Pergamon

PII: S0305-0548(97)00054-3

Computers Ops Res. Vol. 25, No. 3, pp. 207–217, 1998

0305-0548/98 $19.00+0.00

A SYSTEMATIC PROCEDURE FOR SETTING PARAMETERS IN SIMULATED ANNEALING ALGORITHMS

Moon-Won Park‡ and Yeong-Dae Kim†§

Department of Industrial Engineering, Korea Advanced Institute of Science and Technology (KAIST), 373-1 Gusung-dong, Yusong-gu Daejon, 305-701 Korea

(Received June 1996; in revised form June 1997)

Scope and Purpose—Simulated annealing (SA) algorithms have been successfully applied to various difficult combinatorial optimization problems. To apply an SA algorithm to a specific problem, one must design the algorithm by determining methods to represent solutions, to generate neighborhood solutions and to reduce temperature, etc., and then selecting values of parameters used in the algorithm. Since the performance of an SA algorithm generally depends on values of the parameters, it is important to select the most appropriate parameter values. In previous research, parameter values are often selected using experimental designs such as full factorial designs, which require excessive computation time and efforts and frequent human decisions during the design process. In this paper, we suggest a systematic procedure to find appropriate values for parameters quickly without much human intervention by using a nonlinear optimization method.

Abstract—Values of parameters used in simulated annealing (SA) algorithms must be carefully selected since parameter values may have a significant influence on the performance of the algorithm. However, it is not easy to find good parameter values because values of a wide range have to be considered for each parameter and some parameters may be correlated with each other. In this paper, we suggest a procedure which gives good parameter values quickly without much human intervention by using the simplex method for nonlinear programming. To show the performance of the procedure, computational tests are done on a graph partitioning problem, a permutation flowshop scheduling problem and a short-term production scheduling problem. We select values for parameters needed in SA algorithms for the problems using the suggested procedure. The SA algorithms designed with this procedure are compared with existing SA algorithms in which parameter values were selected after extensive experiments.

1. INTRODUCTION

Simulated annealing (SA), which was proposed by Kirkpatrick *et al.* [9] and independently by Cerny [2], has been successfully applied to various difficult combinatorial optimization problems. SA can be viewed as a process which, given a neighborhood structure, attempts to move from the current solution to one of its neighbors. Starting from an initial solution, SA generates a new solution S' in the neighborhood of the current solution S. Then, the change in the objective function value, $\Delta = C(S') - C(S)$, is calculated, where $C(\cdot)$ denotes the objective function value of solution $\cdot$. In minimization problems, if $\Delta < 0$, transition to the new solution is accepted (this transition is called a *downhill move*). If $\Delta \geq 0$, then transition to the new solution is accepted with a specified probability usually obtained by the function $\exp(-\Delta/T)$, where T is a control parameter called the *temperature*. By allowing *uphill moves* (transitions that increase objective function values) like this, SA can escape from a local minimum in its search for the global minimum. SA repeats this process L times at a temperature, where L is a control parameter called the *epoch length*. The parameter T is gradually decreased by a *cooling function* as SA progresses, until a given stopping condition is satisfied. A typical procedure of SA is given below.

Step 1. Generate an initial solution S.
Step 2. Select a value for the initial temperature, $T_1 > 0$.
  Set the epoch count $k = 1$.
Step 3. Repeat the following L times (L is called the *epoch length*).
  (1) Generate a neighborhood solution S' of S.
  (2) Let $\Delta = C(S') - C(S)$.

† To whom all correspondence should be addressed. E-mail: ydkim@convex.kaist.ac.kr.
‡ Moon-Won Park is a Ph.D. student at the Department of Industrial Engineering, Korea Advanced Institute of Science and Technology (KAIST). He received a BS degree from Korea University and an MS degree from KAIST, both in industrial engineering. His current research focuses on the area of planning and scheduling for assembly systems.
§ Yeong-Dae Kim is an associate professor at the Department of Industrial Engineering, Korea Advanced Institute of Science and Technology (KAIST). He received a BS degree from Seoul National University and an MS degree from KAIST, both in Industrial Engineering. He received a Ph.D. degree in Industrial an Operations Engineering from the University of Michigan.

(3) If $\Delta<0$, let S be S' (downhill move).

(4) If $\Delta\geq 0$, let S be S' with probability, $\exp(-\Delta/T_k)$ (uphill move).

Step 4. If a given stopping condition is satisfied, stop. Otherwise, let $T_k+1=F(T_k)$ and $k=k+1$, and go to step 3. (T_k and F denote the temperature at the kth epoch and the cooling function, respectively.)

To apply an SA algorithm to a specific problem, a number of decisions have to be made. For example, one must determine schemes for encoding a solution and for generating an initial solution, and clearly define neighborhood of a solution. These decisions may differ if problems are different, that is, these decisions are problem specific. On the other hand, for any implementation of SA, we have to make decisions to constitute an *annealing schedule* or *cooling schedule*. Such a schedule is often characterized by the initial temperature (T_1), cooling function ($F(\cdot)$), epoch length (L), and stopping condition. In addition, several parameters called the *annealing parameters* are needed to specify an annealing schedule. For example, if an SA algorithm uses a stopping condition of terminating the procedure when the epoch count reaches a predetermined maximum count, M, a specific value of the parameter M is needed for the stopping condition. A more detailed description of the annealing schedule and the annealing parameters is given in the next section.

An experimental design such as a factorial design has often been used for selecting parameter values in previous research. However, extensive experiments are required to find values that maximize performance of an SA algorithm, since values of a wide range have to be considered for each parameter. Moreover, when parameters are correlated with each other, additional experiments may be needed. Examples of such correlation can be found in studies of Johnson *et al.* [6] and Kim *et al.* [8]. Since selecting parameter values using such an experimental design or other arbitrary methods may require a significant amount of time and efforts, a quick and simple procedure is needed for an efficient design of SA algorithms. In this research, we suggest a systematic procedure, based on the simplex method for nonlinear programming, to determine parameter values by searching a relatively small number of candidate sets of parameter values.

Of course, various non-quantitative parameters or methods have a significant impact on the performance of SA algorithms. For example, Cheh *et al.* [3] show that the performance is significantly affected by the neighborhood structure. However, even if a good neighborhood structure is used, appropriate values for various parameters have to be determined. Also, if a sufficiently long time is allowed for an SA run, selecting appropriate parameter values may not be an important issue, since an SA algorithm with a high initial temperature and a slow cooling scheme performs well as shown in the work of Anily and Federgruen [1]. However, a good solution has to be found by an SA algorithm in a short time in many circumstances. For example, computation time cannot be ignored when an SA algorithm is used to solve daily production scheduling problems on a personal computer or an SA algorithm is embedded in another algorithm as a subroutine which is called many times to solve a problem. In these cases, values of parameters must be carefully selected even if an SA algorithm is well-defined and suitable for the feature of a problem, since these values have a significant impact on the performance of the SA algorithm.

2. ANNEALING SCHEDULE

In this research, a procedure is suggested for finding the most appropriate values of annealing parameters for an annealing schedule defined by a given scheme. In other words, what is to be determined here is not a scheme for defining the annealing schedule but values of annealing parameters. However, description of such schemes may be needed to show how the suggested procedure works. In the following, we briefly describe some of schemes and annealing parameters commonly used to specify annealing schedules. See Eglese [5], Koulamas *et al.* [10] and Rose *et al.* [16] for other schemes for defining annealing schedules.

2.1. *Initial temperature (T_1)*

In Kirkpatrick *et al.* [9], the value of T_1 is set large enough to make the initial probability of accepting transitions be close to 1. However, it is known that too high initial temperature may cause a long computation time or a bad performance. They suggest that the initial temperature be chosen in such a way that the fraction of accepted uphill transitions in a trial run of annealing process is equal to a given parameter, P_1, called the initial acceptance probability. In this scheme, a number of transitions are made and the average increase in the objective function values $\bar{\Delta}$ is calculated with uphill transitions only. Then, the value of T_1 is calculated from the equation $T_1=\bar{\Delta}/\ln P_1$.

2.2. *Cooling function* ($F(T_k)$)

In much research, temperature is reduced with a so-called *proportional cooling function*, which was originally suggested by Kirkpatrick *et al.* [9]. In methods that use this function, the temperature at the kth epoch is determined with $T_k=\alpha T_k-1$, where α $(0<\alpha<1)$ is a parameter called the *cooling ratio*. If this cooling function is used, a value for α should be determined in advance. Potts and Van Wassenhove [15] suggest a method to obtain a value of α using the initial temperature (T_1), the total number of epochs (M), and the temperature at the final epoch (T_M), that is, α is determined with $\alpha=(T_M/T_1)^{1/(M-1)}$. Values of these parameters (T_1, M and T_M) should also be determined in advance for this scheme. Lundy and Mees [11] propose another cooling function, which is $T_k=T_{k-1}/(1+\beta T_{k-1})$, $\beta>0$. This gives slower cooling than the proportional cooling function at a later stage of SA algorithms. In Connolly [4], the value of β is determined with $\beta=(T_1-T_M)/\{(M-1)T_1T_M\}$. On the other hand, different values for β are used at different epochs in Van Laarhoven *et al.* [18]. That is, the value at the kth epoch is given by $\beta=\ln(1+\delta)3\sigma_k$, where σ_k is the standard deviation of objective function values of neighborhood solutions generated at the kth epoch and δ is a parameter, called the *distance parameter.*

2.3. *Epoch length (L; the number of trials allowed at each temperature level)*

The value of L can be set to be proportional to the size of the problem instance. For example, in cases of a single machine sequencing problem, L can be set to the product of the number of jobs and a predetermined parameter. As another alternative, L can be set to be proportional to the number of neighborhood solutions of a given solution. On the other hand, if the total number of trials (total number of neighborhood solutions generated) is fixed, L can be set to be proportional to the total number of trials.

2.4. *Stopping condition*

SA can be terminated when the epoch count reaches a predetermined maximum count, M or the total number of trials made so far reaches a predetermined limit, K. Another commonly used stopping rule is to restrict CPU time with a limit (T_{CPU}). In a method suggested by Van Laarhoven *et al.* [18], SA is stopped when temperature drops down to a pre-selected final temperature (T_f). On the other hand, in a method suggested by Johnson *et al.* [6], a counter is incremented by one when an epoch is completed with the fraction (or percentage) of accepted moves less than a predetermined limit (F_{MIN}) and the counter is reset to 0 when a new incumbent solution is found. In the method, SA is terminated when the counter reaches a predetermined limit I_M. This method will be denoted as the *Johnson's stopping rule* throughout this paper.

It is difficult to select an annealing schedule that gives the best performance of an SA algorithm since various schemes for defining the schedule should be selected and values of parameters included in the schemes should be determined at the same time. Moreover, some of these parameter values must be determined simultaneously because of correlation among them.

3. SIMPLEX METHOD

This section presents a method for determining values of parameters in SA algorithms. This method can be effectively used to find the most appropriate parameter values for a given scheme for defining an annealing schedule. For example, if an annealing schedule is defined by a scheme in which the initial temperature is selected with the initial acceptance probability P_1, the temperature is reduced with the proportional cooling function, the epoch length is defined as the product of the neighborhood size and a parameter s_N, and a fixed maximum trial count K is given as the stopping condition, the method suggested here can be used to find appropriate values for P_1, α, and s_N.

In previous research, parameter values are often selected using experimental designs that require extensive experiments such as the full factorial design. Therefore, only a small number of candidates for schemes that define annealing schedules are considered and most of time and efforts are spent to determine parameter values for these given candidates. As a result, the best annealing schedule may not be obtained in many cases. In the following, we suggest a systematic procedure for determining parameter values so that more candidates can be considered and hence a better annealing schedule can be obtained with less time and efforts. The procedure is based on the simplex method (for nonlinear programming).

The simplex method, which was originally suggested by Spendley *et al.* [17], can be applied to problems of minimizing $f(\mathbf{x})$, $\mathbf{x} \in \mathbf{R}^n$, where f is a function of decision variables, $\mathbf{x}=(x^1,x^2,\ldots,x^n)^T$ that are not constrained. In this method, a number of (different) solutions that form a simplex are maintained and used to generate a new and better solution, which will replace one of the current solutions in a new simplex at the next iteration. A simplex is a convex hull of $n+1$ points, $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_{n+1}$, in $\mathbf{R}^n$ such that $\{\mathbf{x}_2-\mathbf{x}_1,\mathbf{x}_3-\mathbf{x}_1,\ldots,\mathbf{x}_{n+1}-\mathbf{x}_1\}$ is a linearly independent set (for example, a line segment in $\mathbf{R}^1$, a triangle in $\mathbf{R}^2$). Consider a case where there are N parameters to be set. Let $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_{N+1}$ be points in $\mathbf{R}^N$ that form the current simplex. The main idea of the method is to replace the worst point $\mathbf{x}_W$ (having the maximum function value) among the $N+1$ points with a new and better point. The replacement of the point involves three types of operations: reflection, expansion, and contraction. These operations use a reflection coefficient $\xi>0$, an expansion coefficient $\zeta>1$, and a contraction coefficient $\psi>1$, which must be selected before the simplex method is executed. A typical procedure of the method is given below.

Step 1. Choose initial points $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_{Nn+1}$ to form a simplex in $\mathbf{R}^N$.

Step 2. If a given stopping condition is satisfied, stop. Otherwise, find $\mathbf{x}_B,\mathbf{x}_W \in \{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_{N+1}\}$ such that $f(\mathbf{x}_B)=\min_{1\le j\le N+1} f(\mathbf{x}_j)$, $f(\mathbf{x}_W)=\max_{1\le j\le N+1} f(\mathbf{x}_j)$. Let $\bar{\mathbf{x}}=\sum_{j\neq W} \mathbf{x}_j/N$.

Step 3. Let. $\mathbf{x}_R=\bar{\mathbf{x}}+\xi(\bar{\mathbf{x}}-\mathbf{x}_W)$. If $\max_{1\le j\le N+1}\{f(\mathbf{x}_j): j\Rightarrow\neq W\}\ge f(\mathbf{x}_R)$, then replace $\mathbf{x}_W$ with $\mathbf{x}_R$ (*reflection*) to form a new simplex and go to step 4. Otherwise, go to step 5.

Step 4. Let $\mathbf{x}_E=\bar{\mathbf{x}}+\zeta(\mathbf{x}_R-\bar{\mathbf{x}})$. If $f(\mathbf{x}_B)>f(\mathbf{x}_R)$ and $f(\mathbf{x}_R)>f(\mathbf{x}_E)$, replace $\mathbf{x}_R$ (which was $\mathbf{x}_W$ before reflection at step 3) with $\mathbf{x}_E$ (*expansion*) to form a new simplex. Go to step 2.

Step 5. Let $\mathbf{x}'$ be defined by $f(\mathbf{x}')=\min\{f(\mathbf{x}_R),f(\mathbf{x}_W)\}$ and let $\mathbf{x}_C=\bar{\mathbf{x}}+\psi(\mathbf{x}'-\bar{\mathbf{x}})$. If $f(\mathbf{x}_C)\le f(\mathbf{x}')$, then replace $\mathbf{x}_W$ with $\mathbf{x}_C$ (*contraction*) to form a new simplex. Otherwise, replace $\mathbf{x}_j$ with $\mathbf{x}_j+(\mathbf{x}_B-\mathbf{x}_j)/2$ for $j=1,2,\ldots,N+1$. Go to step 2.

The procedure is terminated (at step 2) if the points forming the current simplex converge to a point ($|x_i^k-x_j^k|\le\epsilon^k$, for all i, j and k, where and x_i^k are x_j^k the kth elements of points $\mathbf{x}_i$ and $\mathbf{x}_j$ in the current simplex, and each element of vector ϵ has a given small positive value), or the total number of points generated reaches a predetermined number. In the simplex method used in this research, 1/3, 2.0, and 1.5 are used for ξ, ζ and ψ, respectively, which were obtained after a series of preliminary tests. Since appropriate values for these three coefficients usually depend on the orders of magnitudes of the decision variables, the above values can be applied to search for parameter values of SA algorithms for other problems. In fact, the simplex method using these coefficient values performed well when selecting parameter values for an SA algorithm for a parallel machine scheduling problem, although it is not reported here.

In the above procedure, $\mathbf{x}$ and $f(\mathbf{x})$ correspond to parameters and the performance of an SA algorithm, respectively. Note that transitions of solutions in SA algorithms are stochastic, and hence different final solutions may be obtained even when the same set of parameter values is used in an SA algorithm for the same problem instance. Because of such stochastic nature, several replications of SA runs on a set of parameter values may be necessary. Therefore, on a point, $\mathbf{x}$, the SA algorithm is run several times with the same parameter values associated with $\mathbf{x}$, then the average of the final solution values can be used as the function value $f(\mathbf{x})$. Although other measures, i.e., other functions of the solution values, can be used to define $f(\mathbf{x})$, such as the variance of the solution values and a weighted sum of the average and the variance of the solution values, the one used here is a rather simple but most common one, the average value.

If an SA algorithm should be used for a variety of problem instances with different sizes and/or data, a more sophisticated method may be needed to evaluate the function value. As an alternative for such a case, the following method may be used. For each simplex, a number of problem instances are generated in such a way that the resulting problems well represent the problems for which an SA algorithm will be used. These generated problems are solved using the SA algorithms with candidate sets of parameter values, i.e., $\mathbf{x}_j$, $j=1,2,\ldots,N+1$. Then, performances of the sets are evaluated from the solutions using a relative performance measure such as performance ratios or ranks of the solutions for each of the problems. Here, a relative performance measure is preferred to an absolute one, i.e. a solution value itself, since data in different problems may differ and so may the magnitudes of the solutions. (If the absolute measure is used, variance in the solutions may become too large to evaluate the points correctly.) The average values of the relative performance measures over the problems can be used as the function value $f(\mathbf{x})$.

While the original simplex method was developed for problems with unconstrained variables, some

parameters in an SA algorithm may be bounded. Because of such bounds, the simplex method is slightly modified as follows. Whenever an element of a point generated by the simplex method exceeds a bound on the value of the corresponding parameter, the bound is used as the value for the element. There are theoretical bounds for certain parameters, but tighter bounds can be given in practice for some of these parameters. For example, in the proportional cooling function, theoretically the parameter α has a value between 0 and 1, while Eglese [5] reports that values of α that can be used in practice lie between 0.8 and 0.99. In such cases, tighter bounds are used instead of rather loose theoretical bounds in the simplex method.

4. COMPUTATIONAL EXPERIENCE

To show the performance of the suggested procedure, computational tests are done on a graph partitioning problem, a permutation flowshop scheduling problem and a short-term production scheduling problem.

4.1. Graph partitioning problem

The graph partitioning problem is the problem of partitioning vertices of a given graph into equal-sized sets with the objective of minimizing the cut size. Here, the size of a set denotes the number of vertices in the set and the cut size denotes the number of edges that have end-points in different sets. Johnson *et al.* [6] suggest an SA algorithm for the problem. In their research, values of parameters needed for the SA algorithm are selected through extensive computational experiments and analysis of the effects of changes in parameter values. In this paper, we determine values of parameters needed in the SA algorithm using the suggested simplex method, and then compare the performance of the resulting SA algorithm with that of Johnson *et al.* [6].

First, we briefly describe the SA algorithm suggested by Johnson *et al.* (denoted by SAJ hereafter) in the following. A solution is defined as a partition (V_1,V_2) of a given vertex set V (the two sets, V_1 and V_2, may be of different sizes), while two partitions are defined as neighbors if one can be obtained from the other by moving a single vertex from one set to the other (rather than by exchanging two vertices in different sets). In the SA algorithm, the objective function is modified to $c(V_1,V_2)=|\{u,v\}\in E: u\in V_1, v\in V_2\}|+\tau\cdot(|V_1|-|V_2|)^2$, where $|\cdot|$ is the cardinality (number of elements) of set $\cdot$ and τ is a non-negative parameter called the *imbalance factor.* In this function, an infeasible partition, of which the two partitioned sets are of different sizes, is penalized according to the square of the difference. An initial solution is obtained by generating a random partition, that is, each vertex is randomly assigned to V_1 or V_2 with equal probability. Note that infeasible solutions are generated and treated as if they were feasible during the search process. If the algorithm generates a feasible solution that is better than the current incumbent solution, the solution is saved as a new incumbent. If the best solution obtained is infeasible when SAJ terminates, a greedy heuristic is used to make the best solution feasible. The greedy heuristic selects a vertex in the larger set that can be moved to the other set with the least increase in the cut size, and moves it. The method repeats such movements until the two sets are of the same size. The resulting feasible solution is then compared with the incumbent solution for the final solution.

In SAJ, the initial temperature is selected with a commonly used scheme, i.e., one with the initial acceptance probability P_1, while temperature is reduced with the proportional cooling function with cooling ratio, α. The epoch length is defined as the product of the neighborhood size and a parameter s_N called the *size factor.* SAJ uses as the termination criterion the *Johnson's stopping rule* with two parameters F_{MIN} and I_M, which was described in Section 2. To select values of the parameters and to test SAJ, Johnson *et al.* generated problem instances (random graphs) using two parameters, the number (n) of vertices and the probability (p) of connectivity of each pair of vertices. They generate one random graph with $p=0.01$ and $n=500$ to select parameter values. After extensive experiments to find the most appropriate parameter values for the random graph, they finally select 0.4, 0.95, 0.02, 5, 16 and 0.05 for the annealing parameters, P_1, α, F_{MIN}, I_M, s_N, and τ, respectively.

Since the suggested procedure devised for selecting appropriate parameter values, we use the same schemes for our SA algorithm (denoted as SA_{NEW1} hereafter) as those for SAJ to define an annealing schedule, except for the stopping rule. SA_{NEW1} is terminated when CPU time reaches a predetermined time limit T_{CPU} or temperature falls below T_f. For a value of T_{CPU}, we randomly generated 10 test problems, ran SAJ five times for each problem, and obtained the average of computation times of these fifty runs. This average CPU time was used as T_{CPU} for the stopping rule in SA_{NEW1} so that the two SA

algorithms can be given approximately the same CPU time. This is because there is a strong correlation (although it may not be a linear one) between solution quality and computation time in SA algorithms. When the Johnson's stopping rule is used with smaller values of F_{MIN} and larger values of I_M in an SA algorithm, better (or at least, equally good) solutions can be obtained, since an SA algorithm with such values runs for a longer CPU time and in general, SA algorithms give better solutions if they run longer. Therefore, if values of the parameters are to be selected with the simplex method, values that will be obtained are those that give better solutions although they require a very long CPU time.

In most cases, it is not necessary to run an SA algorithm any longer when temperature is very low, since moves to neighborhood solutions are rarely accepted. Therefore, SA_{NEW1} was terminated when temperature dropped down to 0.1 even though CPU time did not reach the CPU time limit, T_{CPU}. (At the temperature of 0.1, the probability that an uphill transition with $\Delta=1$ is accepted becomes approximately 0.) This additional stopping condition is used to save computation time for the simplex method, since at earlier stages of the method it is likely that the method generates and evaluates bad (inappropriate) parameter values.

As mentioned in the previous section, bounds on parameter values are used in the suggested procedure. Considering results of previous research on SA algorithms, we used (0.2, 0.8), (0.8, 0.99), (4.0, 32.0), and (0.02, 0.5) for the ranges (lower and upper bounds) of P_1, α, s_N, and τ, respectively. The simplex method was terminated if points forming the current simplex converged to a point, that is, differences in elements of any two points in the current simplex were less than 0.01, 0.005, 0.2, and 0.005 for the four parameters, respectively, or the total number of points generated reached a predetermined limit, $I_{MAX}=70$. The suggested procedure and SA algorithms were coded in Pascal and all the tests were done on a personal computer with a Pentium processor.

First, to determine parameter values using the suggested procedure, we generated a random graph with $n=500$ and $p=0.01$. (Note that Johnson *et al.* also used only one problem instance to select parameter values needed for SAJ.) In the simplex method, the function value, $f(\mathbf{x})$, of a point $\mathbf{x}$ was calculated as follows. For each point in the simplex, SA was run three times with a set of parameter values corresponding to the point, and the average of the solutions from these runs was used as $f(\mathbf{x})$. The simplex method generated 47 different points (it was stopped since the points forming the simplex converged to a point). Values of the parameters, P_1, α, s_N, and τ, obtained from this method are given with those of SAJ in Table 1.

Next, we randomly generated 15 new test problems with $n=500$ and $p=0.01$ for a comparison of the performance of SAJ and SA_{NEW1} and ran both SA algorithms ten times for each problem. The results are summarized in Table 2, which shows solution qualities of the two algorithms and CPU times of SAJ (CPU time of SA_{NEW1} was fixed at 1800 s). To compare solution qualities of the two algorithms, analysis of variance is done and the result is given in Table 3. As can be seen from the ANOVA table, there is statistically no difference in the solution qualities. Although SA_{NEW1} did not outperform SAJ, a smaller number of alternative sets were evaluated in the simplex method as compared with the experiments done by Johnson *et al.* [6] for SAJ. A total of 141 SA runs was executed (47 points $\times$ 3 runs per point) in the simplex method, while it is estimated that parameter values of SAJ were obtained after more than 1600 runs (200 runs for selecting the value of τ, more than 180 runs for P_1, and 1280 runs for α and s_N). Moreover, the simplex method could determine parameter values automatically.

4.2. *Permutation flowshop scheduling problem*

In the permutation flowshop scheduling problem considered here, n jobs are to be processed on machines $1,\ldots,m$ in that order. The processing times (p_{ij}) of job i on machine j, for $i=1,2,\ldots,n$, and $j=1,2,\ldots,m$, are given. At any time, each machine can process at most one job, and preemption is not allowed. In the problem, only permutation schedules, in which sequence of the jobs on all the machines are the same, are considered, and the objective is to minimize the makespan, i.e., the maximum completion time C_{max}. For this problem, Osman and Potts [14] and Ogbu and Smith [12] give SA algorithms, and Ogbu and Smith [13] compare performance of the two SA algorithms. Since it is reported

Table 1. Parameter values obtained by the two procedures

Parameter	P_1	α	s_N	τ
Simplex method	0.627	0.908	14.491	0.0315
SAJ	0.4	0.95	16.0	0.05

Table 2. Performance of SAJ and SA_{NEW1}

Problem	SAJ		SA_{NEW1}
	Final solution	CPU seconds	Final solution
1	258.3	1674.6	262.8
	(5.93)	(281.4)	(6.80)
2	254.1	1826.8	259.5
	(5.28)	(397.9)	(6.93)
3	244.1	1743.9	235.6
	(10.06)	(281.9)	(9.22)
4	252.1	1699.1	256.0
	(5.34)	(214.4)	(6.13)
5	251.3	1866.4	255.9
	(3.89)	(267.8)	(5.61)
6	262.1	1845.3	268.4
	(9.09)	(374.7)	(6.12)
7	242.1	1708.9	226.2
	(6.38)	(274.8)	(5.77)
8	263.4	1732.2	261.2
	(5.36)	(345.6)	(6.48)
9	250.0	1715.7	254.4
	(4.14)	(240.6)	(8.38)
10	252.2	1842.4	258.2
	(5.73)	(324.1)	(5.07)
11	248.2	1773.5	269.4
	(7.45)	(185.3)	(2.27)
12	213.5	1678.8	217.1
	(5.46)	(226.5)	(7.31)
13	244.6	1655.6	232.3
	(7.97)	(271.3)	(5.83)
14	259.9	1799.7	257.7
	(10.31)	(375.7)	(7.94)
15	249.4	1693.1	229.1
	(8.17)	(229.8)	(8.23)

Average and standard deviation (in parentheses) are given.
CPU time of SA_{NEW1} for each problem is 1800 seconds.

that the SA algorithm suggested by Osman and Potts is slightly better than that of Ogbu and Smith [12], the former SA algorithm is used in the test for the simplex method suggested here. In other words, the simplex method is used to find the most appropriate values of parameters needed in the SA algorithm, and then the SA algorithm with these parameter values are compared with the SA algorithm of Osman and Potts.

Now, we briefly describe the SA algorithm suggested by Osman and Potts (denoted by SAO hereafter) in the following. In the algorithm, starting from a randomly generated initial sequence, a neighborhood sequence of a given sequence is obtained by removing a job placed at the ith position in the given sequence and inserting it at the jth position for randomly selected i and j. As the stopping condition, the total number of neighborhood solutions generated (the total number of trials) is fixed at $K=\max\{3300 \ln n+7500 \ln m-18250, 20000\}$. This number K is determined by Osman and Potts through a regression analysis with observations on the number of iterations during which a significant improvement in the objective value occurs for various values of $m\leq 20$ and $n\leq 100$.

In SAO, a single trial (for transition) is performed at each temperature (i.e., $L=1$) in order to reduce the number of parameters to be set. Osman and Potts argue that there would not be much difference in the performance of SA algorithms that perform several trials at the same temperature and that perform these trials at slightly different temperatures, and they set the total number of epochs, M, to be equal to the total number of trials. Temperatures, $T_1, T_2, \ldots, T_M$, are obtained from the function $T_{k+1}=T_k/(1+\beta T_k)$ which was suggested by Lundy and Mees [11]. A value for β can be obtained by $\beta=(T_1-T_M)/\{(M-1)T_1T_M\}$ if the total number of epochs (M), the initial temperature (T_1) and the final temperature (T_M) are fixed. Since M is fixed in SAO, only two parameters, T_1 and T_M need to be determined. Osman and Potts select $T_1=\Sigma_i\Sigma_j p_{ij}/(5mn)$ and $T_M=1$ through a series of experiments.

Table 3. ANOVA table for the comparison of SAJ and SA_{NEW1}

Source	Degree of freedom	Sum of squares	Mean square	F-value (F_0)	p-value ($\Pr>F_0$)
Algorithms	1	1.08	1.08	0.02	0.90
Problems	14	51,163.49	3654.53	50.89	0.0001
Error	284	20,395.62	71.82		
Total	299	71,560.19			

To test the simplex method suggested in this paper, we implement a new SA algorithm of which the schemes (including the annealing schedule) are the same as those for SAO. Like in SAO, the initial temperature in the new SA algorithm is also obtained from the function, $T_1\Sigma_i\Sigma_j p_{ij}/(q_T mn)$ where q_T is a parameter. Values of the two parameters, q_T and T_M, are selected using the suggested simplex method for the new SA algorithm. In this test, 0.5 and 10 are used for the lower and upper bounds on the value of q_T, respectively, and 0.1 and 10.0 are used for those of T_M. Note that when q_T is equal to 0.5 (or 10.0), the initial acceptance probability (fraction of accepted uphill transitions) is approximately 80% (or 2%). Also, if T_M=0.1, the probability of accepting an uphill move of which the increase in the objective function value is 1 is less than 0.005%, which is sufficiently small. On the other hand, if T_M=10.0, the probability of accepting such an uphill move is greater than 90%. The simplex method is terminated if the points forming the current simplex converge to a point, that is, differences in the values of the two elements, q_T and T_M, of any two points in the current simplex are both less than 0.05, or the total number of points generated reaches a predetermined limit, I_{MAX}=70.

To find a (single) best set of parameter values for the scheduling problems with up to 100 jobs and up to 20 machines, 12 problems were randomly generated with $n \in \{20, 50, 100\}$ and $m \in \{4, 7, 10, 20\}$ whenever a new iteration was started (whenever a new simplex was formed) in the simplex method. Processing times of operations were generated from a discrete uniform distribution with a range, [1, 100]. For each point, SA was run 12 times (single run per each problem) with a set of parameter values corresponding to the point, and 12 solutions for the problems were obtained at each point. However, as mentioned in the previous section, it is not desirable to use the objective function values as the function value $f(\mathbf{x})$ of a point $\mathbf{x}$. To evaluate a point (a set of parameter values) in the simplex method, ranks of the objective function values of all points in the current simplex were obtained for each of 12 problems, and then the sum of the ranks of a point is used for evaluation of the point. (Therefore, a point with a smaller rank sum is better.)

The simplex method generated 46 different points (it was stopped since points forming the final simplex converged to a point). Values of the parameters, q_T and T_M, obtained from this procedure were 9.063 and 1.612, respectively. A total of 552 SA runs was executed (46 points × 12 runs per point) in the suggested procedure and it took less than 4 hours to select the parameter values for the new SA algorithm on a personal computer with a Pentium processor. (We cannot estimate how many SA runs were performed to select the values of parameters for SAO, since the process for selecting the values is not reported in Osman and Potts [14].)

Next, 120 new test problems were generated randomly (ten test problems were generated for each of 12 combinations of values for n and m shown in Table 4) for a comparison of the performance of SAO and the new SA algorithm (denoted by SA_{NEW2} hereafter). These algorithms were run five times for each problem. The results are summarized in Table 4, which shows (relative) performance of SA_{NEW2}, which is obtained by dividing the average objective function value of SA_{NEW2} by that of SAO for each test problem. Differences in the performances of the two SA algorithms were less than 1%. To compare the solution qualities of the two algorithms, analysis of variance was done and the result is given in Table 5. As can be seen from the ANOVA table, statistically there is no difference between the performance of the two SA algorithms. Although the simplex method did not give a better SA algorithm (performance

Table 4. Average relative performance ratios

Size(n/m)	Test problems										Average
	1	2	3	4	5	6	7	8	9	10	
20/4	1.001	1.004	0.997	1.000	1.000	0.999	1.000	0.996	1.000	1.000	1.000
20/7	1.001	0.999	1.002	0.999	1.000	1.004	0.999	1.000	0.996	1.001	1.000
20/10	0.998	0.998	1.003	1.001	1.000	0.993	0.992	1.003	0.996	1.002	0.999
20/20	0.998	0.998	0.999	1.002	1.001	1.002	1.002	1.002	1.004	1.001	1.001
50/4	1.000	1.000	1.001	1.000	1.000	1.000	1.000	0.997	1.004	1.000	1.000
50/7	1.003	1.000	1.002	1.002	1.001	0.996	0.994	0.999	1.002	1.000	1.000
50/10	1.003	0.993	1.000	1.003	0.998	1.005	0.997	1.002	1.001	0.998	1.000
50/20	0.998	1.005	1.000	1.000	1.009	1.006	1.003	1.003	1.003	0.997	1.002
100/4	1.001	1.001	1.001	0.999	1.001	0.999	1.000	1.000	1.000	1.000	1.000
100/7	0.999	1.002	1.000	1.000	0.999	1.000	1.000	1.001	0.996	0.998	0.999
100/10	1.000	0.993	1.001	1.003	0.996	0.994	0.999	1.002	0.994	1.000	0.998
100/20	1.000	0.998	1.002	0.999	1.001	1.001	1.003	1.007	0.997	1.005	1.001
Overall											1.000

A relative performance ratio is the ratio of a solution value of SA_{NEW2} to that of SAO.

Table 5. ANOVA table for the comparison of SAO and SA_{NEW2}

Source	Degree of freedom	Sum of squares	Mean square	F-value (F_0)	p-value (Pr>F_0)
Algorithms	1	92	92	0.29	0.5935
Problems	119	3,910,679,136	32,862,850	99,999.99	0.0001
Error	1079	349,775	315		
Total	1199	3,911,029,003			

of SA_{NEW2} was not better than that of SAO), it found good parameter values automatically and very quickly without much human intervention.

4.3. Short-term production scheduling problem

In the short-term product scheduling problem considered here, final products are composed of subassemblies and components, and each subassembly is also made of lower-level subassemblies and/or components. Operations are aggregated into two basic ones, machining for components and assembly for subassemblies and final products. Also, processing units for the operations are aggregated into a machining shop and an assembly shop. Due dates are given for final products, and therefore components and subassemblies must be completed when they are needed for final products. If a product is completed later (earlier) than its due date, a certain amount of tardiness (earliness) penalty is incurred. The objective of the problem is to minimize the weighted sum of earliness and tardiness penalties for the final products and earliness penalties for the subassemblies and components.

Kim and Kim [7] study this problem and suggest an SA algorithm (denoted by SAK hereafter). In SAK, a solution is encoded to a string of numbers representing priorities of the items. The encoded string has one number for each item. Positions of the numbers in the string correspond to the indexes of the items, while their values denote the priorities of their corresponding items. Since items are classified into two categories, the strings are logically divided into two substrings, one for components and the other for subassemblies and final products. A feasible schedule can be generated from a solution (i.e., a string of numbers) as follows. First, final products are scheduled to be processed on their due periods. Among available items of which the parent item has been scheduled, an item with the lowest priority is selected and scheduled. If an item cannot be processed in a period because of overload, it is scheduled to be processed one (or several) period earlier. If a feasible schedule cannot be obtained, one of the final products is selected randomly and its processing is delayed by one period and then a new schedule is generated using the above procedure. This schedule revision is repeated until a feasible schedule is obtained.

In SAK, starting from a randomly generated initial solution, a neighborhood solution of a given solution is obtained by the *one-position random change method*, in which a number in the string of numbers is selected and its value is altered randomly. The initial temperature is selected with the same scheme as that for SAJ. Recall that the initial acceptance probability, P_1, is included in this scheme. The temperature at the kth epoch is determined with the proportional cooling function, i.e., $T_k = \alpha T_{k-1}$. The epoch length is set to be equal to the product of the number of items and a *size factor*, s_N. SAK uses as the termination criterion the *Johnson's stopping rule* with two parameters F_{MIN} and I_M. In SAK, values for these parameters, F_{MIN} and I_M, are (arbitrarily) set to 0.005 and 5, respectively. The other annealing parameters, P_1, α and s_N, are set to 0.3, 0.85 and 3.0, respectively, after a series of experiments.

To test the simplex method suggested in this paper, the same methods as those for SAK to define an annealing schedule are used in our SA algorithm (denoted as SA_{NEW3} hereafter) except for the stopping rule. SA_{NEW3} is terminated when CPU time reaches a predetermined time limit T_{CPU}. For a value of T_{CPU}, 10 test problems were randomly generated for each of three levels for the problem size (n), i.e., the number of components (100, 120 and 140), and SAK was run ten times for each problem. The average of computation times for each problem size was used as T_{CPU} for the stopping rule in SA_{NEW3} so that the two SA algorithms can be given approximately the same CPU time. (The average CPU times were 242 s, 372 s and 543 s when $n=100$, 120 and 140, respectively.)

What needs to be determined in the simplex method are values of P_1, α and s_N. Note that parameters, F_{MIN} and I_M that define a stopping condition are not needed in SA_{NEW3} since SA_{NEW3} is terminated when CPU time reaches T_{CPU}. Ranges (lower and upper bounds) of parameter values examined in the suggested procedure are (0.2, 0.8), (0.8, 0.99), and (0.1, 16.0) for P_1, α, and s_N, respectively. The simplex method was terminated if differences in elements of any two points in the current simplex were less than 0.01, 0.005 and 0.2 for the three parameters, respectively, or the total number of points generated reached a

Table 6. Average relative performance ratios

Size(n)	Test problems										Average
	1	2	3	4	5	6	7	8	9	10	
100	1.003	1.000	1.009	1.000	0.998	1.001	0.990	1.003	1.005	0.999	1.001
120	0.995	0.997	1.000	0.990	0.998	1.001	0.994	0.992	1.001	1.001	0.997
140	1.000	0.994	1.002	1.000	0.996	1.000	0.987	0.999	0.999	0.990	0.997
Overall											0.998

A relative performance ratio is the ratio of a solution value of SA_{NEW3} to that of SAK.

predetermined limit, $I_{MAX}=70$.

In the simplex method, the function value, $f(\mathbf{x})$, of a point $\mathbf{x}$ was calculated from solutions of three problems with $n=100$, 120 and 140 as follows. For each point, SA was run 3 times (one run for each problem size) with a set of parameter values corresponding to the point. Ranks of the objective function values of the points in the current simplex were obtained for each of the three problems, and then the sum of the ranks of a point is used for evaluation of the point. The simplex method generated 38 different points (it was stopped since the points forming the simplex converged to a point). Values of P_1, α and s_N obtained from this method were 0.334, 0.920 and 2.211, respectively.

Next, to compare the performance of SAK and SA_{NEW3}, 30 new test problems were randomly generated, 10 problems for each of the three problem sizes, $n=100$, 120 and 140, and both SA algorithms were run ten times for each problem instance. The results are summarized in Table 6, which shows (relative) performance of SA_{NEW3}, the ratio of the average objective function value of SA_{NEW3} to that of SAK for each test problem. Differences in the performances of the two SA algorithms were less than 1%. To compare solution qualities of the two algorithms, analysis of variance was done and the result is given in Table 7. As can be seen from the ANOVA table, there is no significant difference in the solution qualities. Although SA_{NEW3} did not outperform SAK, less effort was made to select values of parameters in the simplex method than in the experiments for SAK. Moreover, the simplex method automatically gives the parameter values without human intervention. A total of 114 SA runs was executed (38 points $\times$ 3 runs per point) in the simplex method, while it is reported that values for P_1, α and s_N in SAK were obtained after 180 runs. Although results of only 180 runs were reported in Kim and Kim [7], much more test runs were made to obtain such results. (Note that one of the authors of this paper was involved in the research, and this research was motivated by such a cumbersome, trial-and-error like procedure for tuning the SA algorithm.)

5. CONCLUDING REMARKS

In this paper, a procedure was suggested for determining appropriate values for parameters needed in SA algorithms. The procedure uses the simplex method for nonlinear programming. To show the performance of the suggested procedure, values of parameters were determined with the procedure for SA algorithms for three combinatorial optimization problems and the resulting SA algorithms were compared with existing SA algorithms in which parameter values were selected through extensive experiments. The results showed that there was no statistical difference in the performance of two SA algorithms for each of the three problems. The suggested procedure could find good parameter values by evaluating a smaller number of candidate sets of parameter values. Moreover, those parameter values could be obtained without much human intervention.

The suggested procedure may be effective especially when a specific stopping criterion is given. However, the procedure can be used even for cases in which one wants to find parameter values that give relatively good solutions within a *reasonable* amount of time. In these cases, one can run the suggested procedure with different values for the parameter T_{CPU} and select the most appropriate values for other parameters that give best solutions on the average. Then, more sophisticated stopping conditions such as

Table 7. ANOVA table for the comparison of SAK and SA_{NEW3}

Source	Degree of freedom	Sum of squares	Mean square	F-value (F_0)	p-value (Pr>F_0)
Algorithms	1	57,641.6	57,641.6	0.08	0.7833
Size (n)	2	360,715,252.0	180,357,626.0	236.98	0.0001
Problems	9	122,973,302.3	13,663,700.3	17.95	0.0001
Error	587	446,755,436.3	761,082.5		
Total	599	930,501,632.2			

the Johnson's stopping rule can be applied to the SA algorithm with parameter values that are selected by the suggested procedure except for T_{CPU}.

In this paper, we used two methods to compare points (sets of parameter values) in a simplex. As discussed in Section 3, however, other methods may be used for computing the function values $f(\mathbf{x})$ of the points. It is assumed in this study that the expected value of the final solutions is considered to be the measure of performance of SA algorithms. If one considers SA algorithms and the parameter values are better when they give lower variances in the solution values over different runs, variance of the final solution values can be used to compute $f(\mathbf{x})$. Also, more replications may be needed to identify statistically meaningful (in)difference among performance of SA algorithms with different sets of parameter values although longer computation time is needed.

Acknowledgements—This research was supported in part by Sanhak Foundation.

REFERENCES

1. Anily, S. and Federgruen, A., Simulated annealing methods with general acceptance probabilities. *Journal of Applied Probability*, 1987, **24**, 657–667.
2. Cerny, V., Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 1985, **45**, 41–51.
3. Cheh, K. M., Goldberg, J. B. and Askin, R. G., A note on the effect of neighborhood structure in simulated annealing. *Computers and Operations Research*, 1991, **18**, 537–547.
4. Connolly, D., An improved annealing scheme for the QAP. *European Journal of Operational Research*, 1990, **46**, 93–100.
5. Eglese, R. W., Simulated annealing: A tool for operational research. *European Journal of Operational Research*, 1990, **46**, 271–281.
6. Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C., Optimization by simulated annealing: An experimental evaluation; Part 1, Graph partitioning. *Operations Research*, 1989, **37**, 865–892.
7. Kim, J-U. and Kim, Y-D., Simulated annealing and genetic algorithms for scheduling products with multi-level product structure. *Computers and Operations Research*, 1996, **23**, 857–868.
8. Kim, Y-D., Lim, H-G. and Park, M-W., Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. *European Journal of Operational Research*, 1996, **91**, 124–143.
9. Kirkpatrick, S., Gelatt, C. D.Jr. and Vecchi, M. P., Optimization by simulated annealing. *Science*, 1983, **220**, 671–680.
10. Koulamas, C., Antony, S. and Jaen, R., A survey of simulated annealing applications to operations research problem. *Omega*, 1994, **22**, 41–56.
11. Lundy, M. and Mees, A., Convergence of an annealing algorithm. *Mathematical Programming*, 1986, **34**, 111–124.
12. 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*, 1990, **17**, 243–253.
13. Ogbu, F. A. and Smith, D. K., Memoranda: Simulated annealing for the permutation flowshop problem. *Omega*, 1990, **19**, 64–67.
14. Osman, I. H. and Potts, C. N., Simulated annealing for permutation flow-shop scheduling. *Omega*, 1989, **17**, 551–557.
15. Potts, C. N. and Van Wassenhove, L. N., Single machine tardiness sequencing heuristics. *IIE Transactions*, 1991, **23**, 346–354.
16. Rose, J., Klebsch, W. and Wolf, J., Temperature measurement and equilibrium dynamics of simulated annealing placement. *IEEE Transactions on Computer Aided Design*, 1990, **9**, 253–259.
17. Spendley, W., Hext, G. R. and Himsworth, F. R., Sequential application of simplex design of optimization and evolutionary operations. *Technometrics*, 1962, **4**, 441–461.
18. Van Laarhoven, P. J., Aarts, E. H. L. and Lenstra, J. K., Job shop scheduling by simulated annealing. *Operations Research*, 1992, **40**, 113–125.