"LOCAL SEARCH HEURISTICS FOR SINGLE MACHINE SCHEDULING WITH BATCHING TO MINIMIZE TOTAL WEIGHTED COMPLETION TIME"

by

H.A.J. CRAUWELS*
C.N. POTTS**
and
L.N. VAN WASSENHOVE†

95/28/TM

* KIHDN, Sint-Katelijne-Waver, Belgium.

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

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

Printed at INSEAD, Fontainebleau, France

Local Search Heuristics for Single Machine Scheduling with Batching to Minimize Total Weighted Completion Time

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

January, 1995

Abstract

Local search heuristics are developed for a problem of scheduling jobs on a single machine. Jobs are partitioned into families, and a set-up time is necessary when there is a switch in processing jobs from one family to jobs of another family. The objective is to minimize the total weighted completion time. Four alternative neighbourhood search methods are developed: multi-start descent, simulated annealing, threshold accepting and tabu search. The performance of these heuristics is evaluated on a large set of test problems, and the results are also compared with those obtained by a genetic algorithm. The tabu search method generates high quality schedules relative to the other methods at modest computational expense.

Keywords: scheduling, single machine, batches, set-up time, local search heuristics, descent, simulated annealing, threshold accepting, tabu search.

1 Introduction

In many manufacturing industries, families of jobs are to be processed on one or more machines. On each machine, a set-up time is required at the start of each batch, where a batch is a largest set of contiguously scheduled jobs from the same family. Although production efficiency is maximized by selecting large batches, this strategy may cause delays to high priority jobs. Thus, batching and scheduling decisions must be integrated in order to achieve a satisfactory compromise between these conflicts.

In this paper, we consider a single machine scheduling problem in which N jobs are partitioned into F families. A sequence independent set-up is

required whenever there is a switch from processing a job in one family to a job in another family. We assume that all the jobs are available at time zero, and that each job has a given processing time and an associated positive weight. The objective is to find a schedule which minimizes the total weighted completion time of the jobs.

When all set-up times are zero, the problem is solved by the shortest weighted processing time (SWPT) rule of Smith [18]. Monma and Potts [13] consider a variety of single machine scheduling problems with batch set-up times under the assumption that sequence dependent batch set-up times satisfy the 'triangle inequality': the set-up time associated with a changeover from family f to h is assumed to take no longer than that for changeover from family f to g followed by a changeover from family g to h (our sequence independent set-up times clearly satisfy this triangle inequality). They show that jobs within each family are sequenced in SWPT order in an optimal schedule. Using this property, they derive a dynamic programming algorithm for minimizing the total weighted completion time under this set-up time structure: the time requirement is $O(F^2N^{F^2+2F})$, which reduces to $O(F^2N^{2F})$ time when set-up times are sequence independent. For the case of unit weights, a more efficient dynamic programming algorithm of Ahn and Hyun [1] solves the problem in $O(F^2N^F)$ time, for both sequence dependent and independent set-up times. For arbitrary weights and both sequence dependent and independent set-up times, Potts [15] derives an $O(N^3)$ dynamic programming algorithm for $F = 2$, and Ghosh [6] proposes an $O(F^2N^F)$ dynamic programming algorithm. Although these various dynamic programming algorithms require polynomial time for a fixed number of families, they are largely of theoretical interest unless the number of families is very small.

Gupta [8] proposes a SPT-based heuristic method for the problem with sequence dependent set-up times and unit weights. Essentially, it is a single pass job sorting routine which uses the SPT rule (where the processing time incorporates the set-up time) in a 'greedy' fashion. It has modest computational requirements, but often suffers the disadvantage of generating poor quality schedules. Nevertheless, it can be usefully employed to construct a starting solution for neighbourhood search heuristics.

Another heuristic is developed by Ahn and Hyun [1]. It is an descent algorithm based on a large neighbourhood. They use the SWPT ordering of jobs within each family to reduce the size of the neighbourhood. Further details of the neighbourhood are given in Section 3. Mason [11] proposes a genetic algorithm in which a batch-based representation of solutions is used. More precisely, a binary string is used to represent a solution, where each position in the string corresponds to a job, and the corresponding string element indicates whether or not the job starts a new batch. Using SWPT properties, which are described in the next section, for jobs within a family and for batches, it is straightforward to construct a sequence of jobs from this representation.

Mason and Anderson [12] derive a variety of powerful dominance criteria, which they use in a branch and bound algorithm. Relatively weak but quickly computed lower bounds are employed. Crauwels et al. [2] develop a tighter lower bounding scheme using a Lagrangean relaxation of machine capacity constraints. Computational results show that the branch and bound algorithm with the Lagrangean lower bounding scheme can solve instances with up to 50 jobs, whereas computation times become prohibitive when there are more than 30 jobs in the algorithm of Mason and Anderson.

The complexity of this batching problem for an arbitrary number of families is still open (see Potts and Van Wassenhove [17]). Nevertheless, the various attempts to solve the problem using branch

and bound indicate its challenging nature. Therefore, it is worthwhile to investigate some heuristic techniques. Over the past decade, several types of local search methods have been developed: these include simulated annealing, threshold accepting, tabu search and genetic algorithms. An introductory overview of these methods and some applications are given by Pirlot [14].

In this paper, we investigate the performance of several local search methods. Various computational tests are performed to select between different versions of the same algorithm. Also, the results of comparative tests are used to compare the performance of these methods with that of the genetic algorithm of Mason [11].

In Section 2, we give a formal statement of our problem, and describe some dominance criteria derived by Mason and Anderson. Sections 3, 4, 5 and 6 describe the different heuristic methods: descent, simulated annealing, threshold accepting, and tabu search. Section 7 reports on computational experience, and some concluding remarks are given in Section 8.

2 Problem formulation and dominance criteria

To state our problem of scheduling with batch set-up times more precisely, we are given N jobs that are divided into F families. Each family f, for $f = 1, \ldots, F$, contains n_f jobs which are labeled $(1, f), \ldots, (n_f, f)$. All jobs are available for processing at time zero, and are to be scheduled on a single machine. Let p_{if} denote the processing time of job (i, f), and w_{if} is its associated positive weight.

A sequence independent set-up time $s_f \geq 0$ is required whenever a job in family f is processed immediately after a job in a different family. Also, an initial set-up time s_f is incurred if a job from family f is the first to be processed.

Any sequence of jobs defines a schedule. It is convenient to regard a sequence S as a series of batches, where a batch is a maximal consecutive subsequence of jobs in S from the same family (and any preceding or succeeding job is from a different family). If B_b is the b'th batch in S, then

$$S = (B_1, \ldots, B_\beta),$$

where β is the total number of batches. Each batch B_b can be viewed as a single composite job with processing time P_b and weight W_b. If batch B_b contains jobs $j, \ldots, l$ from some family f, then P_b and W_b are defined by

$$P_b = s_f + \sum_{i=j}^{l} p_{if}, \qquad W_b = \sum_{i=j}^{l} w_{if}.$$

We also define the weighted processing time of batch B_b as $\mathrm{WPT}(B_b) = P_b/W_b$.

It is useful to present some results about the ordering of jobs within a family and the ordering of batches. There exists an optimal schedule with the following properties.

SWPT property for jobs within a family: job (i, f) precedes another job (j, f) if $p_{if}/w_{if} \leq p_{jf}/w_{jf}$ [13].

SWPT property for batches: batch B_b precedes batch B_c if $\mathrm{WPT}(B_b) \leq \mathrm{WPT}(B_c)$ [12].

Any sequence that satisfies both the SWPT property for jobs within a family and the SWPT property for batches can be used as an initial solution for our local search heuristics. For example, by assigning all jobs in each family to a single batch, we can use these properties to define a sequence.

We propose a more general procedure for determining an initial solution. This method, which allows some families to be split into two or more batches, uses the same greedy approach as that in the heuristic of Gupta [8] for an analogous problem. Jobs within each family are scheduled in SWPT order. Initial partial schedules are constructed using an earliest weighted

completion time rule: the job which is appended to the current partial schedule is chosen so that its weighted completion time is as small as possible. The batches in the resulting sequence are then reordered so that they satisfy the SWPT property for batches.

3 Neighbourhood search

A neighbourhood search method requires an initial solution, which can be constructed by some heuristic rule or it can be chosen at random. In our algorithms, we use the adapted version of Gupta's heuristic to create the first initial solution. When several independent runs from different starting solutions are required, the other initial solutions are randomly generated. Having obtained a solution, one of its neighbours is generated by some suitable mechanism, and some acceptance rule is used to decide on whether it should replace the current solution. This process is repeated until some termination criterion is satisfied.

For our problem, various neighbourhoods are possible. For example, Ahn and Hyun [1] suggest that a neighbour can be constructed by shifting a job or a sub-batch forward and backward, while maintaining the SWPT property for jobs within each family. Consider a sequence

$$S = (B_1, \ldots, B_a, \ldots, B_b, \ldots, B_\beta),$$

which comprises β batches. Let $B_a = j_a, f), \ldots, (l_a, f))$, and $B_b = j_b, g), \ldots, (l_b, g))$. A neighbour is created by the forward or backward shift of some sub-batch. For example, suppose that none of the batches $B_a, \ldots, B_{b-1}$ are from family g. Then the sequence

$$\begin{aligned}(B_1, \ldots, B_{a-1}, (j_a, f), \ldots, (k_a, f), (j_b, g), \ldots, (k_b, g),\\ (k_a + 1, f), \ldots, (l_a, f), B_{a+1}, \ldots, B_{b-1}, (k_b + 1, g), \ldots, (l_b, g), B_{b+1}, \ldots, B_\beta)\end{aligned}$$

is a neighbour that is created from S by the forward shift of the sub-batch $j_b, g), \ldots, (k_b, g))$, where $k_b \in \{j_b, \ldots, l_b\}$ and $k_a \in \{j_a - 1, \ldots, l_a\}$. A backward shift of the sub-batch $(k_b, g), \ldots, (l_b, g)$ is defined analogously. We refer to the set of all forward and backward shift neighbours as the *shift sub-batch* neighbourhood.

Included in the shift sub-batch neighbourhood is a special case in which only one job is shifted. For a forward shift of a single job, we select the first job of a batch and insert it immediately before the start of some earlier batch. For example, by moving job (j_b, g) immediately before batch B_a in S, where none of the batches $B_a, \ldots, B_{b-1}$ are from family g, we obtain the sequence

$$(B_1, \ldots, B_{a-1}, (j_b, g), B_a, \ldots, B_{b-1}, (j_b + 1, g), \ldots, (l_b, g), B_{b+1}, \ldots, B_\beta).$$

Similarly, the backward shift of a single job moves the last job of a batch to a position immediately after the end of a later batch. This *shift job* neighbourhood is smaller than the more general shift sub-batch neighbourhood.

Note that a shift sub-batch or a shift job neighbour does not necessarily satisfy the SWPT property for batches. Although we could remedy this defect by reordering the batches for every neighbour, the computational expense associated with such a reordering makes this approach unattractive. Instead, we executed a reordering for selected neighbours only.

The acceptance rule in a neighbourhood search method is usually dependent on a comparison of objective function values for the current solution and for its neighbour. If the former is larger (for a minimization problem), then we refer to the neighbour as an *improving move*; if the latter is larger, it is a *deteriorating move*; if both are the same, then it is a *neutral move*.

In a *descent* method, only improving moves are allowed. A potential move is rejected if it is deteriorating or neutral. When no further improvement can be found, the procedure stops. Although the resulting solution is a local optimum, it is not necessarily a global optimum. A classical remedy for this drawback is to perform multiple runs of the procedure starting from different initial solutions and to take the best sequence as final solution. Such an approach is called *multi-start descent*. Our implementation of multi-start descent is consistent with the algorithm of Ahn and Hyun [1] in that there is no reordering of the batches to enforce the SWPT property for batches.

Another possibility is to adopt an acceptance rule which allows non-improving moves. Consequently, the procedure can escape from a local optimum and continue its search for a global optimum. Simulated annealing, threshold accepting and tabu search are examples of procedures which allow deteriorating and neutral moves during the search.

4 Simulated annealing

In a simulated annealing procedure, improving and neutral moves are accepted, while deteriorating moves are accepted according to a given probabilistic acceptance function. Following the lead of Kirkpatrick et al. [10] who suggest simulated annealing as a solution method for the traveling salesman problem, many papers are devoted to both the theoretical and computational aspects of this method. A review is provided by Eglese [5].

The most common form of the acceptance function is $p(\delta) = \exp(-\delta/t)$, where $p(\delta)$ is the probability of accepting a move which results in an increase of δ (where $\delta > 0$) in the objective function value. The parameter t is known as the *temperature*, and its value changes at suitably chosen intervals.

Let L denote the number of different temperatures that are considered. In our implementation of simulated annealing, each temperature t_i, for $i = 1, \ldots, L$, is derived from an acceptance probability K_i, as described by Potts and Van Wassenhove [16]. More precisely, K_i is equal to the probability of accepting a one percent increase in Z, where Z is the value of the best solution known thus far. Thus, $K_i = \exp(-0.01Z/t_i)$, from which we obtain $t_i = -0.01Z/\ln K_i$ for $i = 1, \ldots, L$.

For each of the L acceptance probability levels, the complete neighbourhood is searched systematically. Whenever a solution is found that improves on the best solution found thus far, the batches are reordered so that they satisfy the SWPT property for batches. This reordering allows the possibility of a further improvement in the best objective function value. We test the performance of simulated annealing with both the shift sub-batch and the shift job neighbourhoods that are defined in Section 3. Also, two schemes for setting acceptance probabilities are compared. The termination criterion is defined by fixing a priori the number of levels L.

We now describe a scheme in which the acceptance probabilities decrease in a *linear* pattern. The initial and final acceptance probabilities K_1 and K_L are specified, and we set $K_i =$

Towards the beginning of the simulated annealing algorithm, the acceptance probability is high, thus allowing the acceptance of many moves that increase the objective function value. With a large neighbourhood, many solutions are searched at the same temperature, and too many solutions with increased objective function values are accepted. The search process then resembles a random walk amongst neighbouring solutions. This phenomenon tends to occur even when the initial acceptance probability is relatively small. Towards the end of the simulated annealing algorithm, the acceptance probability is low and effectively a pure descent

procedure is executed, with the result that the search may become trapped in a local optimum. Therefore, when the number of acceptance probability levels is increased, much of the extra computation time tends to be wasted.

We propose three modifications to the linear acceptance probability which attempt to overcome these drawbacks. First, to increase the chance for the search to evolve to a local optimum, before proceeding to a new acceptance probability level, we apply descent (where the

acceptance probability is zero thus preventing moves that increase the objective function value). Second, we propose an acceptance probability that is periodic in nature. Thus, the acceptance probability always returns to a high level which allows the search to escape from a local minimum. Third, we use a cutoff to diminish the random walk effect, as suggested by Johnson et al. [9]: as soon as a certain number of moves at a level have been accepted, the temperature is decreased. In our implementation, the temperature is halved each time an objective function value increase is accepted. In this way, all elements of the neighbourhood can be explored within one acceptance probability level without moving too far away from interesting solution subspaces.

We now provide specific details about the acceptance probabilities and temperatures in our *periodic* scheme. Having selected K_1' and K_L', where K_1' and K_L' are parameters that control the range acceptance probabilities, we set

$$K_{10h+4i+j} = \begin{cases} K_L' + (1 + R_{2h})(K_1' - K_L')x_{4i+j}/2 & \text{for } h = 0,1,\ldots,\lfloor L/10 \rfloor,\ i = 0, \quad j = 2,4, \\ K_L' + (1 + R_{2h+1})(K_1' - K_L')x_{4i+j}/2 & \text{for } h = 0,1,\ldots,\lfloor L/10 \rfloor,\ i = 1, \quad j = 2,4,6, \\ 0 & \text{for } h = 0,1,\ldots,\lfloor L/10 \rfloor,\ i = 0,1,\ j = 1,3,5, \end{cases}$$

where R_{2h} and R_{2h+1} are random numbers from the uniform distribution $(0,1)$, and $x_2 = x_4 = 0.25$, $x_6 = x_{10} = 1.0$ and $x_8 = 0$. Note that there are ten distinct values of $4i + j$, since the two combinations $i = 0$, $j = 5$, and $i = 1$, $j = 1$ both yield $4i + j = 5$. When $K_{10h+4i+j} = 0$, we have $t_{10h+4i+j} = 0$ which corresponds to descent iterations, although neutral moves are accepted. Apart from the different random numbers R_{2h} and R_{2h+1}, the values of $K_{10h+4i+j}$ for even j are periodic, and the period is 5. Having computed the temperature $t_{10h+4i+j} = -0.01Z/\ln K_{10h+4i+j}$ at the start of such a level, we reset $t_{10h+4i+j} = t_{10h+4i+j}/2$ whenever a solution which increases the objective function value is accepted.

The use of a 'cooling' scheme which also heats up when deteriorating moves are no longer accepted, has already been suggested by Dowsland [3]. The decision on when to heat up again in that algorithm depends on tests indicating whether the search is trapped in a local optimum. In our method, however, phases of cooling down and heating up are more precisely fixed and are not correlated to solution-specific conditions.

In our implementation, we select $K_1 = 0.1$ and $K_L = 0.001$ when the linear acceptance probability is used, while $K_1' = 0.5$ and $K_L' = 0.001$ for the periodic acceptance probability. It is also possible to adopt a multi-start simulated annealing algorithm. If R is the number of different starting solutions, then the number of acceptance probability levels is given by $L = 2N/R$ when the shift sub-batch neighbourhood is used, and $L = 4N/R$ for the shift job neighbourhood. In the final version of simulated annealing that is used in the comparative tests in Section 7.4, a secondary termination test is used. If no overall improvement in the best objective function is observed for $N/5$ levels, then the algorithm proceeds to the next starting solution, or it terminates if a search from R starting solutions has already been performed.

5 Threshold accepting

Threshold accepting is a similar method to simulated annealing. Whereas simulated annealing uses a probabilistic acceptance rule for solutions that cause a deterioration in the objective function value, threshold accepting is deterministic. Threshold accepting is first introduced by Dueck and Scheuer [4], who claim that it yields better results than simulated annealing (SA).

In threshold accepting, a move is accepted provided that it does not increase the objective function by more than V, where V is a *threshold value*. The threshold value plays a similar role to the temperature in simulated annealing. Thus, the common practice is to select a decreasing sequence of threshold values, so that V is a relatively high value initially, and is fairly low during the final iterations.

Analogously to our simulated annealing algorithm, we use the same threshold value during one search of the complete neighbourhood: we refer to this search as a level. Also, the batches are reordered so that they satisfy the SWPT property for batches whenever a solution is found that improves on the best solution found thus far. Our periodic pattern of threshold values are defined in terms of parameters V_1'

and V_L' that control the range of values. Having chosen the number of levels L and selected values of V_1' and V_L', our threshold values are defined by

$$V_{10h+4i+j} = \begin{cases} V_L' + (1 + R_{2h})(V_1' - V_L')x_{4i+j}/2 & \text{for } h = 0, 1, \ldots, \lfloor L/10 \rfloor,\ i = 0, \quad j = 2, 4, \\ V_L' + (1 + R_{2h+1})(V_1' - V_L')x_{4i+j}/2 & \text{for } h = 0, 1, \ldots, \lfloor L/10 \rfloor,\ i = 1, \quad j = 2, 4, 6, \\ 0 & \text{for } h = 0, 1, \ldots, \lfloor L/10 \rfloor,\ i = 0, 1,\ j = 1, 3, 5, \end{cases}$$

where R_{2h} and R_{2h+1} are random numbers from the uniform distribution $(0, 1)$, and $x_2 = 0.75$, $x_4 = 0.25$, $x_6 = 1.0$, $x_8 = 0.5$ and $x_{10} = 0.0$. When $V_{10h+4i+j} = 0$, threshold accepting becomes a descent method, although neutral moves are accepted. The threshold values $V_{10h+4i+j}$ for even j have an underlying periodic structure.

We implement threshold accepting by setting $V_1' = 0.005Z_0$ and $V_L' = 0.0001Z_0$, where Z_0 is the objective function value of the initial solution. In a multi-start version of this method with R starting solutions, we set the number of different threshold values to be $L = 2N/R$.

6 Tabu search

Tabu search offers an alternative deterministic approach for escaping from a local optimum. The main features of the method are described by Glover [7]. Informally, the method initially resembles a descent procedure, but exploration continues after the first local optimum is found by choosing the best available non-improving move.

However, certain moves are forbidden or are *tabu* for a few iterations to prevent the method cycling by repeatedly searching the same sequence of solutions.

Conventionally, the complete neighbourhood is searched at each iteration to find the best possible non-tabu move. Thus, it is advantageous to choose a small neighbourhood. Therefore, we use the shift job neighbourhood in preference to the shift sub-batch neighbourhood. Also, in our implementation, as soon as the search reveals a non-tabu move that is improving, we execute this move without exploring the remaining neighbours. In this case, a limited reordering of batches which do not appear in SWPT order is undertaken. More precisely, batches which containing a single job are used to partition the sequence of batches into subsequences, and the batches of each subsequence are reordered so that they satisfy the SWPT property for batches.

(The reason for not allowing the batches containing a single job to be reordered is that the resulting sequence may conflict with the tabu list, which is descibed below.)

A tabu list is constructed, the purpose of which is to prevent the search from returning to a previously visited solution, but has the added advantage of helping to drive the search to another part of the solution space. There are several ways to characterize a tabu move: we propose two alternatives. In the first type of tabu list, certain job-position assignments are prohibited. More precisely, a move is tabu if, for the neighbour, some job is shifted to a position in the sequence, where this job-position pair is stored on the tabu list.

After a move is executed, the job that is shifted and its original position are stored in the tabu list. Thus, the tabu list prevents a shifted job from returning to its original position. For the special case that a job is shifted one position so that effectively two jobs are transposed, the job and original position of the other job are additionally stored. The second type of tabu list comprises a list of jobs, and a move is tabu if it corresponds to shifting one of the jobs in the list. After a move is executed, the job that is shifted is stored on the tabu list, or both jobs are stored if the move is effectively the transpose of adjacent jobs. We refer to the two types of lists as *position* and *job* lists, respectively. Clearly, more neighbours are forbidden for the job list. The tabu list length is fixed at the start of the algorithm. Whenever the list becomes full and a new entry is to be added, the oldest element is overwritten.

It is sometimes possible that a neighbour is tabu but has a better objective function value than all previously generated solutions. To prevent the occasional loss of good solutions, an aspiration criterion is incorporated. If the solution value of a tabu neighbour is better than that for all solutions generated thus far, then its tabu status is overridden.

Based on the results of initial experiments, a tabu list of length 25 is chosen for the position tabu list, and of length 7 for the job tabu list. We adopt a stopping rule that terminates the search from a starting solution after $2N$ iterations. For the comparative tests in Section 7.4, we use a secondary termination test. If no overall improvement in the best objective function is observed for $N/2$ iterations, then the algorithm either proceeds to the next starting solution or terminates according to the number of different starting solutions that have currently been considered.

7 Computational results

The heuristics were tested by coding them in ANSI-C and running them on a HP 9000/825 computer. Test problems with 30, 40 and 50 jobs, and with 4, 6, 8 and 10 families were generated as follows. For each combination of N and F, the jobs are uniformly distributed across families, so that each family contains $\lfloor N/F \rfloor$ or $\lceil N/F \rceil$ jobs. Processing times and weights are randomly generated integers from the uniform distribution defined on $[1, 10]$. Since the size of set-up times relative to processing times may affect the relative performance of the different heuristics, we generated problems with small (S), medium (M) and large (L) set-up times. Medium set-up times are randomly generated integers from the uniform distribution defined on $[1, 10]$. Having generated an instance with medium set-up times s_f for $f = 1, \ldots, F$, corresponding instances with small set-up times $\lceil s_f/2 \rceil$ and with large set-up times $2s_f$ were constructed. For each of the 12 combinations of N and F, 50 test problems with small, medium and with large set-up times were created.

Some features of our neighbourhood search algorithms are fixed on the basis of results of computational tests. For example, a decision is required on whether the larger shift sub-batch neighbourhood should be used in preference to the smaller shift job neighbourhood

in descent, simulated annealing and threshold accepting. Also, experimentation is necessary to determine whether one long run of a heuristic is preferable to a multi-start approach in which there are several short runs from different starting solutions. For tabu search, a choice between the two types of tabu list is needed. Due to the similarity between simulated annealing and threshold accepting, we do not provide results comparing different variants of threshold accepting: decisions about the design features of simulated annealing are also adopted for threshold accepting.

Before running the local search heuristics, an optimal solution to each test problem is obtained with the branch and bound algorithm of Crauwels et al. [2]. These optimal solution values are used to assess the quality of solutions generated by the heuristics.

For our initial investigations on the design features, the test problems with 50 jobs and with medium set-up times are used. Heuristics are compared by listing, for each value of F, the number of times out of 50 that an optimal solution is found (NO), the maximum relative percentage deviation (MPD) of the heuristic solution value from the optimal value, and the average computation time (ACT) in seconds on a HP9000/825. Average relative deviation are, in most cases, smaller than 0.10%, and consequently do not constitute a useful performance measure for distinguishing between different local search heuristics.

If R denotes the number of starts in a multi-start version of an algorithm, then we can adopt the following abbreviations:

DM(R): descent, where M is replaced by S or J depending on whether the shift sub-batch or shift job neighbourhood is used;

SAMK(R): simulated annealing, where M describes the neighbourhood that is used as in DM, and K is replaced by L or P depending on whether the acceptance probability scheme is linear or periodic;

TASP(R): threshold accepting with the shift sub-batch neighbourhood and a periodic pattern of threshold values;

TSJT(R): tabu search based on the shift job neighbourhood, where T is replaced by P or J according to whether the position or job tabu list is used;

GA(R): the genetic algorithm of Mason [11] with a population size of $2N$ and a maximum of N iterations (we refer to the original description for full details).

7.1 Descent methods

Table 1 presents results comparing the shift sub-batch and shift job neighbourhoods in a descent method. Multi-start versions of these algorithms are also included.

Table 1. Results for the descent methods

		DS(1)			DJ(1)			DS(8)			DJ(16)		
N	F	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT
50	4	15	1.27	0.22	8	2.42	0.04	39	0.26	1.88	33	0.86	0.36
	6	18	0.85	0.28	5	1.10	0.06	43	0.13	2.40	24	0.36	0.62
	8	17	0.80	0.28	12	1.29	0.08	39	0.20	2.56	27	0.36	0.92
	10	20	0.66	0.32	11	1.54	0.10	45	0.10	2.84	21	0.21	1.26

We first observe that the performance of DS(1) and DJ(1) is unimpressive, with relatively large deviations of the heuristic solution value from that of the optimum. A local optimum is reached fairly quickly, so the computational requirements of these descent methods are modest. Note that DS(1) is basically the heuristic proposed by Ahn and Hyun [1]. In the multi-start versions of these methods, the solution quality is significantly better. Although a direct comparison is difficult because of the smaller computation times under the shift job neighbourhood, the benefits of the better quality solutions for the shift sub-batch neighbourhood outweigh the extra computational requirements. Thus, for descent, we subsequently concentrate on multi-start versions with the shift sub-batch neighbourhood.

7.2 Simulated annealing

Table 2 gives results for simulated annealing with the shift sub-batch neighbourhood. Both linear and periodic acceptance probabilities are considered in a single and a multi-start version of the algorithm. Corresponding results are given in Table 3 for the shift job neighbourhood.

Table 2. Results for simulated annealing with the shift sub-batch neighbourhood

		SASL(1)			SASP(1)			SASL(4)			SASP(4)		
N	F	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT
50	4	40	0.11	6.82	47	0.37	7.70	40	0.23	6.82	48	0.07	7.60
	6	26	0.38	7.06	46	0.10	8.96	32	0.14	7.12	48	0.07	8.88
	8	21	0.39	6.84	46	0.06	9.34	29	0.22	7.06	46	0.06	9.32
	10	16	0.32	6.80	44	0.37	9.88	26	0.30	7.14	46	0.07	10.24

Table 3. Results for simulated annealing with the shift job neighbourhood

		SAJL(1)			SAJP(1)			SAJL(4)			SAJP(4)		
N	F	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT
50	4	29	1.21	1.22	25	1.82	0.98	37	0.47	1.20	41	0.47	1.02
	6	28	0.38	1.96	31	0.48	1.66	34	0.20	2.00	37	0.17	1.74
	8	14	0.30	2.64	25	1.01	2.32	31	0.30	2.76	39	1.01	2.44
	10	13	0.28	3.26	35	0.45	3.12	28	0.29	3.46	41	0.45	3.20

Our first observation from Tables 2 and 3 is that all multi-start versions of simulated annealing gives better results than their counterparts which are based on one long single run. Also, the shift sub-batch neighbourhood is preferred to the shift job neighbourhood, in spite of the extra computation time required by the former. Note that these conclusions are similar to those for descent. Comparing results for the linear and periodic acceptance probabilities, it is clear that many more optimal solutions generated are obtained under the latter scheme. This superiority is more pronounced when the number of families is large. We conclude that SASP(4) is the best of the tested versions of simulated annealing.

7.3 Tabu search

Table 4 gives results for the different versions of our tabu search method. The two types of tabu list are considered and a comparison is made between 100 iterations for TSJP(1) and TSJJ(1) and 50 iterations from each start for TSJP(2) and TSJJ(2). (The number of iterations is fewer than the $2N$ from each start which are used in Secion 7.4 for our comparative tests.)

Table 4. Results for tabu search													
		TSJP(1)			TSJJ(1)			TSJP(2)			TSJJ(2)		
N	F	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT
50	4	29	0.88	0.26	35	0.88	0.18	36	0.75	0.24	44	0.75	0.18
	6	23	1.00	0.48	36	0.61	0.28	31	0.35	0.44	38	0.17	0.28
	8	31	0.58	0.74	29	0.68	0.40	33	0.32	0.68	35	0.58	0.38
	10	31	0.66	1.06	29	0.66	0.54	39	0.23	0.98	41	0.33	0.48

As for descent and simulated annealing, we observe that a multi-start version of tabu search yields better results than the corresponding single start version. Comparing the results for TSJP(2) and TSJJ(2), more optimal solutions are generated for TSJJ(2) at smaller computational expense. This saving in computation is attributed to the reduced computational effort of searching the entries on a smaller tabu list length. Thus, algorithm TSJJ(2) is preferred. Further initial experiments with multi-start versions of TSJJ(R) indicate that, provided that R is not too large, superior solutions are obtained as R increases.

7.4 Comparison of different local search techniques

From the previous results and those obtained in further initial experimentation, we have identified an effective version of descent, simulated annealing, threshold accepting and tabu search. These versions, DS(N/3), SASP(4), TASP(4) and TSJJ(N/3), together with GA(2), an effective implementation of the genetic algorithm of Mason [11], are the subject of our final comparative computational tests. Recall that secondary termination tests are used in SASP(4) and TSJJ(N/3). For these final experiments, we use all test problems; with 30, 40 and 50 jobs, and with different set-up time ranges. Results are listed in Table 5.

The results in Table 5 show that all five local search heuristics are very effective. In each case, optimal solutions are generated for many of the test problems and computation times are reasonable. Even when optimal solutions are not obtained, the deviations of the heuristic solution values from those of the optimum are small. However, there is clear evidence that the TSJJ(N/3) is the best heuristic. This tabu search method fails to generate an optimal solution for only 31 of the 1800 test problems. Moreover, its computation times are less than for the other methods. The performance of descent DS(N/3) and of our simulated annealing and threshold accepting algorithms SASP(4) and TASP(4) is similar, with results slightly favouring the former. The solution quality for the genetic algorithm GA(2) is poorer than for the other methods.

Comparing results across different set-up time classes, maximum percentage deviations tend to become smaller and numbers of optimal solutions become larger as set-up times increase. It is noticeable that TSJJ(N/3) generates optimal solutions for all problems with large set-up times. There is no evidence that the relative performance of the different local search heuristics depends on the size of the set-up times.

The results show that our tabu search algorithm TSJJ(N/3) is of high quality. Its computational demands are reasonable, and it generates excellent quality solutions across a variety of test problems. Compared to previously suggested heuristic methods in the literature (DS(N/3) and GA(2)), it offers an attractive alternative. For large

problems where it is impractical to use the branch and bound algorithm of Crauwels et al. [2], TSJJ(N/3) can be applied with a high likelihood of generating a near-optimal solution.

Table 5. Comparative computational results

Set-up times	N	F	DS($N/3$)			SASP(4)			TASP(4)			TSJJ($N/3$)			GA(2)		
			NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT
S	30	4	49	0.03	0.8	47	0.10	1.1	47	0.16	1.3	50	0.00	0.4	46	0.20	1.4
		6	48	0.12	1.1	48	0.14	1.4	48	0.06	1.4	49	0.13	0.7	47	0.08	1.5
		8	46	0.22	1.3	50	0.00	1.5	45	0.22	1.6	48	0.22	1.0	48	0.11	1.6
		10	50	0.00	1.5	48	0.13	1.6	43	0.06	1.6	50	0.00	1.2	50	0.00	1.5
	40	4	47	0.09	2.2	50	0.00	2.9	48	0.01	3.1	49	0.01	0.8	36	0.31	3.4
		6	49	0.01	2.8	47	0.24	3.4	43	0.24	3.6	49	0.15	1.4	41	0.20	3.5
		8	47	0.05	3.4	48	0.11	3.6	42	0.10	3.7	49	0.05	2.2	47	0.15	3.6
		10	45	0.25	3.7	47	0.20	3.7	46	0.13	3.7	46	0.21	2.8	50	0.00	3.4
	50	4	47	0.06	4.2	45	0.13	5.1	45	0.06	5.5	47	0.13	1.4	33	0.38	6.5
| |		6	40	0.14	5.6	47	0.06	6.7	43	0.08	6.6	48	0.04	2.6	39	0.13	7.2
		8	42	0.07	6.8	44	0.13	7.2	43	0.09	7.0	48	0.05	3.9	43	0.10	6.8
		10	48	0.01	8.2	43	0.30	7.7	42	0.13	7.3	45	0.28	5.5	49	0.01	6.7
M	30	4	49	0.35	0.7	50	0.00	1.0	47	0.35	1.1	50	0.00	0.3	46	0.47	1.3
		6	48	0.09	0.8	49	0.13	1.1	50	0.00	1.2	50	0.00	0.5	49	0.18	1.2
		8	49	0.25	0.9	50	0.00	1.2	48	0.25	1.3	49	0.25	0.8	50	0.00	1.2
		10	46	0.07	1.0	48	0.05	1.2	48	0.02	1.3	50	0.00	0.9	50	0.00	1.2
	40	4	47	0.05	1.9	48	0.04	2.5	47	0.13	2.9	48	0.05	0.7	45	0.11	2.9
		6	49	0.02	2.2	49	0.32	2.8	49	0.03	3.1	48	0.05	1.1	48	0.17	3.0
		8	48	0.11	2.7	50	0.00	3.2	46	0.08	3.3	50	0.00	1.7	49	0.01	2.9
		10	46	0.09	2.5	48	0.14	2.8	45	0.07	3.0	49	0.00	2.1	49	0.03	2.6
	50	4	46	0.26	3.9	48	0.04	4.8	47	0.03	5.5	50	0.00	1.2	40	1.42	5.7
		6	47	0.03	4.9	45	0.10	5.7	46	0.07	6.2	50	0.00	2.1	44	0.17	5.9
		8	43	0.08	5.5	46	0.21	6.2	45	0.03	6.3	47	0.05	3.1	46	0.08	5.5
		10	46	0.10	6.1	45	0.09	6.4	43	0.09	6.4	50	0.00	4.2	50	0.00	5.6
L	30	4	50	0.00	0.4	50	0.00	0.7	49	0.06	0.8	50	0.00	0.3	47	0.94	1.0
		6	48	0.11	0.5	50	0.00	0.7	49	0.07	0.9	50	0.00	0.4	50	0.00	0.9
		8	49	0.04	0.6	50	0.00	0.8	47	0.16	1.0	50	0.00	0.5	50	0.00	0.9
		10	47	0.05	0.7	50	0.00	0.9	49	0.03	1.0	50	0.00	0.7	50	0.00	0.8
	40	4	48	0.24	1.4	50	0.00	1.8	49	0.23	2.3	50	0.00	0.5	47	0.14	2.5
		6	48	0.08	1.5	49	0.01	2.0	49	0.01	2.3	50	0.00	0.8	50	0.00	2.2
		8	50	0.00	1.7	50	0.00	2.1	50	0.00	2.4	50	0.00	1.2	50	0.00	2.2
		10	49	0.02	1.6	49	0.04	1.8	47	0.28	2.2	50	0.00	1.5	50	0.00	1.9
	50	4	48	0.05	3.1	50	0.00	3.9	49	0.05	4.6	50	0.00	1.0	45	0.42	4.8
		6	50	0.00	3.4	49	0.05	4.1	49	0.00	4.7	50	0.00	1.6	47	0.12	4.4
		8	47	0.11	3.6	49	0.07	4.2	47	0.08	4.6	50	0.00	2.2	49	0.11	4.2
		10	45	0.03	4.0	46	0.06	4.4	44	0.14	4.9	50	0.00	3.1	48	0.41	4.3
Average			47	0.09	2.7	48	0.08	3.1	46	0.10	3.3	49	0.05	1.6	46	0.18	3.2

8 Concluding remarks

Local search methods are studied for a single machine scheduling problem with families of jobs. Although each family can be split into batches of jobs which are processed contiguously, a set-up time is required for each batch. A descent method of Ahn and Hyun [1] is available (DS(1)), but it suffers the disadvantage of generating many solutions that are local optima rather than global optima. To circumvent this disadvantage, we have developed a multi-start version of this algorithm, together with simulated annealing, threshold accepting and tabu search algorithms. Computational results are provided which compare different versions of the same algorithm, and which give an overall comparison of algorithms. Included in this overall comparison is the genetic algorithm of Mason [11].

Amongst these algorithms, the best quality solutions are generated by tabu search with multiple runs (TSJJ($N/3$)), and this method also requires the least computation time. Descent, simulated annealing and threshold accepting produce slightly better results than Mason's genetic algorithm. However, non-standard versions of simulated annealing and threshold accepting are needed to achieve these results: the temperature in simulated annealing and the threshold value in threshold accepting both follow a periodic pattern.

Acknowledgement

This research was partly supported by the International Association for the Promotion of Cooperation with Scientists from the Independent States of the Former Soviet Union, INTAS-93-257.

References

[1] Ahn, B.-H. and Hyun, J.-H., "Single facility multi-class job scheduling", *Computers and Operations Research,* 17, 265–272 (1990).

[2] Crauwels, H.A.J., Hariri, A.M.A., Potts, C.N., and Van Wassenhove, L.N., "A branch and bound algorithm for single machine scheduling with batching to minimize total weighted completion time", to appear, (1995).

[3] Downsland, K.A., "Some experiments with simulated annealing techniques for packing problems", *European Journal of Operations Research,* 68, 389–399 (1993).

[4] Dueck, G., and Scheuer, T., "Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing", *Journal of Computational Physics,* 90, 161–175 (1990).

[5] Eglese, R.W., "Simulated annealing: a tool for operational research", *European Journal of Operations Research,* 46, pp. 271-281 (1990).

[6] Ghosh, J.B., "Batch scheduling to minimize total completion time", *Operations Research Letters,* to appear (1995).

[7] Glover, F., "Tabu search, Part I", *ORSA Journal on Computing,* 1, 190–206 (1989).

[8] Gupta, J.N.D., "Single facility scheduling with multiple job classes", *European Journal of Operations Research,* 8, 42–45 (1988).

[9] Johnson, D.S., Aragon, C.R., McGeoch, L.A. and Schevon, C., "Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning", *Operations Research,* 37, 865–892 (1989).

[10] Kirkpatrick, A., Getlatt Jr., C.D., Vechi, M.P., "Optimization by simulated annealing", *Science,* 220, 671–680 (1983).

[11] Mason, A.J., " Genetic Algorithms and Scheduling Problems", Ph.D. thesis, Department of Engineering, University of Cambridge, U.K. (1992).

[12] Mason, A.J. and Anderson, E.J., "Minimizing flow time on a single machine with job classes and setup times", *Naval Research Logistics* 38, 333–350 (1991).

[13] Monma, C.L. and Potts, C.N., "On the complexity of scheduling with batch setup times", *Operations Research,* 37, 798–904 (1989).

[14] Pirlot, M., "General local search heuristics in combinatorial optimization: a tutorial", *Belgian Journal of Operations Research, Statistics and Computer Science*, 32, 8–67 (1992).

[15] Potts, C.N., "Scheduling two job classes on a single machine", *Computers & Operations Research,* 18, 411–415 (1991).

[16] Potts, C.N., and Van Wassenhove, L.N., "Single Machine Tardiness Sequencing Heuristics", *IIE Transactions,* 23, 346–354 (1991).

[17] Potts, C.N., and Van Wassenhove, L.N., "Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity", *Journal of the Operational Research Society,* 43, 395–406 (1992).

[18] Smith, W.E., "Various optimizers for single-stage production", *Naval Research Logistics Quarterly,* 3, 59–66 (1956).