ELSEVIER

European Journal of Operational Research 83 (1995) 330–346

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

A comparative study of both standard and adaptive versions of threshold accepting and simulated annealing algorithms in three scheduling problems

C.K.Y. Lin [a,b,*], K.B. Haley [a], C. Sparks [a]

[a] *School of Manufacturing and Mechanical Engineering, University of Birmingham, Birmingham B15 2TT, UK*

[b] *Department of Applied Statistics & Operational Research, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon Tong, Hong Kong*

Abstract

Local search techniques are reported to be effective for many optimization problems with arbitrary objective functions and constraints. This paper compares simulated annealing with a less frequently mentioned approach, threshold accepting. Three scheduling problems of jobs with arbitrary time lags in a single-server system are considered. The optimization criteria used are maximum completion time and mean completion time. Two adaptive versions of both techniques: the adaptive neighbourhood search and the adaptive temperature/threshold values are proposed which guide the future search according to the recent search performance. These adaptive concepts prove to be effective and may be applied into other local search techniques. Computational results show that threshold accepting algorithms achieve better or comparable performance with simulated annealing in the standard and adaptive versions. Incorporating descent approaches into threshold accepting may lead to substantial improvement.

Keywords: Threshold accepting; Simulated annealing; Adaptive search; Scheduling; Jobs with arbitrary time lags

1. Introduction

Simulated Annealing (SA) has been a useful local search technique for obtaining near-optimal solutions in a wide variety of combinatorial optimization problems. Much literature has been written on SA. An extensive survey of the theory, implementation and applications is given in van Laarhoven and Aarts [10]. SA, a convergent algorithm (with proofs given in [10] and also by Lundy and Mees [13]), has been applied to problems in manufacturing by Chen and Srivastava [3], Venugopal and Narendran [20], Vakharia and Chang [19], resource-contrained scheduling by Jeffcoat and Bulfin [8], machine scheduling by Osman and Potts [16], Ogbu and Smith [15], Matsuo et al. [14], vehicle-scheduling by Wright [21], graph partitioning by Johnson et al. [9], bin packing by Dowsland [5] and in other environments.

Threshold Accepting (TA), also a convergent algorithm (with proof given by Althofer and Koschnick [2]), has been referenced less often. TA adopts a simpler acceptance criteria for new solutions and does not require the generation of random numbers and exponential functions. Suc-

* Corresponding author.

SSDI 0377-2217(95)00011-9

cess over SA has been reported for the Travelling Salesman Problem of 442 cities and for finding good error-correcting codes by Dueck and Scheuer [6]. Sampson and Weiss [18] implicitly applied the concept of TA to tackle the generalized resource constrained project scheduling problem. However, for the study of job shop scheduling by Aarts et al. [1], TA (having different sets of parameters but similar running time to SA) was found to be outperformed by SA and genetic local search. Hence, one purpose of this study is to compare the standard versions of SA and TA under the same parameter settings in three scheduling problems.

Both the solution quality of SA and TA can be improved by making appropriate choices of the neighbourhood and the cooling schedule in SA or the threshold parameters in TA. However, standard SA and TA each require a fixed set of predetermined search parameters. The search explores the complete solution space for acceptable solutions with no exploitation in the dominant subsets of the solution nor adjustment based on recent search performance. Reeves [17] proposed a revised approach in tabu search to balance between exploration of the solution space and the exploitation of the information obtained for the $n/m/P/C_{\max}$ scheduling problem. Superior results were reported over the 'brute-force' standard tabu search and one of the best SA procedures. This paper proposes two adaptive approaches to enable the SA or TA search to be self-guided in finding respectively the dominant neighbourhoods and the effective parameters in the acceptance criteria.

The problem environment involves jobs with arbitrary time lags in a single-server system. In many practical systems, delays in handling, transportation, and transmission, or thinking time between phases of an operation, may occur as a time lag. Scheduling jobs with multiple phases of work with arbitrary time lags to minimize maximum completion time or mean completion time in this study is an extension of work from Lin and Haley [11].

This paper is organized as follows. The differences in the structures of SA and TA are compared in Section 2. The same parameter settings for standard SA and TA in the three problems in this paper are presented in Section 3. Two adaptive versions, guiding the search towards 'promising' neighbourhoods and varying the temperature/threshold values according to recent search performance are proposed in Sections 4 and 5 respectively. A descent approach is incorporated into the standard and adaptive versions of TA (Section 6). Sections 7–9 describe the three scheduling problems respectively,with the implementation of standard and adaptive SA and TA algorithms, followed by their computational results. The last section concludes the findings of this study.

2. Comparison of the structures of SA and TA

Standard SA and TA are both convergent algorithms which provide the ability to escape from local minima through an acceptance criteria for inferior solutions. Both require a good neighbourhood structure and search parameters to improve their search efficiency. SA and TA will gradually approach descent algorithms which accept only equally good solutions as running time increases. The SA approach (with the TA differences shown in brackets) for a minimization problem is explained as follows:

SA [TA] Approach

Step 1. Initialization:

Choose an initial solution

Choose an initial temperature $T > 0$ [threshold $T > 0$]

Step 2. Generation of new solution and comparison:

Choose a new neighbouring solution from the old solution

Compute

Δ = function of new solution
  − function of old solution

If $\Delta \leq 0$ [$\Delta <$ threshold T]
  Then old solution = new solution
  If new solution value
    < current best solution value
    current best solution = new solution

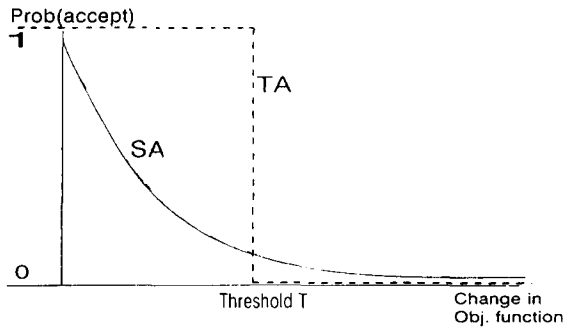

Fig. 1. Probability of acceptance curve for SA and TA.

Else with probability $e^{-\Delta/T}$ [this step does not exist in TA]

old solution = new solution

Step 3. Update new temperature [threshold]:

If there is no improvement in the objective function over many iterations

Then lower temperature T [threshold T]

If the last temperature T [threshold T] has been used, stop.

Else repeat from Step 2.

In general, the temperatures in SA and the threshold parameters in TA need not be the same. In this study, they are given the same values for comparison. Dueck and Scheuer [6] stated that "the essential difference between SA and TA consists of the different acceptance rules. TA accepts every new configuration which is not much worse than the old one (SA accepts worse solutions only with rather small probabilities)". Fig. 1 shows the probability of acceptance curves in SA and TA. TA accepts all solutions below the threshold value T and rejects those above it. SA accepts all better solutions and will accept inferior solutions with some positive probability, decreasing exponentially as the change in the objective function increases. Both the threshold values in TA and the temperatures in SA are chosen to be *non-increasing* values which is essential for their asymptotic convergence property to hold. In the following section, the parameters required by standard SA and TA will be described with their implementation on the three problems in Sections 7–9.

3. Parameter settings in standard SA and TA

The choice of the cooling schedule in SA or the threshold parameters in TA, the neighbourhood structure and search manner are important factors for the algorithm efficiency. To ensure an effective comparison between SA and TA, they are assigned the same parameter settings. Their only difference lies in the acceptance criteria of new solutions.

3.1. Cooling schedule in SA / threshold parameters in TA

The following controls the acceptance criteria in SA and TA:

– Initial temperature/threshold value (T_1). For each problem in sections 7–9, T_1 is set to be a large upper bound of the corresponding objective function such that nearly all new solutions are accepted initially to allow for diversity.

– Final temperature/threshold value (T_K). T_K may be a predetermined small number such that the algorithms approach descent algorithms which accept only equally good solutions near the end of the run. (Accepting equally good solutions, instead of strictly better ones may lead to substantial improvement as reported by Jeffcoat and Bulfin [8].) T_K is chosen to be 1 (as in Osman and Potts [16]) for the three problems in Sections 7–9.

– Length of the temperature/threshold sequence (K). K is chosen here to depend polynomially on the problem size or subject to the time limit given.

– Decrement rule for the temperature/threshold sequence. The decrement of successive temperatures is chosen to be small such that the stationary distributions of the system for two succeeding temperatures should be close. The decrement rule for the three problems here, originally proposed in Lundy and Mees [13], used in Osman and Potts [16], is given by

$$T_{i+1} = \frac{T_i}{1 + \beta T_i}, \quad i = 1, \ldots, K, \tag{1}$$

where

$$\beta = \frac{T_1 - T_K}{(K-1)T_1 T_K}.$$

($\{T_i\}$ is generated as a set of real numbers.)

– Number of iterations for each temperature/threshold. This is determined such that the running time is similar to other algorithms for comparison.

3.2. Neighbourhood structure, search manner and initial solution

The representation of a solution may be the full sequence of all jobs or a partial sequence which is proved sufficient to represent the full sequence. The choice of neighbourhood, search manner and the initial solution are problem-specific.

Common types of neighbourhood used to generate new solutions are the shift (S), interchange (I) and adjacent pairwise interchange (API) as studied in Osman and Potts [16]. S neighbourhood shifts a chosen job in the current sequence either before or after another chosen job. I neighbourhood will simply interchange the two chosen jobs to obtain a new neighbour. API neighbourhood exchanges a pair of adjacent jobs. Their advantages are the easy computability. More sophisticated neighbourhoods, tailored to the specific problem, may be defined at the expense of more computational time. A random search is adopted for the three problems here. The initial solution is chosen from a good sequencing rule or heuristic with low complexity. The choice of the initial solution will effect the type of neighbourhood used if improvement is to be made effectively. The specific details of the parameter settings tailored to each of the three problems are given in Sections 7–9 respectively.

4. Adaptive neighbourhood search (ANS)

This adaptive version is based on the concept of the existence of dominant neighbourhoods. In the optimal sequences, there may exist both a subset of jobs which dominates the objective and some non-critical jobs whose ordering has no significant effect on the objective. Permuting the dominant neighbourhoods will give larger change to the objective (hopefully better ones) than expending effort on the non-critical jobs. Instead of searching for acceptable solutions in the entire solution space as in the standard method, the adaptive neighbourhood search (ANS) method for SA and TA is introduced to guide the search towards 'promising' subsets of the solution according to their recent performance. Given N jobs in the solution, they are partitioned into S (a factor of N) ordered subgroups of similar sizes N/S. Equal amount of permutations will be performed in each selected subgroup $g \in \{1, \ldots, S\}$ (by choosing random element(s) in the subgroup to permute) and its search performance $I(g)$ is recorded. ($I(g)$ may be measured by the number of times the current best objective value is improved, or the magnitude of improvement, etc). A normalised function f, known as the adaptive function f, will transform $I(g)$ into a set of selection probabilities of each subgroup for permutation in the next set of iterations. (Initially, the S subgroups are examined systematically to initiate the function f and its cumulative probability function. Thereafter, a random number in (0, 1) is generated each time and compared with the cumulative function of f to select a subgroup with replacement.) $I(g)$ is recorded again for each subgroup $g \in \{1, \ldots, S\}$ to generate new probabilities $\{f(g)\}$. Hence, if dominating subgroups exist, the search will concentrate in further permutations of their elements. Moderate functions of f should be used. An exaggerated f may result in bias towards selecting certain subgroups before the system reaches a steady state.

In addition to the parameters in the standard methods (temperature/threshold sequences, neighbourhood structure, search manner and initial solution), ANS requires the appropriate choices of the following:

– Job partitioning. This defines the set of exclusive and exhaustive neighbourhoods for the adaptive search. The two criteria used here for partitioning N jobs into subgroups of similar sizes are either by their positions or their natural ordering. If by positions, the S subgroups formed are $\{J_{[1]}, J_{[2]}, \ldots, J_{[N/S]}\}$, $\{J_{[N/S+1]}, \ldots, J_{[2N/S]}\}, \ldots, \{J_{[(S-1)N/S+1]}, \ldots, J_{[N]}\}$ where $J_{[i]}$ denotes the job in the current i-th position. In the case of natural ordering, the S subgroups formed will be $\{J_1, \ldots, J_{N/S}\}, \ldots, \{J_{(S-1)N/S+1},$

..., J_N}. (Jobs are initially labelled according to some problem-specific priority rule.) For problems involving precedence-related tasks, partitioining by their positions seem to be a more reasonable choice. The size of the subgroup, S, which defines the boundaries of job partitioning, needs to be determined through some preliminary experiments.

– Adaptive function f. f determines the selection probability of subgroups for permutation in the next set of iterations. It may be chosen as a positive function of $I(g)$ (recent search performance in subgroup g). Say if $I(g)$ measures the number of times the objective function is improved in the last set of iterations on subgroup g, a simple choice of f could be

$$f(g) = \frac{I(g)+1}{\sum_{s=1}^{S}(I(s)+1)}, \quad g = 1,\ldots,S.$$

(1 is added to each $I(g)$ value to avoid the case when all $I(g)$ equal zero, leading to an undefined $f(g)$ value. This also allows a positive probability for each subgroup to be chosen.)

5. Adaptive temperature / threshold values (ATV)

An approach to allow the algorithm, SA or TA, to control its acceptance criteria of new solutions in different search environments is to make its temperature/threshold values (T) adaptive. For problems where the objective function is very 'hilly' (consists of many local minima and maxima), it will be more difficult for standard SA and TA to locate the global minimum for a minimization problem. Since both algorithms require decreasing functions for their parameters (temperature in SA and the threshold value in TA), the longer the running time, the less tendency will they have to escape from a local minimum. The adaptive temperature/threshold values (ATV) which adjusts the T values based on the recent search performance is introduced.

As in standard SA and TA, the initial T values are set to be large for the search to accept a large variety of solutions. Adjusting the T values in the initial stage will make the T values even larger and search effort will be inefficiently spent in accepting much inferior solutions. The adaptive procedure for the T values will only begin after a certain time has elapsed. The initial search will follow a steady non-increasing function D of T values. Thereafter, each new T value will be adjusted according to the search performance of the last T value. The measure of the search performance used here is the number of times the best objective decreases in the ith set of iterations using T_i, denoted by $n(i)$. If $n(i)$ from the current value T_i is no less than $n(i-1)$ from the previous value T_{i-1}, and both $n(i)$, $n(i-1)$ are not 0, the new value T_{i+1} will follow the steady non-increasing function D. Otherwise, it may imply that the search area is near a local or global minimum. Two sets of iterations are independently run for the two trial values of T (T_L and T_S) and the number of times the best objective decreases (n_L and n_S respectively) are recorded. A large T value T_L ($=\alpha T_i$ where $\alpha > 1$) will be tried to enable the search to jump out from a possibly local minimum. A small T value T_S ($= T_i/\alpha$) will speed up the search in ending up at a minimum. If the larger of the number of times of decrease in the objective function between T_L and T_S exceeds that from the current T_i, then that 'T' giving the larger number will be chosen as T_{i+1}. T_{i+2} is also set to the value of T_{i+1} to enable more search under the same T. If the improvement from T_L and T_S is no better than that from T_i, a moderate T_{i+1} is estimated by

$$T_{i+1} = \begin{cases} \frac{1}{2}(T_i + T_L) & \text{if } n_L > n_S, \\ \frac{1}{2}(T_i + T_S) & \text{if } n_L \le n_S. \end{cases}$$

ATV TA is summarised in the following. (ATV SA is only different in the acceptance criteria of new solutions.)

ATV TA algorithm. (adjusting threshold values)

Step 1. Initialization:

Choose an initial solution

$i = 1$ (1st set of iterations)

Choose an initial threshold $T_i > 0$

Step 2. Standard TA for the first few set of iterations:

$n(i) = 0$ (number of times the objective function improves under T_i)

Run subroutine STANDARDTA
Generate T_{i+1} from a steady non-increasing function D on T_i
Next $i(\rightarrow i+1)$
If i is in the first few set of iterations, repeat Step 2.
Else goto Step 3.

Step 3. Adaptive:
$n(i) = 0$
Run subroutine STANDARDTA
If $n(i) \geq n(i-1)$ and NOT $(n(i) = n(i-1) = 0)$ (more improvement in the best objective from T_i than T_{i-1})
Then generate T_{i+1} from D on T_i
Else run STANDARDTA independently with each of the following thresholds:

$$T_L = \alpha T_i \text{ and } T_S = (1/\alpha)T_i$$

where $\alpha > 1$
Record the number of times of improvement in the best objective as n_L and n_S respectively
If $\max(n_L, n_S) > n(i)$
  If $n_L \geq n_S$, $T_{i+1} = T_L$ and $n_{i+1} = n_L$
  Else $T_{i+1} = T_S$ and $n_{i+1} = n_S$
  $T_{i+2} = T_{i+1}$
  Next i $(\rightarrow i+2)$
Else $T_{i+1} = \frac{1}{2}(T_i + T_j)$, $j \in \{L, S\}$
where $n_j = \max\{n_L, n_S\}$ (n_S in case of ties)
Next i $(\rightarrow i+1)$
If the stopping criterion is reached, stop
Else repeat from Step 3.

Subroutine STANDARDTA

Step i. Generation of new solutions and comparison:
Choose a new neighbouring solution from the old solution
Compute
$\Delta =$ function of new solution
  – function of old solution
If $\Delta <$ threshold T
  Then old solution = new solution
  If new solution value < current best solution value
  current best solution = new solution;
  $n(i) = n(i) + 1$
Next iteration. Proceed to Step ii

Step ii. Repeat:
If the end of the ith set of iterations is reached, return to main program
Else repeat from Step i.

ATV adopts the same neighbourhood structure, search manner and initial solution as the standard method. Additional parameters to be considered involve the following:

– Non-increasing function D and initial temperature/threshold values T. The standard method (SA or TA) will run for an initial set of T values to accept any exchanges before the T values are made adaptive in ATV. In the standard method, the number of iterations is predetermined. Hence, the initial T values and its length in ATV are predetermined, depending on whether the initial stage or the adaptive stage records more improvement. D may adopt the same decrement rule (Eq. (1)) as in the standard method.

– Adjusting factor α for T_L and T_S. During the adaptive stage, when the current search performance is worse than the previous one, or both are 0, a large T_L and a small T_S are used to help escape from a local minimum and speed up the search in ending at a minimum respectively. α determines the degree of enlargement and diminuition of T_i to give $T_L(=\alpha T_i)$ and $T_S(=(1/\alpha)T_i)$ respectively. Moderate values of $\alpha > 1$ should be chosen to avoid the search from jumping around (α too large) or being trapped in a local minimum (α too small). The relationship of T_L, T_S and α is not specific and a simple choice is taken here.

– Stopping criterion. The number of iterations in the standard method is predetermined, independent of the previous and current search performance. ATV may adopt a stopping rule which terminates the search when there is no improvement in the objective function for some γ consecutive T values. (γ may be a function of the problem size.) Alternatively, a target time limit is given for fair comparison with other methods.

5.1. Asymptotic convergence of ATV

The determination of $\{T_i\}$ in the adaptive stage ensures that each T_i is non-zero. When the re-

cent search performance is 0 or decreasing ($n(i) = n(i-1) = 0$ or $n(i) < n(i-1)$), T is allowed a large and small value (T_L and T_S) respectively. When the non-improving search performance persists (i.e. $\max(n_L, n_S) \leq n(i)$), T will not increase indefinitely. The reason is even as T increases, the worst case of $n(i) = n_L = n_S = 0$ becomes more probable. By the construction of ATV, T_{i+1} is determined by

$$T_{i+1} = \tfrac{1}{2}(T_i + T_S)$$
$$= \tfrac{1}{2}(T_i + (1/\alpha)T_i) < T_i$$

(since $\alpha > 1$). Hence, $\{T_i\}$ has the possibility of increment at each stage, but will still maintain a non-increasing sequence asymptotically when no improvement results. The non-increasing property of sequence T is also possessed by the standard SA or TA, which enables them to converge asymptotically towards the global optimal solutions. ATV does similarly.

6. Descent methods

Descent methods often provide a useful tool in the optimization of many scheduling problems. Analytically, they cannot escape from local minima or 'flat' surfaces where cycling may result. Hence, they will not always provide the optimal solution. However given a good initial solution, the performance can be better than the local search methods with varying parameters. Another advantage is in the fewer number of parameters need to be estimated. Hence, a descent approach will be applied to each of the standard and adaptive (ANS and ATV) TA algorithms, giving the three descent methods with notations Des, Des ANS and Des ATV respectively. For maximum completion time (respectively mean completion time) problems, a sequence of 1s ($1/N$ s, where N is the number of jobs) will be chosen as the entire threshold sequence in standard TA, ANS TA and the initial threshold sequence in ATV TA. (This allows for accepting equally good solutions on the 'flat' surfaces instead of strictly better ones if 0 is used. Accepting equally good solutions instead of strictly better ones was found to lead to significant improvement in [8].) SA, whose acceptance probability for worse solutions is always non-zero, will only approach a descent algorithm as running time increases.

For the Des ANS and Des ATV methods, the number of subgroups (S) and the adjusting factor (α) are respectively chosen from the value giving the best results in ANS TA and ATV TA.

Concluding from Sections 3–6, the following methods of SA and TA in 1)–7) and the randomly generated schedules in 8) will be compared on the three scheduling problems in Sections 7–9. All computations are run on an IBM3090 machine.

1) Standard method.

2) ANS (with job partitioning on positions (p) or natural ordering (n), number of subgroups (S)).

3) ATV (with adjusting factor (α)).

4)–7) Des, Des ANS, Des ATV (descent versions of TA in 1)–3) respectively, where such do not exist for SA).

8) Random.

7. Problem 1: Scheduling two-phase jobs with time lags to minimize maximum completion time

This problem considers a set of N jobs where each job J_i consists of 2 operations a, b (a precedes b) with processing times p_i^a, p_i^b respectively. All phase a tasks are readily available. A minimum delay time lag of d_i ($i = 1, \ldots, N$) must elapse after the completion of phase a and before phase b can begin ($i = 1, \ldots, N$). The objective is to minimize the maximum completion time (or makespan) for all jobs on a single server. This problem is proved to be strongly NP-hard in Lin and Haley [11]. However, local search algorithms can be easily applied here due to the specific optimal property. It was proved in [11] that in at least one optimal sequence, all phase b tasks start after the completion of all phase a tasks. (However, processing sequences of the phase a tasks and the phase b tasks may be different in an optimal sequence.) When the objective is minimum makespan, any idle time is avoided. Hence, once the phase a tasks are sequenced (with no

idle gap), the phase b tasks will be processed on a first-come-first-served basis. The solution representation in the search methods (standard, adaptive methods of SA/TA, descent methods, random) is only the phase a sequence, which determines the resulting makespan. The phase a sequence is then perturbed to search for a new sequence in the standard and adaptive methods according to their respective neighbourhood structure. The following problem-specific parameters will complement those in Sections 3–6.

7.1. Common setting for the local search methods

For each instance, two runs are carried out with different initial sequences and the corresponding effective neighbourhood to generate new neighbours: 1) longest-delay jobs first (LGD) sequence with adjacent pairwise interchange (a selected task is exchanged with its immediate predecessor in the current solution); 2) earliest arriving phase b (EAB) sequence with interchange neighbourhood. (The better result from the two runs is taken.) Random search (for task(s) to permute) is preferred to deterministic search in this problem from experimentation. (For the ANS or Des ANS methods, the random search is on elements in the selected subgroup.)

For the non-descent methods which require a temperature/threshold sequence, the initial T_1 is chosen as a large upper bound of the makespan to allow for diversity. T_1 is estimated by the sum of phase a processing times + makespan of sequence $\delta_{(d',p^{b'})}$, where $\delta_{(d',p^{b'})}$ is the first-come-first-served sequence of jobs with no phase a but revised delay and phase b parameters $\{d', p^{b'}\}$. $\{d'_i\}$ and $\{p_i^{b'}\}$ are respectively arranged in non-increasing order of their magnitudes. (A long delay matched with a long phase b is possibly the worst combination for the delay and phase b system.)

7.2. Algorithm-specific parameters

To make effective comparison among all the algorithms, similar running time is used by adjusting parameters in the temperature/threshold sequence in each type of SA and TA.

Standard methods. A sequence of length of 30 N (N is the problem size) is adopted for $\{T_i\}$ in SA and TA. If a solution is better than the current one, it is accepted and a new neighbour is generated without altering the current T_i. Otherwise, T_i is decremented according to Eq. (1).

ANS. For each T_i, S (a factor of problem size N) subgroups are selected and N/S iterations are performed in each selected subgroup. By adopting a sequence length of 30 for $\{T_i\}$, the total number of iterations is $S \times (N/S) \times 30 = 30N$, which approaches that of the standard methods. As there are two runs for each instance, the job partitioning in ANS by natural ordering will have jobs labelled according to the LGD or EAB initial sequence. 3 different values of S (number of partitioned subgroups) are used in each problem size.

ATV. The running time of standard SA is taken as the stopping criterion. The standard SA/TA method with a sequence of 30 for $\{T_i\}$ is allowed to run for the first half of the time, i.e. the first 15 T_i values are taken. Thereafter, T_i becomes adaptive as described in Section 5. For each T_i, T_L or T_S, N iterations are run to calculate the search performance of $n(i)$, n_L or n_S respectively. A moderate value of 2 is chosen for the adjusting factor α.

7.3. Computational results

100 random instances are generated for each problem size of 30, 50 and 100 jobs. Table 1 shows the parameters $\{p_i^a, d_i, p_i^b\}$ as integers generated uniformly from various ranges of values, representing server utilization between 0% and 50%. Instances are examined to see if any belong to the special cases of optimality in Lin and Haley [11]. The rest will require the more sophisticated local search methods. These constitute (76, 71, 72) out of the 100 instances generated for $N = (30, 50, 100)$ jobs respectively. Any feasible sequence with no idle gap or makespan equal to the span of the longest job has already reached optimality.

The performance measures used for comparison include the relative deviation index (R), number of proved optimal cases (some may be optimal but not proved) and the average computer

Table 1
Parameter ranges and the number of initially solved and unsolved cases in Problem 1 for $N = 30, 50, 100$ jobs

N	p^a	d	p^b	Number of solved cases (out of 20)	Total number of unsolved cases (out of 100)
30	1–19	1–599	1–19	2	76
	1–19	1–999	1–19	9	
	1–19	1–799	1–19	7	
	1–9	1–599	1–19	1	
	1–19	1–499	1–9	5	
50	1–19	1–999	1–19	1	71
	1–19	1–1999	1–19	9	
	1–19	1–1499	1–19	8	
	1–9	1–999	1–19	4	
	1–19	1–799	1–9	7	
100	1–19	1–2999	1–19	8	72
	1–19	1–1999	1–19	0	
	1–9	1–1999	1–19	5	
	1–19	1–1999	1–9	10	
	1–19	1–2499	1–19	5	

time required (on the IBM3090 machine). The relative deviation index measures the relative solution quality of an algorithm A among other algorithms. It is defined by

$$R(A) = (C_A - C_B)/(C_W - C_B),$$

where C_A, C_B and C_W are makespans from algorithm A, the best algorithm (giving the smallest makespan) and the worst one (giving the largest makespan) respectively. (R is calculated as an average value for instances where not all algorithms give the same objective value.) R is recommended here since it is an invariant measure when an equal amount of delay is added to each job.

Table 2 shows the relative performance of all algorithms with similar running time. It was observed that in almost every instance, the initial sequence from longest-delay jobs first (LGD) with an API neighbourhood provides a better setting for further search than earliest arriving phase b (EAB) with the interchange neighbourhood. The random generated sequences are obviously inferior to other methods. The findings on the local search methods from Table 2 are listed:

– Comparing the performance of SA and TA in each version, there is no significant difference in the standard and ANS methods. However, ATV TA performs better than the ATV SA in all three problem sizes.

– The best performance comes from the descent methods, followed closely by ATV TA. They outperform the standard and ANS methods, which have comparable performance with each other.

– Incorporating a descent approach into standard TA, ANS TA and ATV TA substantially improves each of their performance. (This coincides with the results in [7] which also had the objective of minimizing makespan. The less effective method, genetic algorithm, gave great improvement after incorporating a descent approach.) This implies that the LGD sequence with an API neighbourhood is a good setting here for straight-forward descent approach. Among the different descent methods, no one value is always best.

The adjusting factor (α) in Des ATV is tested with different values from 1 to 8 and no single value is found to be the best for each problem size. However, a large α is suggested for large problem size due to the resulting increase in the objective value.

8. Problem 2: Scheduling single-phase jobs with release times to minimize mean completion time ($1/r_j/\bar{C}$)

This NP-complete problem has been studied in [4] and is used to test on the performance of the local search methods. It considers scheduling a set of N jobs having unequal release times $r_j \geq 0$ and processing times p_j, $j = 1, \ldots, N$ and the objective is to minimize mean completion time. (Problem 2 is a special case of problem 3 in Section 9. Problem-specific methods here are modified for the more complex structure of problem 3.)

8.1. Common setting for local search methods

In addition to the parameters described in Sections 3–6 for individual methods, the following are problem-specific. The solution represen-

Table 2
Performance of SA and TA versions of local search methods for $N = 30$, 50 and 100 jobs

$N = 30$	Rel. dev. index (out of 59 [a])		Number of proved optimal cases (out of 76)		Average time (CPU sec)	
Methods	TA	SA	TA	SA	TA	SA
Standard	0.247	0.232	13	15	0.240	0.233
ANS(p, $S = 2$)	0.404	0.371	12	13	0.234	0.228
ANS(n, $S = 2$)	0.325	0.379	13	12	0.236	0.233
ANS(p, $S = 5$)	0.399	0.356	13	13	0.231	0.233
ANS(n, $S = 5$)	0.351	0.407	12	14	0.235	0.225
ANS(p, $S = 10$)	0.414	0.398	12	12	0.238	0.235
ANS(n, $S = 10$)	0.365	0.389	13	13	0.232	0.230
ATV($\alpha = 2$)	0.161	0.296	14	13	0.241	0.244
Des	0.145	–	17	–	0.212	–
Des ANS(p, $S = 2$)	0.179	–	17	–	0.210	–
Des ANS(n, $S = 2$)	0.178	–	16	–	0.215	–
Des ATV($\alpha = 2$)	0.123	–	16	–	0.233	–
Random	0.683		7		0.240	
$N = 50$	(out of 53 [a])		(out of 71)			
Standard	0.332	0.310	7	6	0.679	0.668
ANS(p, $S = 2$)	0.364	0.301	7	8	0.655	0.626
ANS(n, $S = 2$)	0.288	0.290	7	7	0.633	0.639
ANS(p, $S = 5$)	0.308	0.347	6	7	0.662	0.652
ANS(n, $S = 5$)	0.343	0.379	9	8	0.623	0.641
ANS(p, $S = 10$)	0.377	0.349	7	8	0.663	0.640
ANS(n, $S = 10$)	0.314	0.306	8	8	0.639	0.653
ATV($\alpha = 2$)	0.145	0.287	8	6	0.680	0.696
Des	0.146	–	9	–	0.624	–
Des ANS(p, $S = 5$)	0.124	–	9	–	0.608	–
Des ANS(n, $S = 2$)	0.096	–	10	–	0.603	–
Des ATV($\alpha = 2$)	0.123	–	10	–	0.658	–
Random	0.749		6		0.654	
$N = 100$	(out of 55 [a])		(out of 72)			
Standard	0.329	0.346	5	5	2.710	2.772
ANS(p, $S = 4$)	0.369	0.431	4	5	2.650	2.555
ANS(n, $S = 4$)	0.370	0.384	5	5	2.646	2.592
ANS(p, $S = 5$)	0.413	0.356	3	5	2.657	2.564
ANS(n, $S = 5$)	0.455	0.378	4	5	2.620	2.580
ANS(p, $S = 10$)	0.343	0.396	5	3	2.642	2.617
ANS(n, $S = 10$)	0.382	0.389	4	4	2.631	2.611
ATV($\alpha = 2$)	0.252	0.341	6	4	2.784	2.810
Des	0.084	–	8	–	2.562	–
Des ANS(p, $S = 10$)	0.154	–	5	–	2.531	–
Des ANS(n, $S = 4$)	0.136	–	7	–	2.483	–
Des ATV($\alpha = 2$)	0.165	–	9	–	2.726	–
Random	0.804		1		2.727	

ANS(p/n, S) denotes the ANS algorithm with job partitioning on positions (p)/natural ordering (n) and S number of subgroups.
[a] Number of cases with same objective given by all methods is 17 out of 76, 18 out of 71 and 17 out of 72 for $N = 30$, 50 and 100 jobs respectively. Hence, the remaining 59 ($N = 30$), 53 ($N = 50$) and 55 ($N = 100$) cases are used in calculating the relative deviation index.

tation is the full sequence of N jobs. The two sequencing rules often used to tackle this problem are the earliest completion time (ECT) rule and the earliest starting (EST) rule ([4]). The initial solution is taken from the ECT or EST sequence which yields the smallest mean completion time. A random search is used to select job(s) for permutation.

Jobs are initially labelled according to the EST rule, i.e. $r_1 \leq r_2 \leq \cdots \leq r_N$. The initial temperature/threshold value for the non-descent local search methods is taken from the mean completion time of the largest processing time first sequence α, starting at the latest release time. In other words,

$$T_1 = r_N + \frac{1}{N}\sum_{j=1}^{N}(N+1-j)p_{\alpha_j}$$

where α_j is the j-th job in sequence α.

8.2. Algorithm-specific parameters

Similar running time is targeted for each method by adjusting the specific parameters.

Standard methods. The neighbourhoods of shift (S), interchange (I) and adjacent pairwise interchange (API) (Section 3.2) are adopted here. A sequence length of 100 is set for $\{T_i\}$ in SA and TA and N iterations are run for each T_i.

ANS. As in the standard methods, the sequence length of $\{T_i\}$ is chosen to be 100 with N iterations run for each T_i. Experimenting on the different combinations of job partitioning (by positions (p) or natural ordering (n)) with neighbourhood structure (S, I, API) in Lin [12] suggests that partitioning of jobs by their natural ordering with the interchange neighbourhood is an effective choice. Two different values of the number of subgroups (S) are used in each problem size.

ATV. The running time of standard SA determines the stopping criteria here. The standard method is run initially for one-third of the time limit ($\sim$ the first $\frac{1}{3} \times 100$ of $\{T_i\}$ used) before the adaptive stage begins. Again for each T_i, T_L or T_S, N iterations are performed. The interchange neighbourhood is adopted after observing the performance of different neighbourhoods in the standard and ANS methods in [12]. The adjusting factor takes two values of $\alpha = 2$ and 4.

Random. As in ATV, the neighbourhood structure is the interchange (I).

Table 3
Performance of SA and TA versions of different local search methods in Problem 2 for $N = 20$ and 50 jobs

$N = 20$	Rel. dev. index (out of 130 [a])		Average time (CPU sec)	
Methods	TA	SA	TA	SA
Standard(S)	0.404	0.421	0.166	0.178
Standard(I)	0.374	0.410	0.156	0.172
Standard(API)	0.544	0.557	0.168	0.176
ANS(n, I, $S = 2$)	0.237	0.397	0.170	0.185
ANS(n, I, $S = 4$)	0.479	0.484	0.172	0.187
ATV(I, $\alpha = 2$)	0.134	0.164	0.173	0.177
ATV(I, $\alpha = 4$)	0.141	0.225	0.173	0.179
Des(I)	0.128	–	0.163	–
Des ANS(n, I, $S = 2$)	0.249	–	0.162	–
Des ATV(I, $\alpha = 2$)	0.106	–	0.173	–
Random	0.986		0.181	
$N = 50$	(out of 155 [a])			
Standard(S)	0.823	0.865	0.918	0.937
Standard(I)	0.829	0.849	0.867	0.911
Standard(API)	0.953	0.890	0.949	0.969
ANS(n, I, $S = 2$)	0.256	0.806	0.924	0.965
ANS(n, I, $S = 5$)	0.328	0.704	0.922	0.966
ATV(I, $\alpha = 2$)	0.316	0.410	0.924	0.957
ATV(I, $\alpha = 4$)	0.307	0.429	0.922	0.963
Des(I)	0.151	–	0.891	–
Des ANS(n, I, $S = 2$)	0.165	–	0.836	–
Des ATV(I, $\alpha = 4$)	0.118	–	0.908	–
Random	0.985		0.966	

S, I, API: shift, interchange, adjacent pairwise interchange neighbourhood

n: job partitioning on the natural ordering of jobs; S: number of subgroups

[a] number of cases with same objective given by all methods is 90 and 65 out of 220 for $N = 20$ and 50 jobs respectively. Hence, the remaining 130 ($N = 20$) and 155 ($N = 50$) cases are used in calculating the relative deviation index

8.3. Computational results

By taking the same parameter ranges for (r_j, p_j) from [4], 220 instances are randomly generated for each problem size of 20 and 50 jobs. The relative deviation index (R) and the average computational time are taken as measures of the

comparative performance of all methods in Table 3. R follows the same formula as in Section 7.3 but is defined on the objective of mean completion time ($\bar{C}$) instead of the makespan (C) in problem 1. Similar to problem 1, R is invariant when the release time of each job is extended by an equal amount.

From Table 3, all methods prove to be superior than the random sequences. The ECT sequence gives a smaller mean completion time than the EST sequence in most cases, hence is usually used to provide the initial solution for all methods. The following observations are made on the local search methods:

– TA outperforms SA in all the standard, ANS and ATV methods except in one case (standard(API) in $N = 50$). This effect is most significant in ANS for larger problem size of 50 jobs.

– Both the adaptive versions of ANS and ATV improve on the standard versions. This effect becomes more obvious for larger problem size of $N = 50$ jobs. The best ANS methods (ANS TA(n, I, $S = 2$)) is comparable with the ATV methods, with ATV methods having better performance in $N = 20$ and the reverse is true when $N = 50$.

– The descent methods almost give the best relative performance among all methods (except for Des ANS(n, I, $S = 2$) when $N = 20$) within the specified time limit. Among the three descent methods, Des ATV gives slightly better results than Des and both are preferred over Des ANS. (This may be explained by the fact that the fixed job partitioning in ANS or Des ANS has violated the reachability property of all feasible solutions.) This implies that given a good starting solution (ECT sequence usually), descent approaches are useful tools in getting better solutions (local/global minimum) efficiently.

9. Problem 3: Scheduling two-phase jobs with time lags to minimize mean completion time

This problem has the same job structure as problem 1 (Section 7) with each job i characterised by the three parameters (p_i^a, d_i, p_i^b), $i = 1, \ldots, N$. The objective is however to minimize mean completion time. As problem 2, a special case of this problem, is NP-complete, so is this problem. Problem 3 is more difficult than problems 1 and 2 since the phase a and phase b tasks may interleave in the optimal sequence, with the precedence relationships becoming more significant than in the previous two problems. The solution representation is the full sequence of $2N$ tasks (phase a and phase b). Problem-specific methods for problem 2 are modified to be used here.

9.1. Common setting for the local search methods

Complementing the parameters in Sections 3–6, the following gives the initial sequence, neighbourhood structure, search manner and initial temperature/threshold (T_1) specific to this problem. The earliest completion time (ECT) concept in problem 2 is modified to give a sequence, denoted as EC, for the $2N$ interleaving (phase a and phase b) tasks. EC provides the initial sequence for all methods.

A shift neighbourhood taking into consideration of the precedence relationships of phase a and phase b is chosen. (Interchange neighbourhood has more constraints since four tasks are effected in one move. API neighbourhood offers slow improvement unless a good starting solution is provided.) A random search for the chosen task and its new position in the sequence is adopted (as in problems 1 and 2).

T_1 is taken from the objective value of a worst case – the sequence α which processes jobs, with no interleaving, in the order of largest sum of (phase a + phase b) processing times first, i.e.

$$T_1 = \frac{1}{N} \sum_{j=1}^{N} (N + 1 - j)\left(p_{\alpha_j}^a + d_{\alpha_j} + p_{\alpha_j}^b\right),$$

where α_j denotes the j-th job in α.

9.2. Algorithm-specific parameters

For the three problem sizes of $N = 6$, 15 and 30 jobs tested in the computations (Section 9.3), the time limits given to all methods are 6, 29 and

Table 4
Performance comparison of local search methods (standard, ATV, ANS, descent, hybrids, ASC, random) for $N = 6$, 15 and 30 jobs

$N = 6$					Avg % absolute error [No. of optimal cases out of 5] (Avg time in CPU sec)									
Sample	p^a	d	p^b	% idle: processing	standard SA	standard TA	ATV SA	ATV TA	ANS SA ($S = 2$)	ANS TA ($S = 2$)	Des	Des ATV	Des ANS ($S = 2$)	Random
1	1–25	0–200	26–50	2.364	0.000	0.000	0.000	0.000	0.462	0.462	1.433	1.433	1.660	0.795
					[5]	[5]	[5]	[5]	[3]	[3]	[1]	[1]	[1]	[1]
					(6.07)	(6.00)	(6.00)	(6.00)	(7.13)	(6.52)	(5.89)	(6.00)	(6.50)	(6.42)
2	1–25	75–175	26–50	8.707	0.000	0.000	0.000	0.000	0.000	0.000	2.679	2.679	3.387	0.871
					[5]	[5]	[5]	[5]	[5]	[5]	[3]	[3]	[2]	[0]
					(6.07)	(6.02)	(6.02)	(6.02)	(7.12)	(6.55)	(5.89)	(6.02)	(6.52)	(6.42)
3	26–50	50–150	1–25	10.651	0.000	0.184	0.265	0.310	0.366	0.366	1.416	1.549	1.416	2.078
					[5]	[3]	[2]	[1]	[2]	[2]	[1]	[0]	[1]	[0]
					(6.07)	(6.00)	(6.00)	(6.00)	(7.11)	(6.49)	(5.90)	(6.00)	(6.51)	(6.42)
4	26–50	50–150	1–25	10.837	0.207	0.145	0.029	0.144	1.752	1.752	2.222	2.475	2.713	2.256
					[4]	[1]	[3]	[3]	[0]	[0]	[0]	[0]	[0]	[0]
					(6.08)	(6.00)	(6.00)	(6.00)	(7.12)	(6.50)	(5.91)	(6.00)	(6.52)	(6.42)
5	26–50	25–175	1–25	12.414	0.013	0.316	0.227	0.083	0.780	0.780	1.981	2.109	2.185	1.768
					[4]	[2]	[4]	[3]	[0]	[0]	[0]	[0]	[0]	[0]
					(6.09)	(6.01)	(6.01)	(6.01)	(7.13)	(6.50)	(5.91)	(6.01)	(6.51)	(6.42)
6	26–50	75–175	1–25	31.235	0.000	0.000	0.000	0.422	0.248	0.248	0.989	1.052	1.052	1.194
					[5]	[5]	[5]	[1]	[3]	[3]	[1]	[1]	[1]	[0]
					(6.08)	(5.97)	(5.97)	(5.97)	(7.11)	(6.47)	(5.88)	(5.97)	(6.48)	(6.42)
7	1–25	0–200	1–25	35.529	0.000	0.000	0.146	0.000	1.644	1.644	1.447	1.072	2.434	1.977
					[5]	[5]	[4]	[5]	[1]	[1]	[0]	[0]	[0]	[0]
					(6.10)	(6.05)	(6.06)	(6.05)	(7.15)	(6.59)	(5.88)	(6.05)	(6.50)	(6.42)
8	1–25	50–150	1–25	38.976	0.000	0.000	0.043	0.000	0.064	0.064	1.435	1.631	1.749	2.094
					[5]	[5]	[4]	[5]	[4]	[4]	[0]	[0]	[0]	[0]
					(6.09)	(6.02)	(6.02)	(6.02)	(7.14)	6.55	(5.86)	(6.02)	(6.47)	(6.42)
9	1–25	25–175	1–25	39.278	0.000	0.000	0.000	0.000	0.285	0.285	1.583	1.782	1.492	1.600
					[5]	[5]	[5]	[5]	[3]	[3]	[0]	[0]	[0]	[0]
					(6.09)	(6.02)	(6.02)	(6.02)	(7.13)	(6.54)	(5.85)	(6.02)	(6.46)	(6.42)
10	1–25	75–175	1–25	43.647	0.000	0.000	0.000	0.000	0.000	0.000	1.110	1.110	1.187	2.040
					[5]	[5]	[5]	[5]	[5]	[5]	[1]	[1]	[1]	[0]
					(6.09)	(6.00)	(6.01)	(6.00)	(7.13)	(6.53)	(5.85)	(6.00)	(6.46)	(6.42)

$N = 15$	Relative deviation index (Avg time in CPU sec)									
1	standard Sa	standard TA	ATV SA	ATV TA	ANS SA ($S = 2$)	ANS TA ($S = 2$)	Des	Des ATV	Des ANS ($S = 2$)	Random
	0.789	0.231	0.988	0.549	0.157	0.006	0.253	0.263	0.435	1.000
	(28.29)	(26.69)	(28.59)	(28.58)	(32.48)	(31.08)	(28.50)	(28.29)	(28.32)	(32.14)
2	0.310	0.096	0.876	0.314	0.121	0.133	0.343	0.304	0.367	1.000
	(28.26)	(29.61)	(28.55)	(28.55)	(32.44)	(31.04)	(28.55)	(28.27)	(28.73)	(32.15)
3	1.000	0.293	1.000	0.587	0.105	0.114	0.489	0.485	0.517	1.000
	(28.00)	(29.18)	(28.26)	(28.26)	(32.24)	(30.58)	(28.52)	(28.00)	(28.02)	(32.15)
4	0.695	0.482	0.870	0.425	0.080	0.302	0.632	0.695	0.849	1.000
	(27.98)	(29.13)	(28.28)	(28.27)	(32.26)	(30.60)	(28.55)	(27.99)	(28.30)	(32.13)
5	0.808	0.421	1.000	0.611	0.062	0.473	0.471	0.472	0.492	1.000
	(27.96)	(29.15)	(28.25)	(28.25)	(32.23)	(30.61)	(28.53)	(27.96)	(28.02)	(32.12)
6	1.000	0.442	0.985	0.689	0.057	0.129	0.380	0.512	0.644	1.000
	(27.97)	(29.11)	(28.23)	(28.22)	(32.17)	(30.52)	(28.49)	(27.97)	(28.05)	(32.13)
7	0.986	0.165	0.997	0.764	0.305	0.124	0.576	0.661	0.625	1.000
	(28.25)	(29.55)	(28.55)	(28.54)	(32.48)	(31.06)	(28.41)	(28.25)	(28.07)	(32.12)
8	0.994	0.029	1.000	0.884	0.386	0.353	0.583	0.530	0.651	1.000
	(28.24)	(29.53)	(28.55)	(28.54)	(32.46)	(31.05)	(28.43)	(28.25)	(28.35)	(32.11)
9	0.986	0.355	1.000	0.825	0.291	0.057	0.435	0.548	0.784	1.000
	(28.22)	(29.50)	(28.52)	(28.51)	(32.39)	(30.92)	(28.41)	(28.22)	(28.34)	(32.14)
10	1.000	0.096	0.998	0.929	0.560	0.481	0.521	0.506	0.691	1.000
	(28.14)	(29.38)	(28.43)	(28.42)	(32.36)	(30.92)	(28.39)	(28.15)	(28.33)	(32.14)

$N = 30$	Relative deviation index (Avg time in CPU sec)											
	standard SA	standard TA	ATV SA	ATV TA	ANS SA ($S = 3$)	ANS SA ($S = 4$)	ANS TA ($S = 3$)	ANS TA ($S = 4$)	Des	Des ATV	Des ANS ($S = 4$)	Random
1	1.000	0.870	1.000	0.876	0.616	0.746	0.605	0.739	0.031	0.012	0.174	1.000
	(146.39)	(157.95)	(146.44)	(146.41)	(167.35)	(168.22)	(164.66)	(165.32)	(149.68)	(146.40)	(158.85)	(170.79)
2	1.000	0.513	1.000	0.951	0.370	0.534	0.385	0.557	0.058	0.007	0.323	1.000
	(146.16)	(157.52)	(146.22)	(146.18)	(167.10)	(168.38)	(164.33)	(165.77)	(149.84)	(146.17)	(159.39)	(171.02)
3	1.000	1.000	1.000	1.000	1.000	0.983	0.286	0.276	0.465	0.465	0.465	1.000
	(143.68)	(154.39)	(143.72)	(143.68)	(165.31)	(165.76)	(161.30)	(161.99)	(149.99)	(143.68)	(158.53)	(171.16)
4	1.000	0.942	1.000	1.000	0.885	0.832	0.610	0.035	0.407	0.380	0.712	1.000
	(143.75)	(154.58)	(143.78)	(143.77)	(165.88)	(166.46)	(161.83)	(163.02)	(151.56)	(143.76)	(158.60)	(171.31)
5	1.000	1.000	1.000	1.000	1.000	1.000	0.440	0.319	0.427	0.390	0.423	1.000
	(143.65)	(154.32)	(143.68)	(143.66)	(165.23)	(165.88)	(161.36)	(162.38)	(151.64)	(143.67)	(158.36)	(171.25)
6	1.000	0.945	1.000	1.000	1.000	0.913	0.374	0.019	0.655	0.670	0.655	1.000
	(143.53)	(154.12)	(143.57)	(143.54)	(164.56)	(165.35)	(160.63)	(161.39)	(151.65)	(143.54)	(158.06)	(171.24)
7	1.000	1.000	1.000	1.000	1.000	1.000	0.996	0.998	0.022	0.002	0.138	1.000
	(146.02)	(156.96)	(146.07)	(146.02)	(166.77)	(167.61)	(163.77)	(164.80)	(151.41)	(146.03)	(157.91)	(171.05)
8	1.000	1.000	1.000	1.000	0.969	1.000	1.000	1.000	0.038	0.011	0.229	1.000
	(146.26)	(157.17)	(146.31)	(146.27)	(166.79)	(167.32)	(163.84)	(164.43)	(151.27)	(146.27)	(158.35)	(171.06)
9	1.000	1.000	1.000	1.000	1.000	0.905	1.000	0.924	0.068	0.110	0.026	1.000
	(146.05)	(156.94)	(146.08)	(146.05)	(166.66)	(167.08)	(163.84)	(164.09)	(151.15)	(146.06)	(157.95)	(171.06)
10	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.164	0.017	0.010	1.000
	(145.71)	(156.28)	(145.74)	(145.71)	(166.09)	(166.58)	(163.00)	(163.54)	(150.81)	(145.71)	(157.58)	(171.13)

144 CPU seconds respectively. (These are set by a complex strategy in Section 6.4.5 of Lin [12] for making fair comparison with other methods.)

Standard methods. For each T_i, $2N$(= total number of phase a and phase b tasks) permutations are carried out. With the given time limit, the corresponding sequence length for $\{T_i\}$ from experimentation are (9000, 8500, 12000) for N = (6, 15, 30) jobs respectively.

ANS. All the $2N$ tasks are partitioned by their positions. (Partitioning by natural ordering is difficult since the solution involves precedence-related tasks.) The number of subgroups (S) for each problem size is chosen such that the subgroup size is no less than 5. The values of S which are most effective are given in the computations (Section 9.3). The sequence of $\{T_i\}$ is the same as that in the standard methods. For each T_i, S subgroups are selected and $2N/S$ permutations are carried out in each chosen subgroup.

ATV. The running time of standard TA is used as the reference basis. The standard method is allowed to run for the first half of the time before T_i becomes adaptive. For each T_i, T_L and T_S, $2N$ iterations are performed as in the standard and ANS methods. A moderate value of 2 is chosen for the adjusting factor α.

9.3. Computational results

For each problem size of 6, 15 and 30 jobs, 50 instances are randomly generated. Initially for $N = 6$ jobs, the instances with various ranges of (p^a, d, p^b) are chosen such that the average % idle : processing time in the optimal sequences varies between 0–45%. These ranges of (p^a, d, p^b) are then used to generate random instances for $N = 15$ and 30 jobs. 10 sets of (p^a, d, p^b) are used and 5 instances are generated in each, giving 50 instances for each problem size.

Table 4 gives the performance comparison of all methods in terms of solution quality and average computational time. For small problem size of $N = 6$ jobs, the average absolute performance can be found by a branch-and-bound method. As problem size increases, the relative deviation index (R) (as in Section 8.3) is used to measure the relative performance of the methods. From Table 4, it was observed that there is no winner in every case. The random sequences are obviously inferior for large problem size, but are comparable with the descent methods, giving the worst performance in small problem size (6 jobs). The findings from Table 4 includes the following.

- The standard methods are preferred over other sophisticated methods for small problem size, where standard SA and TA have similar performance.
- Either the standard TA or the ANS methods outperform the other methods in $N = 15$ jobs.
- For the largest problem size of 30 jobs, the winner is either the ANS methods or the descent methods. (This implies that the partitioning concept and descent approaches become more effective for larger problem size.)
- Between the SA and TA versions of each method (standard, ANS, ATV), TA has better or similar performance as SA.
- The descent methods are more likely to be trapped in local minima in smaller problem size, with improving performance as problem size increases. Among the three different descent methods, Des ANS is on average inferior than Des and Des ATV, which are indifferent between themselves. (The inferior performance of Des ANS may again be due to the fixed job partitioning which restrains the movement of tasks.)

10. Conclusion

Local search techniques of threshold accepting (TA) and simulated annealing (SA) are compared in this paper for three problems of scheduling jobs with arbitrary time lags. Standard and two adaptive versions of SA and TA are proposed. The adaptive neighbourhood search (ANS) is based on finding dominant neighbourhoods in the solution to allocate future search effort. The adaptive temperature/threshold values (ATV) adjusts the acceptance criteria according to recent search performance. Given good starting solutions, the descent methods (on the standard and adaptive versions of TA) provide efficient tools directly, but could be trapped in local min-

ima in small problem size (problem 3). The standard methods are still effective techniques for problems having few precedence-related tasks in the solution representation (problem 1), or for small problem size of those problems with precedence relationships among tasks becoming significant (problem 3). ANS and ATV can improve on the standard methods, especially in the harder problems, like the mean completion time problems and in the environment of precedence-related tasks. Such improvement is more significant as problem size increases. Between TA and SA in the different versions, TA achieves better or comparable performance with SA. In particular, adjusting the threshold values in ATV TA is more effective than adjusting the temperatures in ATV SA.

A natural extension of this work is to apply the methods here to harder problems with benchmark solutions. A comparative study of threshold accepting, simulated annealing and other well-known local search methods (possbily adopting the adaptive concepts) is worthy of investigation. Implementation of the adaptive versions in other environment of precedence-related tasks and improvement of the methods here with incorporation of problem-specific information generate interests for further study.

References

[1] Aarts, E.H.L., van Laarhoven, P.J.M., Lenstra, J.K., and Ulder, N.L.J., "Job shop scheduling by local search", Memorandum COSOR 92-29, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, Netherlands, 1992.

[2] Althofer, I., and Koschnick, K.U., "On the convergence of threshold accepting", *Journal of Applied Mathematics and Optimisation* 24 (1991) 183–195.

[3] Chen, W.-H., and Srivastava, B., "Simulated annealing procedures for forming machine cells in group technology", *European Journal of Operational Research* 75/1 (1994) 100–111.

[4] Dessouky, M.I., and Deogun, J.S., "Sequencing jobs with unequal ready times to minimize mean flow time", *SIAM Journal of Computing* 10/1 (1981) 192–202.

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

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

[7] Glass, C.A., Potts, C.N., and Shade, P., "Genetic algorithms and neighbourhood search for scheduling unrelated parallel machines", Technical Report No. OR47, Faculty of Mathematical Studies, University of Southampton, UK, 1992.

[8] Jeffcoat, D.E., and Bulfin, R.L., "Simulated annealing for resource-constrained scheduling", *European Journal of Operational Research* 70/1 (1993) 43–51.

[9] Johnson, D.S., Aragon, C.R., McGeoch, L.A., and Schevon, C., "Optimization by simulated annealing: Part I. graph partitioning", *Operations Research* 37/6 (1989) 865–892.

[10] van Laarhoven, P.J.M., and Aarts, E.H.L., *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Co., Holland.

[11] Lin, C.K.Y., and Haley, K.B., "Scheduling two-phase jobs with arbitrary time lags in a single-server system", *IMA Journal of Mathematics Applied in Medicine & Biology*, to appear.

[12] Lin, C.K.Y., *Scheduling two-phase jobs with arbitrary time lags in a single-server system*, Phd thesis, University of Birmingham, Birmingham, UK (submitted).

[13] Lundy, M., and Mees, A., "Convergence of an annealing algorithm", *Math Prog.* 34 (1986) 111–124.

[14] Matsuo, H., Chang, J.S., and Sullivan, R.S., "A controlled search simulated annealing method for the single machine weighted tardiness problem", *Annals of Operations Research* 21 (1989) 85–108.

[15] Ogbu, F.A., and Smith, D.K., "The application of the simulated annealing algorithm to the solution of the $n/m/C_{\max}$ flowshop problem", *Computers & and Operations Research* 17/3 (1990) 243–253.

[16] Osman, I.H., and Potts, C.N., "Simulated annealing for permutation flowshop scheduling", *Omega* 17 (1989) 551–557.

[17] Reeves, C.R., "Improving the efficiency of tabu search for machine sequencing problems", *Journal of the Operational Research Society* 44/4 (1993) 375–382.

[18] Sampson, S.E., and Weiss, E.N., "Local search techniques for the generalized resource constrained project scheduling problem", *Naval Research Logistics* 40 (1993) 665–675.

[19] Vakharia, A.J., and Chang, Y.-L., "A simulated annealing approach to scheduling a manufacturing cell", *Naval Research Logistics* 37 (1990) 559–577.

[20] Venugopal, V., and Narendran, T.T., "Cell formation in manufacturing systems through simulated annealing: an experimental evaluation", *European Journal of Operational Research* 63/3 (1992) 409–422.

[21] Wright, M.B., "Applying stochastic algorithms to a locomotive scheduling problem", *Journal of the Operational Research Society* 40 (1989) 187–192.