"LOCAL SEARCH HEURISTICS FOR SINGLE MACHINE SCHEDULING WITH BATCH SET-UP TIMES"

by

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

94/07/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, 77305 Fontainebleau, France.

Local Search Heuristics for Single Machine Scheduling with Batch Set-Up Times

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, 1994

Abstract

Local search heuristics are developed for a problem of scheduling jobs on a single machine. Jobs are partitioned into batches, and a set-up time is necessary when there is a switch in processing jobs from one batch to jobs of another batch. The objective is to minimize the total weighted completion time. Three alternatives neighbourhood search methods are developed: multi-start descent, simulated annealing and tabu search. The performance of these heuristics is evaluated based on a large set of test problems, and the results are also compared with those obtained by a genetic algorithm.

Keywords: scheduling, batches, set-up time, local search heuristics, simulated annealing, tabu search.

1 Introduction

Many practical scheduling problems involve processing several batches of related jobs on common facilities where a setup time is incurred whenever there is a switch from processing a job in one batch to a job in another batch.

In this paper we consider a single machine scheduling problem with sequence independent setup times. 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.

Monma and Potts [10] consider a variety of single machine batch scheduling problems under the assumption that sequence dependent batch set-up times satisfy the 'triangle inequality': the set-up time associated with a changeover from batch a to c is assumed to take no longer than that for changeover from batch a to b followed by a changeover from batch b to c (our sequence independent setup times clearly satisfy this triangle inequality). They present a dynamic programming algorithm and use it to show that, under this set-up time structure, the problem of minimizing total weighted completion time is efficiently solvable for a fixed number of batches. However, this dynamic programming algorithm is largely of theoretical interest unless the number of batches is very small. For the special case of only two batches, Potts [12] shows how the $O(N^4)$ time complexity can be improved to $O(N^3)$ time, where N is the total number of jobs.

Gupta [5] 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. However, it can be used for constructing 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 intragroup SWPT property (see also Monma and Potts [10]) to reduce the size of the neighbourhood.

Besides this intragroup SWPT property, other dominance criteria are derived by Mason and Anderson [8]. They used these criteria to reduce the size of the search tree generated by a branch and bound procedure. Further reduction is obtained by calculating a lower bound on the flow time of the remaining unscheduled jobs and using this lower bound to fathom some additional nodes in the branch and bound search tree. Their algorithm is able to solve problems with up to 30 jobs.

The complexity of this batching problem for an arbitrary number of batches is still open (see Potts and Van Wassenhove [14]). Nevertheless, 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 ten years, several new methods have been proposed and developed: simulated annealing, tabu search and genetic algorithms. An introductory overview of these methods and some applications is given by Pirlot [11].

In this paper, several neighbourhood structures are described and are used in a descent, simulated annealing and tabu search algorithms. The performance of these methods is also compared with a genetic algorithm developed by Mason [9].

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

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 B batches. Each batch b, for $1 \leq b \leq B$, contains n_b jobs. All jobs are available for processing at time zero, and are to be scheduled on a single machine. We let p_{ib} denote the processing time of the i'th job in batch b, and $w_{ib} > 0$

is the associated weight.

A sequence independent set-up time $s_b \geq 0$ is incurred whenever a job in batch b is processed immediately after a job in a different batch. Also, an initial set-up time s_b is incurred if a job from batch b is the first to be processed. We can regard the set-up times as representing a set-up job. In any feasible schedule, each of the original jobs must immediately follow either another job in the same batch or the appropriate set-up job.

It is convenient to regard a sequence S as a series of runs, where a run is a maximal consecutive subsequence of jobs in S from the same batch. If R_k is the k'th run in S then

$$S = (R_1, R_2, ..., R_r).$$

Each run R_k can be viewed as a single composite job with processing time T_k and weight W_k. If run R_k contains jobs $u, u+1, \dots, v$ from batch b, then T_k and W_k are calculated as follows:

$$T_k = s_b + \sum_{i=u}^{v} p_{ib} \qquad \text{and} \qquad W_k = \sum_{i=u}^{v} w_{ib}.$$

We also define the weighted processing time of run R_k as $\text{WPT}(R_k) = T_k/W_k$.

Based on these definitions, Mason and Anderson [8] derive several dominance rules, the most important of which for our study concern the ordering of the jobs within a batch and the ordering of the runs. For any optimal schedule:

1. the i'th job in batch b precedes the j'th job in batch b if $p_{ib}/w_{ib} \leq p_{jb}/w_{jb}$ (intragroup SWPT property);

2. run R_k precedes run R_l if $\text{WPT}(R_k) \leq \text{WPT}(R_l)$. (intergroup SWPT property).

Throughout this paper, we assume that the jobs within each batch have been renumbered so that $p_{1b}/w_{1b} \leq \ldots \leq p_{n_b,b}/w_{n_b,b}$ for $1 \leq b \leq B$ and the batches have been renumbered so that $T_1/W_1 \leq \ldots \leq T_B/W_B$. Any sequence that satisfies the intragroup and itergroup SWPT properties can be used as an initial solution for a neighbourhood search heuristic. For example, by assigning all jobs in each batch to a single run, these properties define a sequence.

A more general rule, which allows some batches to be split into two or more runs, can be derived from the heuristic of Gupta [5] for an analogous problem. Such a method constructs partial schedules using the 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 resulting sequence does not necessarily satisfy the SWPT rule. Therefore, a sequence of adjacent-run interchanges (called "switch-runs") are performed until no further improvements can be found.

3 Descent search

A neighbourhood search method requires an initial solution, which can be constructed by a dispatching rule or it can be chosen at random. A neighbour of this solution 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 the considered problem a neighbourhood can be constructed in various ways by shifting jobs or subruns forward and backward, whereby the intragroup SWPT order is maintained.

The most general definition of a neighbourhood for this problem can be found in Ahn and Hyun [1]. Consider a sequence $(J_1, .., J_k, J_{k+1}, .., J_l, J_{l+1}, .., J_m, J_{m+1}, .., J_N)$, where jobs $(J_{l+1}, ..., J_m)$, which belong to the same batch b form a run R. The neighbourhood consists of sequences created by a forward or backward shift of a subrun. In a forward shift, a subrun $(J_{l+1}, .., J_{l+r})$ of run R, where $1 \leq r \leq m - l$, is removed from its original position and inserted it immediately before some job J_{k+1}, where k is selected so that the subsequence $(J_{k+1}, .., J_l)$ contains no jobs of batch b. A backward shift is be defined analogously. We refer to this neighbourhood as *shift-subrun*.

Included in the shift-subrun neighbourhood is a special case in which only one job is shifted. For a forward shift of a single job, we are restricted to setting $r = 1$, and to choosing k such that J_k and J_{k+1} are from a different batch. Thus, the first job of the run R can be shifted forward to any position between two runs until J_k is from the same batch as run R, at which stage no further forward shifting is allowed. In a backward shift, J_m, the last job of run R, is moved in a similar way to a later position. This *shift-job neighbourhood* is smaller than the more general shift-subrun neighbourhood.

In a descent method, a series of moves from one solution to another solution in its neighbourhood is performed, where each move results in an improvement of the objective function value. 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.

Another possibility is to allow interchanges which lead to an increase in the objective function value. Consequently, the procedure can escape from a local minimum and continue its search for a global minimum.

4 Simulated annealing

In a simulated annealing procedure, solutions which improve upon the current solution value are accepted, while those which cause a deterioration in the objective function value are accepted according to a given probabilistic acceptance function. Following the lead of Kirkpatrick et al. [7] 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 metaheuristic. A review can be found in Eglese [3]. 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 in the objective function value. The parameter t is known as the *temperature*.

In our implementation of simulated annealing, the temperature is derived from an acceptance probability K as described by Potts and Van Wassenhove [13]. More precisely, K is equal to the probability of accepting a one percent increase in Z, where Z is the value of the best solution known so far. If L denotes the number of distinct acceptance probability levels considered, we set $K_{i+1} = K_i - (K_0 - K_f)/(L-1)$ where K_0 and K_f are the initial and final acceptance probabilities. In order to compute the actual acceptance probability for any given increase in the objective function value, the temperature t is found using $K_i = \exp(-0.01Z/t)$, thereby yielding $t = -0.01Z/\ln K_i$.

For each acceptance probability level, the complete neighbourhood is searched systematically. Tests are performed with both both the shift-subrun and the shift-job neighbourhoods that are defined in Section 3. The termination criterion is defined by fixing a priori the number of levels L.

Initial computational experiments show that the performance of simulated annealing is worse than that of our multi-start descent method. Increasing the number of levels L in simulated annealing does not result in a better performance. The main reason for this is the strictly descending nature of the acceptance probability function.

At the beginning of the simulated annealing algorithm, the acceptance probability is high, and so many moves are accepted that increase the objective function value. With a large neighbourhood, many iterations are performed during one level, and sometimes too many interchanges that increase the objective function value are accepted. The search process then resembles a random walk amongst neighbouring solutions. This phenomenon occurs even when the initial acceptance probability is relatively small. To give the process a chance to evolve to a local optimum, each level in simulated annealing is preceded by an application of descent (where no moves that increase the objective function value are accepted). The initial application of descent before the first level searches for strict improvements on the initial solution value.

At the end of the simulated annealing algorithm, the acceptance probability is low and effectively a pure descent procedure is executed which may become trapped in a local optimum. Therefore, when the number of levels is increased, much of the extra computation time tends to be wasted.

Because of these drawbacks, we propose an acceptance probability that changes periodically and has the following properties. The acceptance probability is high throughout the algorithm so that it is always possible to escape from a local minimum. These levels are interwoven with descent steps creating the possibility to evolve to a local optimum. The random walk effect can be diminished by using a cutoff, as in Johnson et al. [6]: as soon as a certain number of moves at a level have been accepted, no further searches are performed on that level. In our method, however, the acceptance probability is decreased within a level each time an objective function value increase is accepted. In this way, all elements of the neighbourhood can be explored during one level without moving too far away from interesting solution subspaces.

More precisely, we use the following acceptance probability function which is periodic and its period is 5 levels:

$$
\begin{aligned}
K_i &= K_f + Ax^2, \\
&\text{where} \begin{cases} x &= (i \text{ } modulo \text{ } 5 - 2)/2 \quad (x \in \{-1.0, -0.5, 0.0, 0.5, 1.0\} \\ A &= (1+R)(K_0 - K_f)/2 \quad \text{with R a random number between 0 and 1.} \end{cases}
\end{aligned}
$$

From this probability K_i, the temperature t_i is computed using

$$t_i = \begin{cases} -0.01Z/lnK_i & i = 1,3,5,\ldots \\ 0.0 & i = 0,2,4,\ldots \end{cases}$$

The iterations for $i = 0,2,4,\ldots$ correspond to the application of descent prior to each acceptance probability level. Within a positive acceptance probability level, the temperature t_i is halved if an increased objective function value is accepted. Thus, we set $t_i = 0.5t_i$, with the aim of reducing the random walk phenomenon.

The use of a 'cooling' scheme which also heats up when deteriorating moves are no longer accepted, has already been suggested by Downsland [2]. 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, phases of cooling down and heating up are more precisely fixed and are not correlated to solution-specific conditions.

5 Tabu search

A deterministic approach for escaping from a local optimum is offered by tabu search (see Glover [4]). In this search heuristic, exploration continues after the first local optimum is found by allowing non-improving moves. However, certain moves are forbidden or *tabu* for a few iterations to prevent a closed loop of movements between a few sequences.

Conventionally, the complete neighbourhood must be searched in each iteration to find the best possible move. Thus, it is advantageous to use a small neighbourhood, and therefore we use the shift-job neighbourhood. Also, the search is also stopped as soon as a non-tabu neighbour is found which improves upon the current objective function value.

A tabu list is constructed, the purpose of which is to prevent the search heuristic from returning to a previously visited local minimum. There are several ways to characterize a tabu move. In the first version of our tabu search algorithm, the return of a specific job to a certain position is prohibited for a number of iterations after the job has moved from that position. This requires that both the number of the job and the number of the position in the sequence that no longer can be assigned to the job are stored on the tabu list. After each job shift, one or two elements are added to the end of the tabu list (whilst the same number of elements are removed at the beginning so that the total number of elements in the list remains constant). The element that is always added, consists of the number of the shifted job and its original position; if the shifted job moves moves exactly one position in the sequence so that two jobs are transposed, than another element with the number of the other transposed job and its original position is also added to the tabu list.

In the second version of our algorithm, the job that is shifted backwards or forwards is given tabu status. Thus, only the number of the shifted job is stored on the tabu list, and any move that shifts a job on the tabu list is forbidden. Clearly, more neighbours are forbidden than in the first version of our algorithm, and it is sometimes possible that such a tabu neighbour has a better objective function value than previously generated solutions. To prevent the occasional loss of good solutions, an aspiration level criterion is incorporated. If the solution value of a tabu neighbour is better than that for all solutions already generated, then its tabu status is overridden. The choice of the jobs that are added to the tabu list is the same as in the first version of the algorithm.

Because of the more specific characteristics of a tabu move in the first version, a longer tabu list is necessary to avoid cycling. For the second method, the classical tabu list size of 7 is satisfactory.

The last parameter to be fixed is the stopping rule. The search terminates after the execution of a prespecified number of iterations.

6 Computational results

The heuristics were tested by coding them in ANSI-C and running them on a HP 9000/825. Fifty random problems were generated for different combinations of numbers of jobs and batches: the values used are $N = 30, 40, 50$ and $B = 4, 6, 8, 10$. The set-up times, job processing times and weights were uniformly distributed integers between 1 and 10. The jobs were distributed uniformly across batches, subject to the constraint that a total of N jobs is required. An optimal solution to each problem is obtained with a standard branch and bound method.

For each test set of 50 problems and each local search algorithm, the number of times the optimal solution (NO) is found and the maximum relative percentage deviation (MPD) of the heuristic solution value from the optimal value are tabulated. The other column (ACT) gives the average computation time in seconds on a HP9000/825. The average relative percentage deviation is in most cases smaller than 0.10% and is not a useful performance measure for distinguishing between different algorithms.

The following abbreviations are used:

AH: the descent heuristic of Ahn and Hyun which uses the shift-subrun neighbourhood;

DJ: the descent heuristic which uses the shift-job neighbourhood;

SANT: simulated annealing, where N is replaced by S or J depanding on whether the shift-subrun or shift-job neighbourhood is used, and T is replaced by L or P depending on whether the acceptance probability is linear descending or periodic;

TSL: tabu search based on the shift-job neighbourhood, where L is replaced by P if the tabu list stores jobs and the corresponding positions to which they cannot return, and L is replaced by J if the tabu list stores jobs which cannot be shifted.

For each algorithm, a number is appended between parentheses to indicate the number of restarts from different initial solutions. When restarts are used, the first initial solution is constructed with the adapted heuristic of Gupta [5]; the others are generated at random.

Simulated annealing based on neighbourhood of Ahn and Hyun.

In Table 1, the descent method of Ahn and Hyun is compared with simulated annealing procedures based on the shift-subrun neighbourhood. SASL starts with a acceptance probability of 10%; SASP with 50%. Both of the simulated annealing algorithms perform 100 levels, and the complete neighbourhood is searched at each level.

Table 1. Results for the shift-subrun neighbourhood

N	B	AH(1)			AH(8)			SASL(1)			SASP(1)		
		NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT
50	4	15	1.27	0.22	39	0.26	1.88	40	0.11	6.82	47	0.37	7.70
	6	18	0.85	0.28	43	0.13	2.40	26	0.38	7.06	46	0.10	8.96
	8	17	0.80	0.28	39	0.20	2.56	21	0.39	6.84	46	0.06	9.34
	10	20	0.66	0.32	45	0.10	2.84	16	0.32	6.80	44	0.37	9.88

The heuristic of Ahn and Hyun (AH(1)) can find good solutions with little computation time. If it is restarted several times (AH(8)), the number of times the exact solution is found can be improved. With the linearly descending acceptance probability function (SASL), poor results are obtained especially when the number of batches is large. For $B = 10$, SASL(1) finds an optimal solution for only for 16 problems, whereas AH(1) generates 20 optimal sequences. This illustrates the phenomenon of drifting away from an initial solution to worse regions of the solution space during the initial stages of the search when acceptance probabilities are high. Even worse solutions result when the starting probability is higher. By changing the acceptance probability function so that fewer moves that increase the objective function value are accepted during the first few levels, as in SASP(1), we obtain better results. However, SASP(1) also requires the most computation time because overall more moves that increase the objective function value are accepted than in SASL. This is due to the higher starting probability (50%) and the high probability levels at various stages throughout the algorithm.

Simulated annealing based on shift-job neighbourhood.

In Table 2, results for the smaller shift-job neighbourhood are given. The initial acceptance probabilities for simulated annealing are the same as those used for the shift-subrun neighbourhood: SAJL starts with 10%, and SAJP with 50%. Even though 200 acceptance probability levels are used, which is double the number for the shift-subrun neighbourhood, less computation time is required for SAJL and SAJP.

Table 2. Results for the shift-job neighbourhood

N	B	DJ(1)			DJ(16)			SAJL(1)			SAJP(1)		
		NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT
50	4	8	2.42	0.04	33	0.86	0.36	29	1.21	1.22	25	1.82	0.98
	6	5	1.10	0.06	24	0.36	0.62	28	0.38	1.96	31	0.48	1.66
	8	12	1.29	0.08	27	0.36	0.92	14	0.30	2.64	25	1.01	2.32
	10	11	1.54	0.10	21	0.21	1.26	13	0.28	3.26	35	0.45	3.12

The overall results are worse for the shift-job neighbourhood, but the same observations can be made as for Table 1. Multiple runs of descent (DJ(16)) result in a better performance than with just one run (DJ(1)). DJ(16) is also better than the simulated annealing procedure SAJL(1). Again, the number of optimal solutions generated by SAJL(1) becomes fewer when the number of batches is large. For the periodic acceptance probability function (SAJP(1)), the results are slightly better, but for $B = 4$ and $B = 8$, we note that DJ(16) still finds more times the optimal sequence and requires less computation time.

Tabu search based on the shift-job neighbourhood.

Table 3 gives results for our tabu search method. The tabu list for TSP contains 25 elements, where the size of the list is only 7 for TSJ. Both TSP(1) and TSJ(1) terminate after 100 iterations. However, 50 iterations only are performed from each initial solution in TSP(2) and TSJ(2), so that execution times are comparable.

Table 3. Results for tabu search

N	B	TSP(1)			TSJ(1)			TSP(2)			TSJ(2)		
		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

Although the same neighbourhood is used as in SAJL(1) and SAJP(1), both of the tabu search procedures TSP(1) and TSJ(1) require less computation time while comparable results are obtained. Results for TSP(2) and TSJ(2) indicate that it is advantageous to restart the tabu search method from a different initial sequence, rather than to perform all available iterations from a single starting solution. The smaller computation times for TSJ than for TSP are attributed to the shorter tabu list length.

Multiple runs of simulated annealing.

Results in Table 1 show that a descent algorithm can be improved by taking the best sequence obtained by running the procedure several times starting from different initial solutions. Also, for tabu search, it is profitable to perform two different runs with fewer iterations instead of one long run, as indicated by the results in Table 3. In Table 4, results with multiple simulated annealing runs are shown: starting from 4 initial solutions, SASL and SASP are run for 25 levels, and SAJL and SAJP are run for 50 levels.

Table 4. Results for multiple runs of simulated annealing

N	B	SASL(4)			SASP(4)			SAJL(4)			SAJP(4)		
		NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT
50	4	40	0.23	6.82	48	0.07	7.60	37	0.47	1.20	41	0.47	1.02
	6	32	0.14	7.12	48	0.07	8.88	34	0.20	2.00	37	0.17	1.74
	8	29	0.22	7.06	46	0.06	9.32	31	0.30	2.76	39	1.01	2.44
	10	26	0.30	7.14	46	0.07	10.24	28	0.29	3.46	41	0.45	3.20

Computation times in Table 4 are approximately the same as those listed for the corresponding algorithms in Tables 1 and 2. Thus, meaningful comparisons can be made. Results indicate that multiple shorter runs are preferable to one long run. Again, when the number of batches is large, we cannot obtain as many optimal solutions with a linear acceptance probability function (SASL(4) and SAJL(4)) as with the periodic acceptance probability function (SASP(4) and SAJP(4)).

Comparison of different local search techniques.

In the first four tables, results for $N = 50$ only are given because for $N = 30$ and $N = 40$ differences between the different methods are less pronounced. In Table 5, we compare the results obtained using 'good' versions of descent, simulated annealing and tabu search method, with those obtained with a genetic algorithm.

A genetic algorithm has already been developed by Mason [9]. He uses an encoding scheme where the chromosome no longer directly represents the solution sequence. Because the order of jobs in a batch is known, it is sufficient to indicate which jobs in this ordering appear at the start of a new run. This can be achieved binary valued genes with the value 1 signifying a new run; otherwise, a zero is used. From this information, every run for each batch can be constructed. By using the SWPT rule to order the runs from all batches, a full sequence can be generated.

Reproduction is based on the "stochastic uniform selection" procedure whereby fitness scaling is applied to a culled population (chromosomes with a bad fitness are removed). The crossover and mutation operators are defined in the usual way. In addition to using a fixed number of generations as stopping rule, a number of different termination criteria are integrated.

The four methods all employ multiple runs from different starting solutions. For descent and tabu search the number of runs is $N/3$, whereas two runs of simulated annealing and the genetic algorithm are performed. Parameter settings for the different algorithms are as follows:

AH	: the descent heuristic of Ahn and Hyun;
SASP	: simulated annealing with $N/2+10$ levels, and a run is prematurely stopped if no overall improvement is observed during $N/2-5$ levels;
TSJ	: tabu search with a maximum of $2N$ iterations, and the procedure is prematurely stopped if no overall improvement is observed during $N/2-5$ iterations;

GA : the genetic algorithm of Mason with a population size $2N$, maximum number of iterations $2N$, f_0-culling $s_f = 0.0$, overall agelevel=N, agelevel=$N/2$, pop convergence=0.90, no elitism, scaling $C = 2.0$ (for the exact meaning of all these parameters, see Mason [9]).

Table 5. Comparative results for different local search method

N	B	AH($N/3$)			SASP(2)			TSJ($N/3$)			GA(2)		
		NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT	NO	MPD	ACT
30	4	49	0.35	0.64	50	0.00	0.76	50	0.00	0.30	47	0.47	1.26
	6	48	0.09	0.76	49	0.13	0.82	50	0.00	0.50	49	0.18	1.22
	8	49	0.25	0.92	49	0.11	0.90	49	0.25	0.72	50	0.00	1.22
	10	46	0.07	0.98	46	0.26	0.96	50	0.00	0.92	50	0.00	1.16
40	4	47	0.05	1.78	48	0.15	2.10	48	0.05	0.66	46	0.11	2.92
	6	49	0.02	2.10	49	0.32	2.42	48	0.05	1.12	47	0.08	3.04
	8	48	0.11	2.58	44	0.29	2.70	50	0.00	1.70	49	0.01	3.04
	10	46	0.09	2.44	43	0.19	2.26	49	0.00	2.08	49	0.03	2.70
50	4	46	0.26	3.68	48	0.04	4.12	50	0.00	1.12	41	1.42	6.08
	6	47	0.03	4.74	42	0.22	5.24	50	0.00	2.06	43	0.13	6.10
	8	43	0.08	5.52	44	0.40	5.56	47	0.05	3.14	46	0.05	5.68
	10	46	0.10	5.92	46	0.37	5.78	50	0.00	4.32	49	0.01	6.04

The four methods are each very successful in generating good quality solutions to the weighted completion time problem with batch set-up times within a reasonable computation time. For more than 80% of problems in each test set, an optimal solution is found by each algorithm. The maximum relative deviation is in all cases less than 0.5%, except for $N = 50$ and $B = 4$ where GA(2) produces a maximum relative deviation of 1.42%. GA(2) also requires the most computation time; approximately 6 seconds to solve one problem of the set with $N = 50$.

TSJ($N/3$) outperforms the other methods; and especially when the number of batches is small, it requires the least computation time. Its computational requirements are highly dependent on the number of batches, because of the structure of the neighbourhood. When the number of batches is large, there are many runs and the complete neighbourhood is searched by shifting the first and last job of each run.

Similarly, for the larger shift-subrun neighbourhood in AH(2) and SASP(2), more computation time is required when there are more batches. The numbers of optimal solutions found are approximately the same for both methods. However, the maximum relative deviation is larger for SASP than for AH. This is probably due to the fact that SASP only uses two different initial solutions. As results for SASP(4) in Table 4 show, the maximum deviation can be reduced by multiple runs, but at the expense of more computation time.

The computational requirement of GA(2) is independent of the number of batches. The unreliability of the method, however, is observed for $N = 50$ and $B = 4$ where the maximum relative deviation is large and only 41 optimal solutions (out of 50) are obtained, although its performance is better for $B = 8$ and $B = 10$.

7 Conclusions

Local search methods are studied for a single machine scheduling problem with multiple batches, where a decision not to process jobs of a batch together results in additional set-up time. We have developed a multi-start descent heuristic (DJ(1)), and on comparison with the procedure of Ahn and Hyun (AH(1)), the latter is found to give better results, but at the expense of more computation time because of the larger neighbourhood size. Both descent algorithms suffer from the problem of getting trapped in a local optimum. We have tried to circumvent this disadvantage with three different techniques: multiple runs of the descent method, simulated annealing and tabu search.

Amongst these algorithms, the best quality solutions are generated by tabu search with multiple runs (TSJ($N/3$)), and this method also requires the least computation time. However, multi-start descent with a large neighbourhood structure (AH($N/3$)) is also quite effective With a pure cooling scheme, the performance of simulated annealing (SASL(1)) is unsatisfactory relative to the other methods. Although improvements are observed after incorporating consecutive phases of cooling and heating and using more than one starting solution (SASP(4)), the resulting algorithm requires more computation time than tabu search.

8 References

1. Ahn, B.-H. and Hyun, J.-H., "Single facility multi-class job scheduling", *Computers and Operations Research,* Vol.17, No.3, pp. 265-272 (1990).

2. Downsland, K.A., "Some experiments with simulated annealing techniques for packing problems", *European Journal of Operations Research,* 68, pp. 389-399 (1993).

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

4. Glover, F., "Tabu search, Part I", *ORSA Journal on Computing,* Vol.1, No.3, pp. 190-206 (1989).

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

6. 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,* Vol.37, No.6, pp. 865-892 (1989).

7. Kirkpatrick, A., Getlatt Jr., C.D., Vechi, M.P., "Optimization by simulated annealing", *Science,* 220, pp. 671-680 (1983).

8. Mason, A.J. and Anderson, E.J., "Minimizing flow time on a single machine with job classes and setup times", *Naval Res. Logist.* 38, pp. 333-350 (1991).

9. Mason, A.J., " Genetic Algorithms and Scheduling Problems", Ph.D. thesis, Department of Engineering, University of Cambridge (1992).

10. Monma, C.L. and Potts, C.N., "On the complexity of scheduling with batch setup times", *Operations Research,* 37, pp. 798-904 (1989).

11. Pirlot, M., "General local search heuristics in combinatorial optimization: a tutorial," *Belgian Journal of Operations Research, Statistics and Computer Science,* Vol. 32, No. 1-2, pp. 8-67 (1992).

12. Potts, C.N., "Scheduling two job classes on a single machine", *Computers and Operations Research,* Vol 18., No. 5, pp. 411-415 (1991).

13. Potts, C. N., and Van Wassenhove, L. N., "Single Machine Tardiness Sequencing Heuristics," *IIE Transactions,* Vol. 23, No. 4, pp. 346-354 (1991).

14. 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,* Vol. 43, No. 5, pp. 395-406 (1992).