ORIGINAL ARTICLE

Ghaith Rabadi · Georgios C. Anagnostopoulos ·
Mansooreh Mollaghasemi

A heuristic algorithm for the just-in-time single machine scheduling problem with setups: a comparison with simulated annealing

Abstract This paper addresses the single machine early/tardy problem with unrestricted common due date and sequence-dependent setup times. Two algorithms are introduced to reach near-optimum solutions: the SAPT, a heuristic tailored for the problem, and a simulated annealing (SA) algorithm. It will be shown that SA provides solutions with slightly better quality; however, SAPT requires much less computational time. SAPT-SA is a hybrid heuristic that combines both approaches to obtain high quality solutions with low computational cost. Solutions provided by the three algorithms were compared to optimal solutions for problems with up to 25 jobs and to each other for larger problems.

Keywords Simulated annealing · Scheduling · Single machine · Setup time · Early-tardy problem · Common due date

G. Rabadi (✉)
Engineering Management and Systems Engineering
Department, Old Dominion University,
241 Kaufman Hall,
Norfolk, VA 23529, USA
e-mail: grabadi@odu.edu
Tel.: +1-757-6834918
Fax: +1-757-6835640

G. C. Anagnostopoulos
Department of Electrical & Computer Engineering,
Florida Institute of Technology,
Melbourne, FL 32901-6975, USA
e-mail: georgio@fit.edu

M. Mollaghasemi
Industrial Engineering and Management Systems Department,
University of Central Florida,
Orlando, FL 32816, USA
e-mail: mollagha@mail.ucf.edu

The problem addressed in this paper is to schedule a set of n jobs, $N=\{1,\ldots, n\}$, with a large common due date d for all jobs on a single machine such that the total earliness and tardiness from d is minimized. All jobs are assumed to be available at time zero, and the machine can process only one job at a time without preemption. The time it takes for job j to be processed is P_j. However, it is assumed that before job j can be processed, the machine requires a setup time S_{ij} whose value depends whether job i precedes j or vice versa. This is known as sequence-dependent setup time and is represented by a matrix $[S]$, in which, in general $S_{ij} \neq S_{ji}$. This problem is referred to as the early/tardy (E/T) problem, where the earliness of job j can be defined as $E_j = \max(0, d - C_j)$ and its tardiness as $T_j = \max(0, C_j - d)$. Here, C_j is the completion time of job j. The objective is then to minimize

$$H = \sum_{j=1}^{n} \left(E_j + T_j\right) = \sum_{j=1}^{n} \left|d - C_j\right| \tag{1}$$

Note that C_j depends on both P_j and S_{ij}, and this makes it similar to the traveling salesman problem, which is NP-hard and obtaining an optimal solution requires a number of computations that depends exponentially on n. The objective of this problem is consistent with the principles of the just-in-time manufacturing philosophy, where jobs are desired to be completed as close as possible to their due dates to avoid holding cost (due to earliness) and penalties for missing the due date (due to tardiness). In this paper, we first introduce a new fast heuristic and a simulated annealing (SA) algorithm to obtain near-optimum solutions for the problem. Secondly, these two approaches are combined into a composite hybrid heuristic and then all three algorithms are compared experimentally. This paper is organized as follows. In Section 1 the literature related to the problem is reviewed, while in Section 2 the properties of the problem are presented. The SAPT heuristic and its experiments are introduced in Sections 3 and 4, respectively, while the SA and its experiments are introduced in Sections 5 and 6, respectively. The hybrid algorithm is presented in Section 7.

SAPT and SA when applied to larger problems are compared in Section 8. Finally, conclusions are summarized in Section 9.

1 Literature review

The single-machine E/T problem was first introduced by Kanet [1], and Baker and Scudder [2] published a comprehensive state-of-the-art review for different variants of the E/T problem. Kanet examined the E/T problem with equal penalties and unrestricted common due date. A problem is considered unrestricted when the due date is large enough not to constrain the scheduling decision. Based on certain optimality properties that will be discussed later, Kanet introduced a polynomial-time algorithm to solve the problem optimally. Since then many researchers worked on various extensions of the problem and investigated optimality conditions (e.g., Hall [3]; Bagchi et al. [4]; Szwarc [5]; Hall and Posner [6]; Alidaee and Dragan [7]; Azizoglu and Webster [8]; Mondal and Sen [9]). Other researchers introduced approximate algorithms for different versions of the problem to solve for a larger number of jobs (e.g., Sundararaghavan and Ahmed [10]; Liman and Lee [11]; Lee and Choi [12]; Sirdharan and Zhou [13]; James [14] and Webster et al. [15]). Allahverdi et al. [16] reviewed the scheduling literature that involved setup times. In their review, few researchers have addressed the E/T problem with sequence-dependent setup times. Azizgolu and Webster [8] considered the problem with job family setup times and introduced a branch-and-bound and a beam search algorithm for the problem. Webster et al. [15] developed a genetic algorithm for the same problem with setup times. In both papers, the authors considered setup times that were not sequence dependent. Coleman [17] introduced a 0/1 mixed integer program (MIP) to find optimal solutions for the problem with sequence-dependent setup times. Due to the complexity of the problem, Coleman's MIP solves problems with a small number of jobs. Rabadi [18] developed a customized branch-and-bound (B&B) algorithm to solve the E/T problem with sequence-dependent setup times for problems up to 25 jobs. When the problem becomes larger than that, obtaining optimal solutions becomes very time consuming. Hence, researchers tend to use heuristic and approximate algorithms to find near-optimum solutions.

Simulated annealing (SA) is a stochastic meta-heuristic for global combinatorial optimization borrowed from statistical mechanics (Metropolis et al. [19]; Kirkpatrick et al. [20]). SA has found wide acceptance as an efficient computational method designed to solve hard combinatorial optimization problems. In the recent past, significant research has been directed towards the application of SA onto scheduling problems. Only a small fraction of this research utilizes SA in the framework of single-machine problems with sequence-dependent setup times. Shapiro and Alfa [21] used SA to minimize the sum of linear sequence-dependent setup costs and linear delay penalties; they concluded that SA matches the performance of tabu

search (TS) (Glover [22, 23]). Tan and Narasimhan [24] compared SA and random search (RS) to minimize the total tardiness. They reported that SA outperformed RS and was able to find good solutions quickly, which, in their opinion, makes SA a valuable approach even for on-line production scheduling. On the other hand, Sun et al. [25] presented a Lagrangian relaxation-based approach for minimizing the weighted sum of squared tardiness. After comparing their method with SA, TS and other heuristics, they concluded that it is capable of finding solutions comparable in quality to the ones reached via SA, however, in a more computationally efficient way than SA. Finally, Tan et al. [26] considered the total tardiness problem and showed that SA and random-start pair-wise interchange exhibit significant merit for large combinatorial problems that are computationally prohibitive to B&B approaches.

Apart from the applications of pure SA on single-machine problems, there are also attempts of using hybrid schemes that involve SA as one of their components. For example, Mittenthal et al. [27] suggested a greedy descent search followed by a SA stage to optimize single-machine problems featuring V-shaped optimal schedules. Note that E/T problems do not feature this type of optimal schedules. The role of the greedy descent search is to reduce the computational complexity of the pure SA approach by providing a preliminary solution, while the SA's role is to perform a more localized stochastic search in the locality of that solution. They eventually showed that the hybrid scheme outperformed a collection of heuristics previously presented in the literature. Adenso-Diaz [28] considered the total tardiness problem and used a dispatching rule, SA and TS in succession, thus saving computations prior to performing TS while maintaining high quality of solutions. Finally, Almeida and Centeno [29] addressed the single-machine E/T problem without setup times and proposed the use of a four-phase hybrid heuristic involving TS, SA, greedy descent search and RS. After comparing their hybrid algorithm to pure SA and pure TS, they concluded that the four-phase heuristic provides good solutions with low computational cost. To the best of our knowledge, the application of pure SA and/or of a hybrid optimization scheme involving SA has not been attempted so far on the single-machine E/T problem with sequence-dependent setup times and a large common due date.

2 The single machine early/tardy problem

Kanet [1] introduced a set of optimality properties for the problem addressed in this paper when there are no setup times. Using the following properties, Kanet introduced an optimal polynomial-time algorithm:

i) In an optimal schedule, there will be no idle time inserted.
ii) There is an optimal schedule in which one job completes exactly on the due date d.

iii) In an optimal schedule, the bth job in the sequence completes on d, where $b = n/2$ if n is even, and $(n+1)/2$ if n is odd.

iv) An optimal schedule is V-shaped. This means that the non-tardy jobs are sequenced in LPT order (longest processing time first) and the tardy jobs in SPT order (shortest processing time first).

When sequence-dependent setup times are included, the first three properties still hold, while property (iv) does not. The optimality of properties (i), (ii), and (iii) can be proven as in Kanet [1]. Noncompliance to property (iv) can easily be shown via simple counter-examples.

Instead of considering the setup matrix $[S]$ and the processing times vector P separately, we introduce the *adjusted processing time matrix [AP]* as follows:

$$\forall j = 1, \ldots, n:$$
$$AP_{ij} = S_{ij} + P_j, \quad \forall i = 1, \ldots, n \tag{2}$$

Utilizing *[AP]*, (see Fig. 1), the total earliness TE and total tardiness TT for a particular sequence become

$$TE = \sum_{j=1}^{b} (j-1)AP_{[j-1][j]} = 0AP_{[0][1]} + 1AP_{[1][2]}$$
$$+ 2AP_{[2][3]} + \cdots + (b-1)AP_{[b-1][b]} \tag{3a}$$

$$TT = \sum_{j=b}^{n-1} (n-j)AP_{[j][j+1]}$$
$$= 1AP_{[n-1][n]} + 2AP_{[n-2][n-1]} + \cdots$$
$$+ (n-b-1)AP_{[n-b-1][n-b]} + (n-b)AP_{[b][b+1]} \tag{3b}$$

Note that the objective function value in Eq. 1 is just the sum of total earliness and total tardiness, i.e., $H = TE + TT$. Since both TE and TT depend only on the sums $S_{ij} + P_j = AP_{ij}$ rather than explicitly on the S_{ij} and P_j quantities, from here on we will only utilize *[AP]*. A similar approach was used by Gendreau et al. [30] and Rabadi et al. [18].

Another observation is that the common due date d does not appear in Eqs. 3a,b; that is, the objective function value does not depend on d as long as it is large enough not to restrict the scheduling decision. An exact lower bound (Δ)

on how large d must be for the problem to be unrestricted can not be calculated beforehand due to the presence of sequence-dependency. Therefore an optimal solution has to be found first, the schedule has to be shifted so that it starts from time $t=0$, and then Δ is calculated by Eq. 4:

$$\Delta = \sum_{j=1}^{b} AP_{[j-1][j]} \tag{4}$$

If $d \geq \Delta$ then the obtained solution is optimal, otherwise if $d < \Delta$ the problem becomes restricted and optimality is not guaranteed.

3 The SAPT heuristic

In an optimal job sequence, jobs with shorter adjusted processing times AP_{ij} tend to be scheduled closer to the median position, and those with longer AP_{ij} values away from the median position. This is consistent with Eq. 3a,b where AP_{ij} values for jobs closer to the median of the schedule are multiplied by higher coefficients. The shortest adjusted processing time first (SAPT) heuristic is based on this concept. SAPT consists of two phases: (1) schedule construction, and (2) schedule Improvement. Phase 1 starts by selecting jobs $i*$ and $j*$ with the smallest entry in *[AP]* and placing them in positions $b-1$ and b, respectively. Another two jobs with the next smallest *[AP]* entry before i or after j are then selected and scheduled either after job j or before job i depending on which location gives a lower objective function value (see Eq. 3a, b). This selection process is repeated until all jobs are scheduled. It is important to maintain feasibility throughout the scheduling process by eliminating the selected *[AP]* entries and their corresponding rows and columns to avoid scheduling conflicts. SAPT is a multi-start heuristic where every time it is executed, it is repeated n times for the nth smallest entries in *[AP]*. The rationale behind restarting with n smallest jobs in the median position is that the optimal schedule will most likely include in its median position a job corresponding to a small entry of *[AP]*, but not necessarily the smallest one.

In Phase 2, the general pairwise interchange (GPI), a neighborhood search method is used to improve the schedule. In GPI, any two jobs (not just adjacent ones) may be swapped starting by swapping the job in the first position with the succeeding jobs one at a time until the nth

Fig. 1 A generic job sequence with adjusted processing time notation
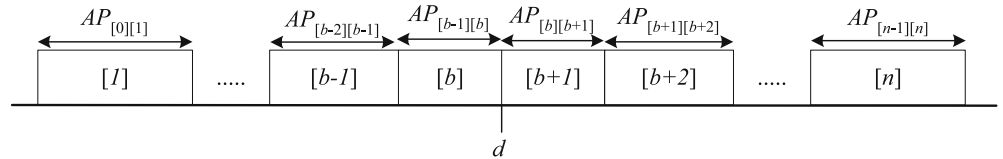
[j]: Job in position j
$AP[j][j+1]$: Adjusted Processing Time for the job in position j followed by the job in position [$j+1$]
[b]: The median position

position. Then, it continues by swapping the job in the second position with the succeeding jobs until the nth position, and so on until the last two jobs in the sequence are swapped for improvement. The neighborhood is generated by every possible pairwise interchange. Hence, for n jobs, the neighborhood would consist of $n(n-1)/2$ sequences. If the swapping of any two jobs improves the schedule, they are left in their current position and GPI continues with the next job. Other neighborhood search methods exist such as adjacent pairwise interchange and job insertion (see Morton and Pentico [31]). After empirically testing several neighborhood search methods, the GPI gave the best improvements and therefore it was used with the SAPT heuristic. Pseudo code for SAPT can be described as follows:

A: tardy jobs array=$\{j \in N | C_j > d\}$; and $|A|$ is the number of elements in array A

B: non-tardy jobs array=$N - A$; and $|B|$ is the number of elements in array $|B|$

If n is *even*, then $b := n/2$; Else $b := (n+1)/2$
If $AP_{i\,j} = -1$, then i and j have been scheduled
$Z_{\text{best}} := \infty$

Procedure SAPT( )
For $m = 1$ to nth smallest entries in $[AP]$ do
Set $A := \varnothing$, $B := \varnothing$
Find i^*, $j^* \in \{1, 2,\ldots, n\}$ with minimum AP_{ij} subject to the constraints $i \neq j$, $AP_{i\,j} \neq 1$
Set $AP_{i^*\,j^*} := -1$
Set $B[b] := j^*$, $B[b-1] := i^*$
For $p = 1$ to n do
Set $AP_{i^*p} := -1$, $AP_{p,j^*} := -1$
end For
Set $AP_{j^*i^*} := -1$
Swap values of i^* and j^*
Set $k := 2$, $u := b$
While $|B| < b$ or $|A| < n-b$ do
Find $i" \in \{1, 2,\ldots, n\}$ such that AP_{ij^*} subject to the constraints $i \neq j$, $AP_{i\,j} \neq -1$
Find $j" \in \{1, 2,\ldots, n\}$ such that AP_{i^*j} subject to the constraints $i \neq j$, $AP_{i\,j} \neq -1$
If $AP_{ij^*} \times (b - k) \leq AP_{i^*j} \times u$ then
Set $B_{b-k} := i"$
For $p = 1$ to n do
Set $AP_{i"p} := -1$, $AP_{p,j^*} := -1$
end For
Set $j^* := i"$
Set $k := k + 1$
else
Set $A_{n-u+1} := j$
For $p = 1$ to n do
Set $AP_{i\,j"} := -1$, $AP_{i^*\,j} := -1$
end For
Set $i^* := j"$
Set $u := u + 1$
end If
end While

Calculate the objective function value H of schedule described by A and B.
If $H < H_{\text{best}}$ then
Set $A_{\text{best}} := A$, $B_{\text{best}} := B$, $H_{\text{best}} = H$
End If
End For
Perform general pair-wise interchange for the sequence that is described in A_{best} and B_{best}.
End Procedure SAPT( )

It can be easily shown that SAPT's time complexity is $O(n^2)$, which explains its computational speed.

4 SAPT computational experience

A full factorial design of experiment was conducted to test the performance of SAPT. The factors considered in the experiments are the number of jobs n and the range R for $[AP]$. For the number of jobs we considered four different levels: 10, 15, 20 and 25. If $unif(a,b)$ denotes the uniform distribution over the interval a to b, in our experiments the entries of $[AP]$ were independently drawn from $unif$ $(10,10+R)$. Furthermore, we considered three levels for R: 50, 100 and 150. Changing the range helps in observing the behavior of the heuristic algorithm when the mean and variance of $[AP]$ change. The number of treatments (experiment settings) were $4 \times 3 = 12$. Each treatment was replicated 15 times by generating 15 different problem instances for each treatment, that is, 180 problems were solved. All experiments were run on a 1.7 GHz Pentium IV processor running Microsoft Windows 2000. To evaluate the performance of SAPT, optimal solutions for the same set of problems were obtained using a B&B developed by Rabadi [18]. The relative errors of SAPT with respect to the optimal solutions were recorded and the results are summarized in Table 1. Both data and results are available at Scheduling Research [32].

Table 1 Experiment results for the SAPT heuristic

Number of jobs n	Range R	Min. relative error (%)	Avg. relative error (%)	Max. relative error (%)	Avg. of avg. relative error (%)
10	Low	0	2.55	11.66	
	Med	0	2.80	12.38	
	High	0	2.80	17.52	2.72
15	Low	0	3.59	8.06	
	Med	0	5.33	12.91	
	High	0	8.22	18.67	5.71
20	Low	0	3.13	5.30	
	Med	0	3.93	12.74	
	High	0	5.95	13.70	4.34
25	Low	1.01	3.21	6.56	
	Med	2.55	5.96	9.37	
	High	1.39	6.83	11.42	5.33
Total Avg.					4.53

As can be seen from Table 1, the overall average relative error is 4.73% and the worst average relative error was 5.71%. The ANOVA for SAPT depicted in Table 2 shows that both n and R have a significant effect on the performance of SAPT (low p-values) while their interaction is insignificant. As n is increased, the number of possible schedules grows exponentially, and therefore, it is expected for the relative error to become larger since the heuristic will have to choose among a much larger number of schedules. As R is increased the effect of sequence-dependency becomes more significant. Consider for example an extreme case of a very small range for *[AP]*. In this case, the significance of the sequence-dependency diminishes because all entries will almost be the same, and so, regardless of what job is sequenced after the other, the optimal solution would not differ much. In fact, in such a case, a heuristic would be of little importance since a good, or perhaps optimal solution, can be obtained by applying Kanet's algorithm directly. As R becomes larger, the problem instances become more difficult, and the importance of a good heuristic becomes more vital.

5 Application of simulated annealing to the E/T problem

SA performs a random walk in the configuration space of combinatorial problems. The transitions from one configuration to another are stochastic and depend on the difference in objective function value between these two configurations as well as on the current "temperature" of the algorithm. The rule that specifies how this temperature is being decreased over the runtime of SA is called *cooling schedule*. Under some general provisions, SA converges to at least a sub-optimal solution of the combinatorial problem.

In our framework, the combinatorial search space S is the set of all possible schedules for a pre-defined number of jobs n. Moreover, each *configuration (state)* s is represented by an array containing the order in which jobs occur in the corresponding schedule. We also define *energy* H_s to be the objective function value associated with schedule s. In our implementation of SA we considered two types of *moves* (transitions) from one schedule to another: (1) pairwise job exchange and (2) single job insertion. According to the first type, the transition is performed by randomly selecting any pair of distinct jobs within the schedule and exchanging their position. The single job insertion, on the other hand, entails selecting at random a job within a particular schedule and randomly inserting it in another location in the schedule while maintaining the relative order of the remaining jobs in that schedule. It can

Table 2 Effect test for the factors in the SAPT experiment

Source	DF	Sum of squares	F ratio	Prob>F (or p-value)
N	3	155.81027	4.1017	0.0077
R	2	299.65651	11.8327	<0.0001
$n \times R$	6	51.98034	0.6842	0.6626

be easily shown that the number of neighbors in S for the pairwise job exchange is $n(n-1)/2$, while for the single job insertion the number of neighbors in S is $n(n-1)$. In other words, the number of neighbors in S per schedule is of order $O(n^2)$ for both types of transition. A move from s to s' is termed as *uphill*, when the resulting schedule s' has higher energy than before (i.e., $H_s < H_s'$). In the opposite case, it is called a *downhill* move. We define as *trial* the attempt to move from one schedule to another via one of the two transition types. When a trial leads to a downhill move, the transition itself from s to s' is accepted (actually performed) and the trial is called *successful*. If a trial leads to an uphill move, it is going to be accepted with probability p_a according to the probability law given by Eq. 5:

$$p_a = e^{-\frac{\Delta H}{T_k}} \tag{5}$$

where $\Delta H = H_s' - H_s > 0$. Note that the lower the current temperature T_k, the smaller the probability of accepting uphill moves. Whenever a trial has been determined as successful, the current state of SA s is updated to s', while in the event of an unsuccessful trial, no state update takes place.

The SA algorithm itself consists of two loops: the outer or temperature step loop and the inner or trial loop. The outer loop implements the cooling schedule by iterating through k_{max} temperature steps. By starting at a temperature $T_o > 0$ and following a geometrically-decaying cooling schedule, the temperature T_k at step k is given by Eq. 6:

$$T_k = aT_{k-1} \tag{6}$$

where $a \in (0,1)$ is the temperature decay rate. The inner loop's function is to perform trials, that is, to attempt transitions different from the current schedule. In our implementation, we consider the following three types of transitions: pairwise job exchange (type 1), single job insertion (type 2) and random selection between types 1 and 2 (type 3). The inner loop dictates a maximum of i_{max} trials to be performed for each temperature value T_k. If at a particular temperature the number of successful trials (total number of uphill and downhill transitions) reaches its maximum of $i_s \leq i_{max}$, it is deemed that the temperature is too high and SA behaves like RS. Therefore, under this condition the inner loop is exited and SA continues with the next outer loop step after decreasing its temperature. The stopping criterion we chose for the SA is the maximum number of consecutive stagnant outer loop iterations $k_s \leq k_{max}$. If during k_s consecutive outer loop iterations all i_{max} trials performed inside the inner loop fail, then it is assumed that the probability of obtaining a successful trial during the next outer loop iteration is extremely low and therefore SA has most likely converged. Note that exhausting all k_{max} outer loop iterations without satisfying the aforementioned stopping criterion earlier during runtime means that k_{max} was set too low and the SA algorithm just stopped rather than converged. After convergence the most

current state (schedule) of SA is being regarded as a (sub-) optimal solution to the problem at hand.

When applying SA to combinatorial optimization problems there are certain considerations with regard to its various operating settings that deserve attention. The quality of the final solution suggested by SA as well as the computational cost spent to reach this solution depend on the choice of T_o, a, k_{max}, i_{max}, k_s, i_s, the SA's initial state s_o and the transition mechanism. As a general rule, there is a trade-off between quality of solution and computational cost. Achieving a balance between those two factors remains the most important challenge of tuning the SA parameters. High values of T_o, a, i_{max}, i_s and k_s improve solution quality, while yielding computationally expensive runs of SA. The opposite is true for low values of these parameters. However, for a well-balanced choice of operating parameters, the dependency on s_o is extremely weak since SA will most likely venture towards better states during runtime. Finally, the type of transition to be used in SA is usually determined by evaluating its effectiveness via experimentation.

6 SA computational experience

The SA parameters discussed earlier may greatly affect its performance. In this section, the SA algorithm is fine-tuned to find good parameter settings that would lead to high quality solutions within reasonable computational time when applied to the E/T problem. Once these parameter settings (and their interactions, if any) are identified, they will be used when comparisons involving the SA are made in the forthcoming experiments in this paper. The SA parameter's notation and meaning are listed in Table 3 for more convenience.

Initial experiments were run for some of the SA parameters. It turned out that transition type 3 (see Section 5) produced the best results, and hence, it was used throughout all experiments. For the rest of the SA parameters, a full factorial experiment was designed with the levels in Table 4. Note that i_s was set to be 10% of i_{max} for both levels and k_{max} was set to 20 for all experiments. Problems with $n=10$ seemed to be easy for the SA. Optimal solutions are available for the E/T problem with up to 25 jobs [18]. Therefore, the low and high levels for n were selected to be 15 and 25, respectively. i_{max} was set to be a function of n^2 because both of the transition types used in this paper are $O(n^2)$ as was explained in Section 5.

Table 4 SA parameters and their levels

Factor	Low (L)	High (H)
N	15	25
R	$Unif{\sim}[10, 60]$	$Unif{\sim}[10, 110]$
a	0.85	0.99
i_{max}	n^2	$15n^2$

The performance of SA as a function of the initial temperature T_o is not only problem dependent, but also data dependent. More specifically, a "good" choice of T_o should be based on the distribution of energies (objective function values), so that (i) T_o is not too high, which would unnecessarily increase the computational cost of SA by performing excessive number of outer loop iterations before converging; and (ii) T_o is not too low, so that SA will not most likely converge to a local minimum of the search space. When comparing SA runs on E/T problems differing in the range R or the number of jobs n, choosing a fixed value for T_o might not be acceptable because R and n drastically influence the distribution of energies. Therefore, it is desirable to choose somehow a "good" setting of T_o for each instance separately. In the absence of concrete, formal methods for the specification of T_o we devised a rule-of-thumb, which we employed in our experiments. This rule specifies T_o as the temperature for which SA is going to transition from the smallest to the highest possible energy state with some extremely small, predetermined probability p_a. Since the extreme energy values are not known a priori, we approximate them using lower and upper bounds instead. Define $AP_{min} = \min_{i,ji \neq j}[AP]$ and $AP_{max} = \max_{i,ji \neq j}[AP]$. Based on Eq. 3a,b lower bounds on earliness, tardiness and maximum objective function value are given as:

$$Earliness_{max} = AP_{max}\sum_{j=1}^{b}(j-1)$$
$$Tardiness_{max} = AP_{max}\sum_{j=b}^{n-1}(n-j) \tag{7}$$
$$H_{max} = Earliness_{max} + Tardiness_{max}$$

From Eq. 7 we derive that

$$H_{max} = \frac{AP_{max}}{2}[(n-b+1)(n-b)+b(b-1)] \tag{8}$$

Table 3 Summary of symbols and notation

T_o	Initial temperature for SA
a	Temperature decrease rate for SA cooling schedule
k_{max}	Maximum iterations of SA's outer loop; maximum number of temperature steps
k_s	Maximum number of consecutive, stagnant outer loop iterations (used as SA's convergence criterion)
i_{max}	Maximum iterations of SA's inner loop; maximum number of trials
i_s	Maximum successful trials after which the SA's inner loop will exit and the temperature will be decreased
s_o	Initial schedule for SA
tt-1, tt-2, tt-3	SA transition (move) types 1, 2 and 3

The lower bound $H_{\min}$ for the minimum objective function value is also of the form displayed in Eq. 8 with $AP_{\max}$ replaced by $AP_{\min}$. Therefore the upper bound for the difference of maximum and minimum energies is given as

$$\Delta H_{\max} = H_{\max} - H_{\min}$$
$$= \frac{R_{AP}}{2}[(n-b+1)(n-b) + b(b-1)] \quad (9)$$
$$\text{where } R_{AP} = AP_{\max} - AP_{\min}$$

Assume that at the very beginning of a SA run we want an uphill move between states differing in energy by $\Delta H_{\max}$ to occur with a predetermined probability p_a. The temperature T_o that will allow such a transition can be found by combining Eqs. 5 and 9 and is given as:

$$T_o = -\frac{\Delta H_{\max}}{\ln p_a} \quad \Rightarrow \quad T_o = \frac{R_{AP}}{4\ln\left(\frac{1}{P_a}\right)} f(n)$$
$$(10)$$
$$\text{where } f(n) = \begin{cases} n^2 - 1 & \text{if } n \text{ odd} \\ n^2 & \text{if } n \text{ even} \end{cases}$$

For all problems that we have considered in our experiments we chose a very small value of p_a. For each problem, after finding the minimum and maximum entries of $[AP]$ to form the difference R_{AP}, the rule-of-thumb yields T_o via Eq. 10. To find T_o for the low and high levels of R in Table 4, a very small value of 10^{-67} for P_a was empirically selected; and by using Eq. 10: $T_o=100$ for $R=100$, and $T_o=50$ for $R=50$.

For a full factorial design with 2 levels and 4 factors, $2^4=16$ settings will be required. Five problems were randomly generated and solved for each setting (i.e., 80

Table 6 Estimated effects and coefficients for the relative error (coded units)

| Term	Effect	Coefficient	Std error coefficient	T	p-value
Constant		3.156	0.166	18.980	0.000
R	1.937	0.969	0.166	5.820	0.000
a	−3.687	−1.843	0.166	−11.090	0.000
$i_{\max}$	−3.389	−1.695	0.166	−10.190	0.000
n	0.605	0.303	0.166	1.820	0.073
$R \times a$	−1.117	−0.559	0.166	−3.360	0.001
$R \times i_{\max}$	−1.019	−0.509	0.166	−3.060	0.003
$R \times n$	0.365	0.182	0.166	1.100	0.277
$a \times i_{\max}$	1.638	0.819	0.166	4.930	0.000
$a \times n$	0.332	0.166	0.166	1.000	0.322
$n \times i_{\max}$	0.399	0.200	0.166	1.200	0.234

instances). SA was used to solve each problem 30 times, that is, it was run 2,400 times and the average relative error from optimal solutions and the CPU times were recorded. A summary is given in Table 5. Both data and results are available at Scheduling Research [32].

To measure the significance of the different parameters, ANOVA was carried out. Assuming that the 3rd and 4th order interactions are part of the residual, it can be seen that some of the second order interactions and the main effects are significant as shown in Table 6. In particular, $R \times a$, $R \times i_{\max}$, and $a \times i_{\max}$ show clear significance (low p-values).

To gain more insight into these interactions, they are inspected in Fig. 2. Since all of the significant second order interactions have similar patterns, they all can be explained in a similar fashion. Increasing a from low to high (from −1 to 1 in Fig. 2) has a significant effect on the relative error where a smaller error is obtained. However, by how much the relative error will be reduced depends on the level of R. From the slope of the lines for that particular interaction,

Table 5 SA experimental results

n	$i_{\min}$	a	R	Min. CPU (sec)	Avg. CPU (sec)	Max. CPU (sec)	Min. relative error (%)	Avg. relative error (%)	Max. relative error (%)
15	Low	Low	Low	0.0	0.61	13.0	0.0	5.43	14.34
15	Low	Low	High	0.0	0.00	0.0	0.0	10.57	33.11
15	Low	High	Low	0.0	0.69	13.0	0.0	1.17	4.37
15	Low	High	High	0.0	0.00	0.0	0.0	1.82	7.24
15	High	Low	Low	0.0	4.14	201.0	0.0	1.62	5.36
15	High	Low	High	0.0	0.00	0.0	0.0	1.82	8.88
15	High	High	Low	3.0	6.08	11.0	0.0	0.05	1.13
15	High	High	High	2.0	3.73	7.0	0.0	0.34	2.66
25	Low	Low	Low	0.00	2.58	41.00	0.00	4.95	13.17
25	Low	Low	High	0.00	0.27	40.00	0.00	9.10	27.52
25	Low	High	Low	0.00	0.53	2.00	0.00	1.95	5.73
25	Low	High	High	0.00	0.50	2.00	0.00	3.82	8.53
25	High	Low	Low	1.00	2.06	3.00	0.00	1.89	5.95
25	High	Low	High	1.00	21.92	605.00	0.00	4.61	11.24
25	High	High	Low	13.00	20.65	35.00	0.00	0.45	13.73
25	High	High	High	12.00	18.52	34.00	0.00	0.91	5.03
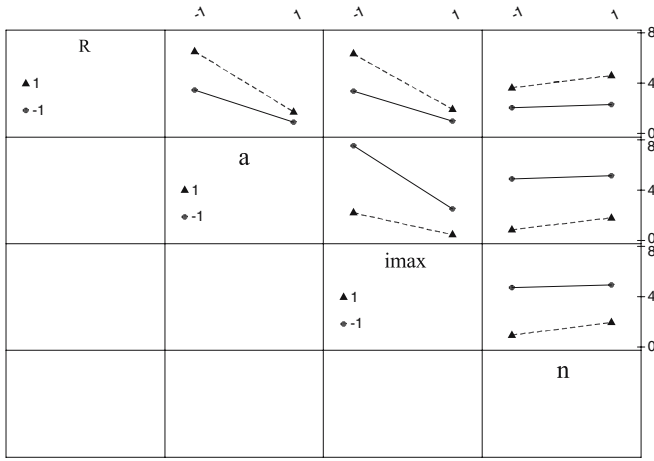
Fig. 2 Second order interactions between the SA parameters with respect to relative error

Table 8 SAPT-SA experimental results

n	R	Avg. CPU (s)	Avg. relative error (%)
15	Low	0.00	1.95
25	Low	1.00	1.11
15	High	0.00	1.87
25	High	1.00	2.53
Total avg.		0.5	1.86

one can see that when R is high the effect of a is more significant. We can conclude that increasing a leads to a better (smaller) relative error. The same reasoning can be applied to the interaction between R and i_{max}. A similar conclusion can be made for the interaction between i_{max} and a, whereas i_{max} is increased from low to high, the relative error decreases. However, depending on to what level a is set, the improvement varies.

Keeping the second order interactions in mind, the main effects can be explained as follows. As i_{max} and/or a are increased, a smaller relative error is expected regardless what value of R is. However, by how much the relative error is reduced depends on whether R is high or low. For high R, the average reduction in relative error is less. This result was expected; however, it is not intuitive to see the significance of R and the insignificance of n (see Table 6). Similar to the case of SAPT, as R decreases, the effect of sequence-dependency decreases. When R goes to zero, the problem reduces to the E/T problem with no setup time, which is a much easier problem. Consequently, one should expect that as R increases, the difficulty of finding an optimal sequence increases. With respect to n, it was

expected for the SA solution to deteriorate as n was increased; however, this was not exactly the case in our experiments where n did not show much significance. This is due to the levels selected for n, i.e., n will show more significance if it is increased to a higher number of jobs, say 50, as will be shown in Section 8.

To measure the computational efficiency of the SA, the CPU times needed for it to converge were recorded, and ANOVA was performed on the average CPU (Table 7).

The only significant second order interaction with respect to the average CPU is $i_{max} \times n$. This is also expected because i_{max} was set as a function of n^2, and the CPU increases significantly when i_{max} is set to a high level of n. Regardless, the average CPU in the worst case of large n and high i_{max} was on average about 20 seconds, which is acceptable. Therefore, the i_{max} will be set to high. As for the main effects on the average CPU, it is obvious that as n and/or i_{max} are increased, the CPU needed for the SA to converge increases. Note that n did not have a significant impact on the quality of the solution, but it had a significant impact on the CPU.

7 SAPT-SA: a hybrid algorithm

The SAPT-SA is a hybrid algorithm based upon the collaboration of the SAPT heuristic and the SA. In SAPT-SA, SAPT finds a solution to the problem and passes it to the SA as an initial schedule s_o. This way, SA starts with a good solution instead of starting with a random one. To test the effectiveness of the SAPT-SA, it is compared to the plain SA. To make the comparison fair, the parameters for SA in SAPT-SA were set to their best values obtained in Section 6. That is, $a=0.99$, $i_{max}=15n^2$, $i_s=10\%$ of i_{max}, and $k_{max}=20$. T_o was set to a very small value because, and based on the discussion in Section 5, the SA's initial solution will be lost if the initial temperature is high since many uphill transitions will be accepted at high temperatures. To measure the effect of the SAPT initial schedule on

Table 7 Estimated effects for the SA results with respect to the average CPU

Term	Effect	Coefficient	Std error coefficient	T	p-value
Constant		5.143	1.291	3.99	0.000
R	0.949	0.475	1.291	0.37	0.714
a	2.392	1.196	1.291	0.93	0.357
i_{max}	8.990	4.495	1.291	3.48	0.001
n	6.472	3.236	1.291	2.51	0.015
R×a	−2.250	−1.125	1.291	−0.87	0.386
R×i_{max}	1.861	0.931	1.291	0.72	0.473
R×n	2.896	1.448	1.291	1.12	0.266
a×i_{max}	2.824	1.412	1.291	1.09	0.278
a×n	0.953	0.476	1.291	0.37	0.713
n×i_{max}	5.827	2.914	1.291	2.26	0.027

Table 9 Plain SA experimental results

n	R	Avg. CPU (s)	Avg. Relative Error (%)
15	Low	6.08	0.21
25	Low	20.65	0.81
15	High	3.73	0.27
25	High	18.52	1.34
Total avg.		12.16	0.66

Table 10 Results of the SA and SAPT for larger problems

n	R	SA Avg. CPU (sec)	SAPT Avg. CPU (sec)	Avg. % of SAPT above SA
40	Low	22.82	0	1.71
	High	18.17	0	2.27
50	Low	24.01	0	0.79
	High	24.02	0	1.43
Total avg.		22.23	0	1.55

the performance of SAPT-SA, T_o was set to 0.1, which helped avoiding uphill transitions of SA. This way, the SA in SAPT-SA works as a refining method for the schedule provided by SAPT. R and n and their levels were kept the same as in Table 4. The same five problem instances generated in Section 6 are used here. Each instance was solved 30 times using the SAPT-SA, i.e., a total of 1,200 SA runs. The results obtained are shown in Table 8. Also, the results of the plain SA obtained in Section 6 are further summarized in Table 9.

By comparing the results above, one can see that the plain SA was slightly better than SAPT-SA (the best improvement at all levels was 1.74%); however, it is obvious that the average CPU time for SAPT-SA is much less. When comparing the performance of SAPT (Table 1) to SAPT-SA (Table 8), SAPT-SA reached better results with almost the same average CPU time. In conclusion, SAPT-SA provides a very good solution quality in a short CPU time when compared to SAPT by itself or SA by itself for problems with size of 25 jobs or less. Both data and results are available at Scheduling Research [32].

8 Comparison between SAPT and SA for larger problems

In all of the previous experiments, the size of the problem was limited to a maximum of 25 jobs as optimal solutions are not available for larger problems. It is more practical, however, to compare both approaches when the number of jobs is larger than 25. In this section, two additional problem sizes of 40 and 50 jobs were considered, where 15 problems for each size were randomly generated and solved by SA and SAPT independently. The SA parameters were specified to their best settings of a=0.99, i_{max}=15n^2, i_s=10% of i_{max}, k_{max}=20 and T_o=100 when R=100 and T_o=50 when R=50. The SA was run 30 times per instance, i.e., 900 SA runs on the same computer used with the previous experiments. The objective function values and CPU times were recorded. The averages of both measures are reported in Table 10. Since no optimal solutions exist for these problem sizes, the quality of the solution was based on relative difference between the averages of the objective function values H for both algorithms; that is, $(H_{SAPT}-H_{SA})100\%/H_{SA}$. It can be seen from Table 10 that SA was slightly better than SAPT with a maximum average

of 2.27%. However, SAPT outperformed SA in average CPU times.

9 Conclusions

The single machine E/T problem with a large common due date and sequence-dependent setup time was addressed. Since this problem is a difficult combinatorial problem, optimal solutions for problems with more than 25 jobs do not exist. In this paper, two new algorithms were developed to find near-optimum solutions for larger problems. The first is the SAPT heuristic, which consists of two phases: a schedule construction phase and a local neighborhood search phase, and its average relative error from the optimum was 4.5% for problems with 25 jobs or less. The second is a simulated annealing (SA) algorithm, which was applied to the problem for the first time when sequence-dependent setup times are included. For problems with 25 jobs or less, the average relative error from the optimum obtained by SA was 0.7%. SAPT, however, was significantly faster than SA. A hybrid algorithm, SAPT-SA, which is based on both the SAPT and SA, was also introduced in which SAPT generates a solution that is then handed over to the SA algorithm to refine. SA starts at very low temperature and reaches an improved solution quickly. A computational experience showed that SAPT-SA reached solutions with quality very comparable to that reached by the plain SA, but with significantly less computational time. For larger problems up to 50 jobs, the SAPT and SA were compared and SA yielded solutions that were on average better than SAPT with 1.5% but needed significantly more computational time than SAPT.

References

1. Kanet JJ (1981) Minimizing the average deviation of job completion times about a common due date. Nav Res Logist 28:643–651
2. Baker KR, Scudder GD (1990) Sequencing with earliness and tardiness penalties: a review. Oper Res 38(1):22–36
3. Hall NG (1986) Single- and multiple-processor models for minimizing completion time variance. Nav Res Logist Q 33:49–54
4. Bagchi U, Sullivan R, Chang Y-L (1986) Minimizing mean absolute deviation of completion times about a common due date. Nav Res Logist Q 33:227–240
5. Szwarc W (1989) Single machine scheduling to minimize absolute deviation of completion times from a common due date. Nav Res Logist 36:663–673
6. Hall NG, Posner ME (1991) Earliness-tardiness scheduling problems, I: weighted deviation of completion times about a common due date. Oper Res 39(5):836–846
7. Alidaee B, Dragan I (1997) A note on minimizing the weighted sum of tardy and early completion penalties in a single machine: a case of small common due date. Eur J Oper Res 96:559–563
8. Azizoglu M, Webster S (1997) Scheduling job families about an unrestricted common due date on a single machine. Int J Prod Res 35(5):1321–1330

9. Mondal SA, Sen AK (2001) Single machine weighted earliness-tardiness penalty problem with a common due date. Comput Oper Res 28(7):649–669

10. Sundararaghavan P, Ahmed M (1984) Minimizing the sum of absolute lateness in single-machine and multimachine scheduling. Nav Res Logist Q 31:325–333

11. Liman SD, Lee C-Y (1993) Error bound of a heuristic for the common due date scheduling problem. ORSA J Comput 5 (4):420–425

12. Lee C-Y, Choi JY (1995) A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. Comput Oper Res 22(8):857–869

13. Sirdharan V, Zhou Z (1996) A decision theory based scheduling procedure for single-machine weighted earliness and tardiness problem. Eur J Oper Res 94:292–301

14. James RJW (1997) Using tabu search to solve the common due date early/tardy machine scheduling problem. Comput Oper Res 24(3):199–208

15. Webster S, Jog D, Gupta A (1998) A genetic algorithm for scheduling job families on a single machine with arbitrary earliness/tardiness penalties and an unrestricted common due date. Int J Prod Res 36:2543–2551

16. Allahverdi A, Gupta JND, Aldowaisan T (1999) A review of scheduling research involving setup consideration. Omega 27 (2):219–239

17. Coleman BJ (1992) A simple model for optimizing the single machine early/tardy problem with sequence-dependent setups. Prod Oper Manag 1:225–228

18. Rabadi G (2004) A branch-and-bound algorithm for the early/tardy machine scheduling problem with a common due-date and sequence-dependent setup time. Comput Oper Res 31 (10):1727–1751

19. Metropolis N, Rosenbluth A, Rosenbluth M, Teller A, Teller E (1953) Equations of state calculations by fast computing machines. J Chem Phys 21(6):1087–1092

20. Kirkpatrick S, Gelatt CD Jr, Vecchi M (1983) Optimization by simulated annealing. Science 220:671–690

21. Shapiro JA, Alfa AS (1995) An experimental analysis of the simulated annealing algorithm for a single-machine scheduling problem. Eng Optim 24(2):79–100

22. Glover F (1989) Tabu search-part I. ORSA J Comput 1:190–206

23. Glover F (1990) Tabu search-part II. ORSA J Comput 2:4–32

24. Tan KC, Narasimhan R (1997) Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach. Omega 25(6):619–634

25. Sun XQ, Noble JS, Klein CM (1999) Single-machine scheduling with sequence dependent setup to minimize total weighted squared tardiness. IIE Trans 31(2):113–124

26. Tan KC, Narasimhan R, Rubin PA, Ragatz GL (2000) A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. Omega 28(3):313–326

27. Mittenthal J, Raghavachari M, Rana AI (1993) A hybrid simulated annealing approach for single-machine scheduling problems with non-regular penalty functions. Comput Oper Res 20(2):103–111

28. Adenso-Diaz B (1996) An SA/TS mixture algorithm for the scheduling tardiness problem. Eur J Oper Res 88:516–524

29. Almeida MT, Centeno M (1998) A composite heuristic for the single machine early/tardy job scheduling problem. Comput Oper Res 25(7–8):625–635

30. Gendreau M, Laporte L, Guimaraes EM (2001) A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times. Eur J Oper Res 133:183–189

31. Morton T, Pentico D (1993) Heuristic scheduling systems: with applications to production systems and project management. Wiley, New York

32. Scheduling Research (2005) http://www.SchedulingResearch.com