced in EDD order.

Example. To illustrate the mechanics of the decoding heuristic, consider the following 5-job problem. The processing times are 10, 7, 9, 5, 6; the job weights 10, 6, 5, 3, 3; and the due dates 22, 30, 8, 17, 0. For the chromosome *01010*, jobs 2 and 4 are early, and jobs 1, 3 and 5 are late.

In Step 1, $E = (4,2)$ and $L = (1,3,5)$. Since E is a feasible set of early jobs, sets E and L are unchanged in Step 2. In Step 3, after observing that no reordering of L is necessary, we compute latest possible start times $s_2 = 23$ and $s_4 = 12$, compute $R = 2.62$, and set $t = 0$ and $e = 4$. In Step 4, $i = 1$, but job 1 cannot be scheduled before job 4 to start at time zero because it would not be late. Job 3, however, is late if scheduled to start at time zero, so $j = 3$. In Step 5, for the candidate job $k = 5$, we compute the values $V_{jek} = 65$ and $V_{kej} = 78$ based on the partial sequences $(3,4,5)$ and $(5,4,3)$ to test condition (b). Thus, k cannot be found, so Step 7 forms the partial containing job 3 and sets $t = 9$ and $L = (1,5)$. After returning to Step 4 and setting $i = 1$ and $j = 5$, the heuristic proceeds to Step 6 to search to find a job of L to be scheduled next so that it completes not later than time $s_4 = 12$. Since no such job can be found, Step 8 schedules job 4 next to give the partial sequence $(3,4)$, sets $t = 14$, $E = (2)$ and $e = 2$.

On returning to Step 4, $i = 1$, and we choose $j = 1$. Since $t + p_j = 24 > 23 = s_2$, job 1 cannot be scheduled before job 2 without making it late. In Step 6, the candidate job $l = 5$ fails because the weighted tardiness of the partial sequence (which starts at time 14) $(5,2,1)$ is $V_{lej} = 210$ which exceeds $V_{ejl} = 201$ that corresponds to $(2,1,5)$. Thus, l cannot be found, and we proceed to Step 8 where job 2 is scheduled to create the partial sequence $(3,4,2)$ and E becomes empty. Step 8 also appends the jobs of L to give the complete sequence $(3,4,2,1,5)$.

Lastly, Step 9 reduces the total weighted tardiness from 206 to 142 by interchanging jobs 2 and 1 to give the sequence $(3,4,1,2,5)$.

3.4 Algorithm

The general structure of the genetic algorithm can be summarized as follows.

Genetic algorithm

Step 1. Create an initial population and set it as the current population. Evaluate the current population.

Step 2. Generate a new population from the current population using the genetic operators reproduction, crossover and mutation.

Step 3. Evaluate the new population. If an aging condition is satisfied, then stop. Set the current population to be the new population. If the maximum number of generations has not been executed, then go to Step 2.

During *evaluation*, the total weighted tardiness is computed for each chromosome of the population using the decoding heuristic described in Section 3.3. Let $V_1, \ldots, V_M$ denote these values. Next, $V_1, \ldots, V_M$ are mapped to fitness values $f_1, \ldots, f_M$. This mapping is necessary because a genetic algorithm always works with a maximization problem: we use $f_k = V_{\max} - V_k$ for $k = 1, \ldots, M$, where $V_{\max} = \max_{k=1,\ldots,M}\{V_k\}$. Sometimes the minimum and maximum fitness values are identical (all chromosomes have the same total weighted tardiness). In that case, the algorithm is terminated. Termination also occurs if a chromosome is found which yields a sequence with total weighted tardiness equal to zero.

The *reproduction* operator is an artificial version of natural selection. Chromosomes with a larger fitness value have a higher probability of surviving to be placed in a *mating pool* since the probability of selection is proportional to the fitness value. Instead of using raw fitness values $f_1, \ldots, f_M$ in the creation of the mating pool, they are replaced by scaled fitness values $F_1, \ldots, F_M$. This scaling prevents premature convergence and, in the latter stages of the algorithm, avoids random walks amongst the mediocre solutions. We use linear scaling, which is defined by

$$F_k = af_k + b \qquad k = 1, \ldots, M, \tag{1}$$

where a and b are non-negative constants. We compute a and b from two scaling relationships. The first dictates that average fitness remains unaltered by scaling, so that

$$\sum_{k=1}^{M} F_k = \sum_{k=1}^{M} f_k. \tag{2}$$

The second relationship specifies that the expected number of copies of the best population member to appear in the mating pool is some given quantity C, which yields

$$\max_{k=1,\ldots,M}\{F_k\} = C \sum_{k=1}^{M} F_k/M. \tag{3}$$

Because of (3), there is a possibility that this scaling could produce negative fitness values. If the constants a and b that are derived from (1), (2) and (3) yield a negative

minimum fitness value, then (3) is replaced by an alternative relationship which specifies that the the minimum scaled fitness is zero. Thus,

$$\min_{k=1,\ldots,M}\{F_k\} = 0. \tag{4}$$

Therefore, a and b are defined by (1), (2) and (3) unless, upon substitution in (1), one or more negative scaled fitness values are obtained, in which case (1), (2) and (4) are used to determine a and b.

Based on the scaled fitness values, chromosomes are selected for the mating pool using deterministic sampling. Since the selection probability of chromosome k ($k = 1,\ldots,M$) is $F_k/\sum_{l=1}^{M} F_l$ and the mating pool is of size M, the expected number of copies in the mating pool is $e_k = MF_k/\sum_{l=1}^{M} F_l$. Firstly, $\lfloor e_k \rfloor$ copies of each chromosome k are placed in the mating pool, and then the population is sorted in non-increasing order of the fractional parts $e_k - \lfloor e_k \rfloor$. The remainder of the chromosomes needed to fill the mating pool are drawn from the start of this sorted list.

In *crossover*, the chromosomes in the mating pool are mated at random in pairs. Instead of applying the usual one- or two-point crossover to the two chromosomes, we use a scheme which interchanges several small sections of the strings, where the sections are randomly selected. For example, suppose that this crossover operator interchanges sections of length two, and is applied to the two chromosomes 1011 0101 and 0010 1001. If the two sections starting at the third and sixth element are interchanged, we obtain the resulting new chromosomes 1010 0001 and 0011 1101. The number of sections to be interchanged and the lengths of the sections are parameters. After each crossover, the two old chromosomes are discarded.

The *mutation* operator prevents loss of diversity in the population by randomly altering elements in the chromosomes. The number of elements that are changed depends upon a mutation probability. Firstly, we computed the expected total number of elements to be mutated (total number of elements in the population multiplied by the mutation probability). This expected number of elements is selected at random in the population and each is mutated by setting 0 to 1 and vice versa.

The motivation for *aging* is to avoid superfluous iterations at the end of the algorithm. When the best solution in a population is not better than that of the previous population for a number of iterations, the procedure is terminated.

4 Computational experience

4.1 Test problems

Test problems were generated as follows. For each job i, an integer processing time p_i was generated from the uniform distribution [1,100] and, for weighted tardiness problems, an integer weight was generated from the uniform distribution [1,10]. Problem 'hardness' is likely to depend on the relative range of due dates (RDD) and on the average tardiness factor (TF). Having computed $P = \sum_{i=1}^{n} p_i$ and selected values of RDD and TF from the set {0.2, 0.4, 0.6, 0.8, 1.0}, an integer due date d_i from the uniform distribution $[P(1-\mathrm{TF}-\mathrm{RDD}/2), P(1-\mathrm{TF}+\mathrm{RDD}/2)]$ was generated for each

job i. Five problems were generated for each of the 25 pairs of values of RDD and TF, yielding 125 problems for each value of n. For the total tardiness problem, instances with $n = 50$ and $n = 100$ were generated, whereas we have $n = 20$ and $n = 40$ for the total weighted tardiness problem. The algorithms of Potts and Van Wassenhove [10,11] were used to obtain an optimal solution value to each problem. For our computational tests, all algorithms were coded in ANSI-C and run on a HP 9000/825 computer.

4.2 Total tardiness problem

In this section, five heuristics for the total tardiness problem (TTP) are compared. A precise description of the first three (WI, DES0, SAPW) can be found in Potts and Van Wassenhove [13]. For the two new methods, details of tabu search (TS) are given in Section 2, whereas the genetic algorithm (GA) is described in Section 3.

WI: the Wilkerson-Irwin heuristic [14] is specifically designed for the TTP and outperforms 'quick and dirty' constructive methods.

DES0: a descent method based on the interchange neighbourhood in which a neighbour is accepted if it leaves the objective function unchanged. The number of iterations is $n(n-1)L/2$, where $L = 10$, i.e. 12250 iterations for $n = 50$ and 49500 for $n = 100$.

SAPW: a simulated annealing algorithm that incorporates descent. The descent method, which uses the adjacent interchange neighbourhood, is applied to find a local optimum whenever an increase of the objective function value is followed immediately by a decrease. The acceptance probability of a 1% increase in the current best solution value is controlled with a convex function; the initial acceptance probability is 0.99. The number of iterations is $n(n-1)L/2$, as in DES0.

TS: the tabu search procedure described in Section 2, which incorporates descent. A tabu list length of 7 is used. The number of iterations is nL, where $L = 10$, although the procedure is also terminated if no overall improvement in the objective function value is observed for $nL/2$ iterations.

GA: the genetic algorithm described in Section 3 with a population size $M = 50$. The maximum number of iterations is $n/2$, but due to aging (no improvement in the best solution value for $n/4$ iterations) the procedure may terminate before this iteration limit is reached. The crossover scheme interchanges $n/5$ two-element sections, and the mutation probability is 0.001. Linear fitness scaling with a parameter $C = 2.0$ is performed, and the best chromosome found so far is preserved in the next generation (simple elitism).

The computation time for the three neighbourhood search methods is determined by the run length parameter L.

Table 1 summarizes our results for the TTP when the heuristic methods described above are applied to our data set of 125 problems with 50 and 100 jobs respectively. The table gives the average relative percentage deviation (ARPD) of the heuristic solution

value from the optimal value, the maximum relative percentage deviation (MRPD), the number of times (out of 125) that an optimal solution is found (NO), and average computation time (ACT) in seconds on a HP 9000/825.

Table 1. Results for the total tardiness problem

	$n = 50$				$n = 100$			
	ARPD	MRPD	NO	ACT	ARPD	MRPD	NO	ACT
WI	1.26	11.39	67	0.06	1.28	12.57	54	2.45
DES0	0.06	1.47	101	1.10	0.27	17.12	79	8.22
SAPW	0.19	2.49	70	2.09	0.74	6.81	40	12.48
TS	0.08	1.47	76	1.90	0.07	1.83	60	13.51
GA	0.01	0.36	111	3.76	0.04	1.23	87	25.05

Although heuristic WI generates substantial numbers of optimal solutions, they mostly correspond to relatively easy problems in which the average tardiness factor (TF) is equal to 0.2 or 1.0. Also, the maximum relative deviation is quite large for WI, which indicates that the method is not robust. Amongst the three neighbourhood search methods, DES0 requires the least computation time and it generates the best results for $n = 50$. It also finds the optimal solution more frequently than SAPW and TS, and the average relative percentage deviation is quite small for $n = 100$. For most problems, the special techniques in SAPW and TS do not seem necessary or particularly effective in escaping from a local optimum. Allowing 'zero interchanges' in descent seems to be sufficient for most problems, although the maximum relative deviation of 17.12% indicates that the search can become trapped at a local optimum. For both $n = 50$ and $n = 100$, TS outperforms SAPW and requires approximately the same computation time. The difference in performance between these two methods is more pronounced when $n = 100$. The best results are obtained with GA, but this method also requires the most computation time. Worthy of note is the small value for the maximum relative deviation (1.23% when $n = 100$). Thus, for all of the test problems, a reasonably good solution can be computed using GA, thereby illustrating the robustness of our genetic algorithm.

4.3 Total weighted tardiness problem

For the total weighted tardiness problem (TWTP), we compare the three heuristics (AU, DES0, SAPW) that are described in detail by Potts and Van Wassenhove [13] with the tabu search method (TS) and genetic algorithm (GA) which are developed in this paper. Brief descriptions of DES0 and SAPW are given in Section 4.2, details of TS are given in Sections 2 and 4.2, and GA is described in Sections 3 and 4.2.

AU: the Apparent Urgency rule of Morton *et al.* [8] which performs quite well relative to other simple constructive heuristics.

DES0: the descent method, with $L = 10$.

SAPW: the simulated annealing algorithm, with $L = 10$.

TS: the tabu search procedure, with $L = 10$.

GA: the genetic algorithm with a population size $M = n$, the maximum number of iterations is $n/4$, and termination due to aging occurs if there is no improvement to the best solution value for $n/8$ iterations.

Table 2 presents results for our 20-job and 40-job test problems. The neighbourhood search methods DES0, SAPW and TS perform substantially better than the 'quick and dirty' constructive heuristic AU. Amongst these three methods, DES0 requires the least computation time. Performing more iterations with DES0 to obtain a comparable computation time with SAPW and TS does not significantly improve the solution quality. Augmenting the value of the run length parameter L in DES0, for example from 10 to 30, decreases the average relative deviation from 0.63% to 0.61% when $n = 40$, but produces no change when $n = 20$. In neither case does the maximum relative deviation change. For the TWTP, allowing 'zero interchanges' in descent is therefore not sufficient to allow the search to escape from a local optimum.

Table 2. Results for the total weighted tardiness problem

	$n = 20$				$n = 40$			
	ARPD	MRPD	NO	ACT	ARPD	MRPD	NO	ACT
AU	16.24	220.85	23	0.02	12.78	169.18	24	0.04
DES0	0.29	9.47	101	0.10	0.63	19.27	76	0.60
SAPW	0.05	3.90	120	0.19	0.86	41.65	79	1.17
TS	0.07	2.20	111	0.22	0.28	6.70	84	1.05
GA	0.07	1.68	111	0.12	0.06	1.21	87	1.00

For $n = 20$, we observe from Table 2 that SAPW generates the most optimal solutions and it has the smallest average relative deviation, although its maximum relative deviation is large in comparison with that of TS and GA. However, for $n = 40$, TS dominates SAPW, but is outperformed by the genetic algorithm GA. Also, GA exhibits the smallest average and maximum relative deviations, and it generates the largest number of optimal solutions. Note also that GA requires less computation time than SAPW and TS.

The four local search heuristics were also tested on a set of 125 total weighted tardiness problems with $n = 50$. Because the optimal solutions are not available, relative deviations for the heuristics are evaluated with respect to the best solution found. The heuristics were applied for different numbers of iterations to observe convergence. Specifically, we consider run length parameter values $L = 10$, $L = 30$ and $L = 50$ for DES0, SAPW and TS. Also, the termination criteria in GA are modified so that the search stops due to aging when no improvement occurs during $L/2$ iterations, but otherwise L populations are generated. Results are given in Table 3.

Table 3. Results for 50-job total weighted tardiness problems												
	ARPD			MRPD			NO			ACT		
L	10	30	50	10	30	50	10	30	50	10	30	50
DES0	0.70	0.59	0.57	11.20	11.20	11.20	56	61	62	1.13	3.28	5.42
SAPW	0.65	0.36	0.30	11.20	11.20	11.20	61	79	79	2.18	5.82	9.07
TS	0.41	0.22	0.16	11.97	5.67	5.05	67	84	88	2.07	5.94	9.38
GA	0.18	0.05	0.04	5.30	1.30	1.16	71	95	100	1.94	5.08	7.92

Table 3 confirms our intuition. There is no substantial improvement in the performance of DES0 when the number of iterations increases because it suffers from the inability to escape from a local optimum. However, SAPW generates better quality solutions with larger computation times, although the maximum deviation remains large which indicates that for some problems the interchange neighbourhood does not permit an adequate exploration of the solution space. This hypothesis is confirmed by the performance of TS, although the results are slightly better than for SAPW. The best overall performance is obtained with GA.

Table 4 shows the distribution of the deviations of the heuristic solution value from the value of the best known solution for the AU rule and the four local search heuristics for $n = 50$ and run length parameter value $L = 30$.

Table 4. Distribution of relative deviations for $n = 50$ and $L = 30$							
		Frequency of relative deviations (%)					
	ARPD	$[0,0]$	$(0,1]$	$(1,5]$	$(5,10]$	$(10,\infty]$	MRPD
AU	11.00	22	32	25	7	39	125.10
DES0	0.59	61	47	14	1	2	11.20
SAPW	0.36	79	39	3	3	1	11.20
TS	0.22	84	34	5	2	0	5.67
GA	0.05	95	29	1	0	0	1.30

For GA, only one problem out of 125 results in a solution value with a relative deviation of more than 1% from the best known value. This confirms the robustness of GA. For each of SAPW and TS, however, there are seven problems with relative deviations of more than 1%, and even more for DES0. Although AU is quite effective for some instances, there are 39 problems where a deviation of more than 10% from the best known solution value is observed.

5 Concluding remarks

This paper proposes two new local search heuristics for the problem of scheduling a single machine to minimize the total weighted tardiness. The first is a tabu search method (TS) which uses the interchange neighbourhood. The other new method is a genetic algorithm (GA), special features of which are the simple binary encoding of the solution and an effective decoding heuristic.

Extensive computational tests for the total tardiness problem (TTP) and the total weighted tardiness problem (TWTP) are used to compare TS and GA with other local search heuristics and with a simple constructive method. Our results for the TTP suggest that the Wilkerson-Irwin heuristic or any simple constructive method should be used with caution. With a relatively small limit on computation time, descent with 'zero interchanges' (DES0) outperforms simulated annealing (SAPW) and tabu search (TS) for the TTP. It can, however, encounter difficulty with larger problem sizes and sometimes generate solutions with a large deviation from the optimal solution value. When more computation time is used, DES0 is not always able to escape from a local optimum, whereas in most cases SAPW and TS allow this possibility. The tabu search method TS dominates SAPW for the TTP, and TS is more robust than DES0 although it finds an optimal solution less frequently. Moreover, TS is superior to SAPW for the TWTP for larger problem sizes ($n \geq 40$).

Our genetic algorithm (GA) generates solutions of very good quality. For the TTP, GA is a viable alternative to other heuristic methods, especially in view of its small maximum relative deviations, although it requires more computation time. For the TWTP, none of the other heuristics presented in this paper can compete with GA when comparable computation times are used.

In conclusion, our new genetic algorithm is superior to other heuristic methods for the total (weighted) tardiness problem. A major contributory factor to its success is the heuristic method which decodes the representation of a solution to produce a sequence of jobs. The new tabu search method is also superior to other neighbourhood search heuristics.

6 References

1. T.S. Abdul-Razaq, C.N. Potts and L.N. Van Wassenhove, 1990. A Survey of Algorithms for the Single Machine Total Weighted Tardiness Scheduling Problem, *Discrete Applied Mathematics 26,* 235–253.

2. J. Du and J.Y.-T. Leung, 1990. Minimizing Total Tardiness on One Processor is NP-hard, *Mathematics of Operations Research 3,* 483–495.

3. F. Glover, 1989. Tabu Search—Part I, *ORSA Journal on Computing 1,* 190–206.

4. F. Glover, 1990. Tabu Search: A Tutorial, *Interfaces 20,* 74–94.

5. D.E. Goldberg, 1989. *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley, Reading, MA.

6. E.L. Lawler, 1977. A "Pseudopolynomial" Algorithm for Sequencing Jobs to Minimize Total Tardiness, *Annals of Discrete Mathematics 1,* 331–342.

7. J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, 1977. Complexity of Machine Scheduling Problems, *Annals of Discrete Mathematics 1,* 343–362.

8. T.E. Morton, R.M. Rachamadugu and A. Vepsalainen, 1984. Accurate Myopic Heuristics for Tardiness Scheduling, GSIA Working Paper No. 36-83-84, Carnegie-Mellon University, PA.

9. M. Pirlot, 1992. General local search heuristics in Combinatorial Optimization: A Tutorial, *Belgian Journal of Operations Research, Statistics and Computer Science 32,* 8–67.

10. C.N. Potts and L.N. Van Wassenhove, 1982. A Decomposition Algorithm for the Single Machine Total Tardiness Problem, *Operations Research Letters 1,* 177–181.

11. C.N. Potts and L.N. Van Wassenhove, 1985. A Branch and Bound Algorithm for the Total Weighted Tardiness Problem, *Operations Research 33,* 363–377.

12. C.N. Potts and L.N. Van Wassenhove, 1987. Dynamic Programming and Decomposition Approaches for the Single Machine Total Tardiness Problem, *European Journal of Operations Research 32,* 405–414.

13. C.N. Potts and L.N. Van Wassenhove, 1991. Single Machine Tardiness Sequencing Heuristics, *IIE Transactions 23,* 346–354.

14. L.J. Wilkerson and J.D. Irwin, 1971. An Improved Algorithm for Scheduling Independent Tasks, *AIIE Transactions 3,* 239–245.