Draft - submitted for publication

A Two-Stage Simulated Annealing Methodology

James M. Varanelli and James P. Cohoon

Department of Computer Science

University of Virginia

Charlottesville, VA 22903

USA

Corresponding Author:

James P. Cohoon

Department of Computer Science

Thornton Hall

University of Virginia

Charlottesville, VA 22903-2442

E-mail: cohoon@cs.virginia.edu

Phone: (804) 982-2210

Fax: (804) 982-2214

Abstract

We propose a two-stage simulated annealing method. While most previous work has focused on ad hoc experimentally-derived constant starting temperatures for the low temperature annealing phase, this paper instead presents a more formal method for generalized starting temperature determination in two-stage simulated annealing systems. We have tested our method on three NP-hard optimization problems using both classic and logarithmic cooling schedules. The experimental results have been consistently very good—on average the running time is halved when using a logarithmic cooling schedule and reduced by a third in the case of the classic schedule—with no loss in solution quality. We also present results for an alternative stop criterion used with the classic schedule that further reduces the two-stage running time by an additional five to ten percent in our problem suite.

Keywords: combinatorial optimization, operations research, simulated annealing, two-stage simulated annealing, VLSI design automation.

1 INTRODUCTION

The simulated annealing (SA) algorithm has proven to be an effective optimization tool in such diverse fields as VLSI computer-aided design, image processing, and operations research [1, 4, 9, 11, 14, 19, 21, 25, 30, 32]. This stems from both its general applicability to a wide range of NP-hard combinatorial optimization problems and that it consistently produces high quality approximate solutions for these problems. SA has only one significant disadvantage—its typically very long computation time.

There has been considerable effort aimed at speeding up SA. Most of this work has concentrated on the development of faster cooling schedules [2, 8, 13, 19, 20, 25]. Another approach is *two-stage simulated annealing* (TSSA) [9, 10, 14, 28, 29]. For TSSA, a faster heuristic algorithm is used to replace the SA actions occurring at the highest temperatures in the cooling schedule. The heuristic is then followed by a conventional SA approach initiated at a lower temperature in an attempt to improve the heuristic solution.

The principal consideration in the design of a TSSA system is the determination of the starting temperature for the SA phase. If the chosen temperature is too low, TSSA will become prematurely trapped in some local optimum resulting in lower solution quality than standard SA. If the chosen temperature is too high, some of the structure of the solution generated by the first-stage heuristic may be wasted because of too much algorithmic hill-climbing. Earlier TSSA approaches [9, 10, 14, 28] are based on finding a reasonable constant starting temperature for the SA phase. This requires a significant amount of experimentation with both the chosen heuristic and the specific SA implementation being used. The primary advantage of a constant starting temperature is that once the temperature has been determined and incorporated into the TSSA system, computational cost is very low. The obvious disadvantage is that if any of the heuristic, the SA implementation, or the problem itself changes, a new starting temperature must be found.

Rose, Klebsch, and Wolf [29] present a more generalized method for determining the starting temperature. Their method is based on Markov equilibrium dynamics. An approximate probability distribution for the change in cost function is found by generating a large number of random moves from the first-stage heuristic solution. This approximation is then used in a binary-search procedure to locate the corresponding SA temperature. At each trial temperature, the approximate distribution is used to calculate the total expected cost of all uphill moves and the total expected cost of all downhill moves. When the magnitudes of the two values are found to be equal, the corresponding temperature is returned as the starting temperature for the SA phase. The method is shown to produce good results for the standard cell placement problem. However, there is no attempt to apply the method to other problems. Additionally, there are both problem- and formulation-dependent constraints on the choice of first-stage heuristic and a high computational cost that have discouraged its widespread adoption.

Analysis of these previous methods for starting temperature determination does offer insight into desirable properties for a new method. The method should be generally applicable with respect to problems and SA implementations; it should be relatively insensitive to the given starting solution so as to avoid constraints on the choice of the first-stage heuristic; and it should be as computationally inexpensive as possible. This paper presents a method for starting temperature determination in TSSA systems that meets these goals. Background information is given in Section 2. Section 3 presents the derivation of the method. Section 4 gives our experimental results, and Section 5 describes our results for an alternative stop criterion used with the classic schedule. The new stop criterion was formulated due to the observation that a significant amount of computation was being performed after the classic schedule had already converged to its final solution.

2 BACKGROUND INFORMATION

In this section we first describe the SA algorithm. We then present behavior characteristics of the SA algorithm that will be used in the derivation of our method of starting temperature determination for TSSA systems given in Section 3.

2.1 The SA Algorithm

The SA algorithm was first introduced by Kirkpatrick, Gelatt, and Vecchi [17] and independently by Cerny [4] as a problem-independent combinatorial optimization technique. It is a generalization of the Metropolis Monte Carlo simulation [23]. It combines the advantages of iterative improvement techniques with randomizing techniques to yield a powerful optimization engine.

The SA process typically starts with a random solution to the optimization problem in question and a high *initial temperature*. Through the use of some *generation mechanism*, a copy of the current solution is randomly perturbed to form a new solution. The new solution is subjected to the *Metropolis acceptance criterion* [17, 23]. The Metropolis criterion always accepts the perturbed solution as the next current solution if its cost—as defined by the given problem's *cost function*—is lower than that of the current solution (assuming minimization). It also allows for the probabilistic acceptance of higher-cost perturbed solutions as the next solution, enabling the SA algorithm to

```
Simulated_Annealing()
{
  initialize($i$, $t$);
  $i_{best} = i$;
  do {
    do {
      $j$ = perturb($i$);
      $\Delta c_{ij}$ = c($j$) - c($i$);
      if (($\Delta c_{ij} \leq 0$) || (random() < exp($-\Delta c_{ij}/t$))) {
        $i$ = newstate($j$);
        if (c($i$) < c($i_{best}$)) $i_{best} = i$;
      }
    } while (equilibrium has not been reached);
    decrement($t$);
  } while (stop criterion has not been met);
  return($i_{best}$);
}
```

Figure 1: Pseudo-code for the SA algorithm using the Metropolis acceptance criterion.

climb out of local optima. This probabilistic acceptance is a function of the current temperature and the difference in cost of the current and perturbed solutions. The sequence of solutions generated at a fixed temperature can be mathematically modelled as a *Markov chain*, due to the fact that the outcome of any given Metropolis trial depends only upon the outcome of the previous Metropolis trial [27]. After a large number of Metropolis trials, the distribution of the solutions will approach the *stationary distribution* for the current Markov chain, known as the *Boltzmann distribution*. At this point, called *quasi-equilibrium*, the temperature is lowered according to some *decrement rule*. This process continues until some *stop criterion* is met, at which point the algorithm is terminated. Pseudo-code for the SA algorithm using the Metropolis acceptance criterion is given in Figure 1. More in-depth analyses of the SA algorithm are available in the literature [1, 11, 14, 19, 25, 27, 31, 32].

There are five items that must be specified for any SA implementation—the initial temperature value, the acceptance function, the length of the Markov chain at each temperature, the temperature decrement rule, and the stopping conditions. Collectively these implementation parameters are known as the *cooling schedule* [16]. There are many proposed cooling schedules present in the literature [2, 4, 8, 13, 17, 19, 20, 25, 30]. For the purpose of this paper, we concern ourselves with traditional cooling schedules that conform to the Markov chain model described above. The reason for this lies in the fact that only the Markov-based SA model has been shown to converge asymptotically to the global optimum [27], implying good heuristic approximations for most NP-hard problems when given a reasonable SA implementation. In addition, the model is relatively problem-independent. However, the Markov-based SA paradigm is computationally expensive.

Traditional SA cooling schedules tend to fall into one of two classes depending upon the employed decrement rule—*exponential* or *logarithmic*. Exponential schedules have decrement rules of the form

$$t_k = t_0 \cdot \alpha^k \tag{1}$$

where $0 < \alpha < 1$, with α usually in the 0.90-0.99 range. Logarithmic schedules have decrement rules of the form

$$t_k \sim t_{k-1} \cdot (\log \Gamma_{k-1})^{-1} \tag{2}$$

where Γ_{k-1} is some function of one or more of the aggregate statistics from the k-1th Markov chain. The seminal papers [4, 17] introduced exponential schedules, so we tend to refer to this type of schedule as the *classic schedule*. Logarithmic schedules tend to produce better solutions than do classic schedules at the expense of increased computation time. The majority of SA applications present in the literature use some variation of the classic schedule.

For the purpose of testing our proposed TSSA method, we chose to implement one classic and one logarithmic schedule for each of the problems in our test suite. The classic schedule is the one proposed by Kirkpatrick, Gelatt, and Vecchi [17]. The logarithmic schedule is that of Aarts and van Laarhoven [2]. Results for the six problem/schedule combinations are presented in Section 4, in an attempt to show that our proposed method of starting temperature determination for TSSA systems is problem- and formulation-insensitive, assuming the chosen SA cooling schedule conforms to the basic Markov chain model. Section 2.2 presents SA behavior characteristics used in the derivation of our TSSA method.

2.2 Characteristic Behavior of SA

One way of ensuring general applicability of the proposed TSSA methodology is to base the determination of the starting temperature on some characteristic behavior of the SA algorithm. We choose the behavior of the cost $c(i_{best})$ of the variate best-seen solution so far i_{best} to serve as the basis for our method. The reason for this choice lies in the fact that the cost of the solution returned

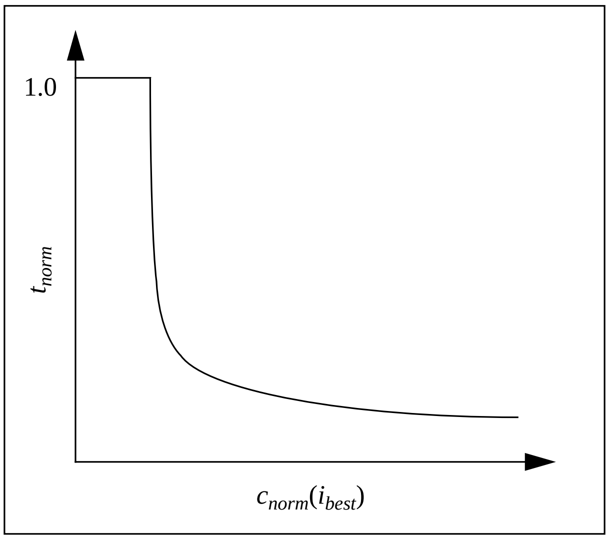


Figure 2: Normalized SA best-seen cost $c_{norm}(i_{best})$ vs. normalized SA temperature t_{norm}.

by the heuristic, i.e. the best-seen solution, is the only piece of information available at the beginning of the SA phase in a TSSA system.

Since both absolute cost and temperature are problem-dependent, a normalization scheme is required for consistency across different problems. Let E_∞ and σ_∞ respectively represent the expected cost and the standard deviation of the cost over all solutions in the state space. We can normalize the cost $c_{norm}(i)$ of a solution i by measuring it in standard deviation units σ_∞ away from the expected cost E_∞. We can normalize the temperature t_{norm} by dividing it with respect to initial SA temperature t_0. Explicitly, this leads to the following normalizations:

$$c_{norm}(i) = (E_\infty - c(i))/\sigma_\infty \tag{3}$$

and

$$t_{norm} = t_k/t_0 \tag{4}$$

where $c(i)$ is the cost of the current solution i and t_k is the k^{th} temperature value. Figure 2 illustrates the characteristic SA solution curve that results from plotting normalized best-seen cost against normalized temperature. Plots from actual SA runs can be seen in Figures 4 and 5.

Given the above normalizations, we can use a well-known behavior of the expected cost E_k and standard deviation σ_k with respect to temperature t_k can serve as the basis for a starting temperature determination scheme. In particular, large-scale numerical studies have been conducted for different pseudo-random combinatorial problems examining solution densities at varying SA temperatures by three different sets of authors [1, 11, 24]. All three independently present evidence that supports a *typical* behavior of the expected cost and standard deviation with respect to temperature, given a traditional Markov-based SA cooling schedule. Specifically, the investigations conclude that at all temperatures except those very close to the temperature corresponding to the optimal value of the cost function, the following behaviors can be noted:

$$E_k \approx E_\infty - \left(\sigma_\infty^2 / t_k \right) \tag{5}$$

and

$$\sigma_k \approx \sigma_\infty \tag{6}$$

Additionally, the investigations independently show that for this same range of temperatures, the probability distribution of the cost values can be closely approximated by a normal distribution. These observations are used in the derivation of the proposed method for starting temperature determination presented in the next section.

3 DERIVATIONS

Given the background information in the previous section, we now present the following two propositions that describe the derivation our methodology.

Proposition 1: Given a solution *i* for some combinatorial optimization problem, the following function can be used to closely approximate the SA temperature $t_k(i)$ at which *i* would be found as the best-seen solution:

$$t_k(i) \approx \frac{\sigma_\infty^2}{E_\infty - c(i) - \gamma_\infty \sigma_\infty} \tag{7}$$

Proof: It should be clear that the absolute cost $c(i_{best})$ of the best-seen solution i_{best} is proportional to SA temperature t_k—the best-seen cost decreases as the temperature is decreased. This along with Equation 3 implies that normalized best-seen cost is inversely proportional to normalized temperature:

$$t_{norm}(i_{best}) \sim \frac{\sigma_\infty}{E_\infty - c(i_{best})} \tag{8}$$

As the temperature decreases, the difference between the best-seen cost and the expected cost over all solutions (E_∞ - $c(i_{best})$) becomes greater. This relation can be seen graphically in Figures 2, 4, and 5. If t_0 is set equal to σ_∞ as proposed by White [31], replacing t_{norm} with Equation 4 leads to the following proportionality relation for absolute temperature t_k:

$$t_k(i_{best}) \sim \frac{\sigma_\infty^2}{E_\infty - c(i_{best})} \tag{9}$$

We now convert this proportion into a usable function. Solving Equation 5 with respect to absolute temperature t_k gives:

$$t_k \approx \frac{\sigma_\infty^2}{E_\infty - E_k} \tag{10}$$

This relation is quite similar to Equation 7. A function relating E_k and $c(i_{best})$ would nearly complete the proof. If i_{kmin} is the minimum-cost solution found during the kth Markov chain executing at temperature t_k, then we know that the following relation holds:

$$E_k \geq c(i_{kmin}) \tag{11}$$

This implies that the expected cost at temperature t_k is some number of standard deviation units σ_k greater than the minimum-cost solution over the kth Markov chain i_{kmin}. We call this number the *offset* and denote it by the symbol γ. In this context, Equation 11 can be expressed as:

$$E_k = c(i_{kmin}) + \gamma_k \sigma_k \tag{12}$$

If we replace the σ_k term in Equation 12 with Equation 6 and approximate i_{kmin} with the global best-seen solution i_{best}, we get:

$$E_k \approx c(i_{best}) + \gamma_k \sigma_\infty \tag{13}$$

Using the behavior of the standard deviation σ_k described by Equation 6 combined with the assumption that the configuration density is nearly normally distributed, we expect the offset γ_k to remain approximately constant at the higher temperatures while converging quickly towards zero close to the temperature corresponding to the optimal value of the cost function. Our experimental evidence indeed supports this behavior. This can be seen graphically in Figures 3 and 7, showing the evolution of the offset over the course of the SA algorithm for a number of runs. This behavior allows us to closely approximate the γ_k term in Equation 13 with γ_∞. Also, replacing the E_k term in Equation 10 by Equation 13 gives us the following:

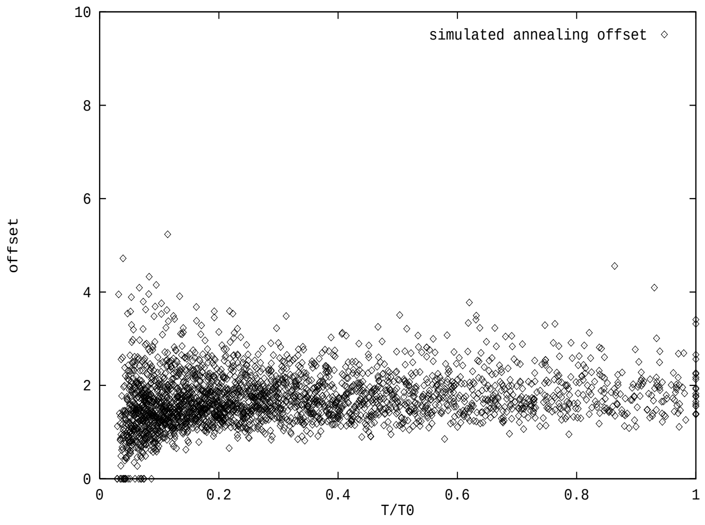


Figure 3: Evolution of the offset γ_k for the 16-terminal RSMT instance.

$$t_k(i_{best}) \approx \frac{\sigma_\infty^2}{E_\infty - c(i_{best}) - \gamma_\infty \sigma_\infty} \tag{14}$$

Finally, instantiating Equation 14 with the first-stage heuristic solution *i* for i_{best} gives Equation 7.

■

Equation 7 serves as the basis for our proposed method of starting temperature determination. However, the offset γ_∞ is still an unknown. It is important to note that without the offset term, Equation 7 generates temperature approximations that are too low, resulting in TSSA solution quality less than that of standard SA. The following proposition describes the calculation of γ_∞.

Proposition 2: Given a SA formulation for some combinatorial optimization problem with Markov chain length L_M, the offset γ_∞ can be calculated probabilistically with the equation:

$$P[E_\infty - \gamma_\infty \sigma_\infty < X < E_\infty + \gamma_\infty \sigma_\infty] \approx 1 - |L_M|^{-1} \tag{15}$$

Proof: As discussed in the previous section, we make use of the observation that we can use a normal distribution to closely approximate the probability distribution of the cost values over each Markov chain. Using this assumption, the cost values seen over the course of a Markov chain can be represented by a normally distributed random variable *X* with probability distribution function:

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left[\frac{x-\mu}{\sigma}\right]^2/2} \tag{16}$$

A simple change in variable leads to the following cumulative distribution function (CDF):

$$\Phi(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt \tag{17}$$

where $t = (x - \mu)/\sigma$ and $dx = \sigma\ dt$. This CDF can then be used in such a way as to determine the probability of generating a solution with a given cost. Explicitly, this can be written as

$$P[X \le x] = \Phi\left(\frac{x-\mu}{\sigma}\right) \tag{18}$$

If we let $z = (x - \mu)/\sigma$, the term $z_{\rho/2}$ determines the percentile ρ of the normal distribution, $\Phi(z_{\rho/2}) = \rho$, to which $X = x$ is likely to belong. For our purposes, it is useful to consider these normal probabilities in terms of standard deviation units away from the mean. In this context, $z_{\rho/2}$ can be used in the following manner:

$$P[\mu - z_{\rho/2}\sigma < X < \mu + z_{\rho/2}\sigma] = 1 - \rho \tag{19}$$

It should be clear that the $z_{\rho/2}$ term in the above equation corresponds to the offset γ used in our starting temperature determination method. The problem remains to calculate appropriate ρ values and their corresponding $z_{\rho/2}$ values to be used in determining the expected value for γ_∞. Using the behavior characteristics presented in Section 2.2, we can use the offset value γ_1 for the first Markov chain to obtain a reasonable approximation for γ_∞. We need now only determine the expected offset γ_1 for the first Markov chain.

Since virtually all L_M transitions will be accepted during the first Markov chain of the SA cooling schedule, the stationary distribution of this chain is uniform over the state space [1, 11, 24]. This implies that the mean and standard deviation of the cost values seen over the first Markov chain will approach E_∞ and σ_∞ respectively. Using this fact, we can assume that each generated solution has a probability very close to $|L_M|^{-1}$ of having the minimum cost value seen over the course of the first Markov chain. This will serve as the value for ρ. Conversely, each generated solution will have a probability very close to 1 - ρ of having a cost greater than that of the minimum cost value seen over the course of the first Markov chain. Using these values, we can now determine $z_{\rho/2}$, and hence our offset γ_∞. Since we are considering the first Markov chain, we can substi-

cells	L_M	$1 - \rho$	$z_{\rho/2}$ computed	γ_∞ observed
100	100	0.9900	2.58	2.76
500	500	0.9980	3.09	3.02
833	833	0.9988	3.22	3.11
1500	1500	0.9993	3.36	3.18
3014	3014	0.9995	3.52	3.34
10000	10000	0.9998	3.86	3.88

Table 1: Experimental results for approximating the offset for several VLSI-NPP instances.

terminals	L_M	$1 - \rho$	$z_{\rho/2}$ *computed*	γ_∞ *observed*
9	39	0.9744	2.23	1.87
11	77	0.9870	2.48	2.27
13	47	0.9787	2.30	2.18
16	83	0.9880	2.51	2.87
20	380	0.9974	3.00	2.90
30	695	0.9986	3.18	3.08

Table 2: Experimental results for approximating the offset for several RSMT instances.

cities	L_M	$1 - \rho$	$z_{\rho/2}$ *computed*	γ_∞ *observed*
20	190	0.9947	2.78	2.68
42	861	0.9988	3.22	3.33
50	1225	0.9992	3.32	3.31
57	1596	0.9994	3.38	3.62
100	4950	0.9998	3.61	3.55
318	50403	0.9999	4.26	4.31

Table 3: Experimental results for approximating the offset for several TSP instances.

tute E_∞ for μ and σ_∞ for σ in Equation 19, giving us Equation 15.

■

Given the appropriate value for ρ, the corresponding $z_{\rho/2}$ value can be calculated via numerical methods or by a table lookup. If both of these methods are infeasible for the given TSSA system, another γ_∞ approximation method can be employed at the expense of increased computation time. A single Markov chain can be executed from a randomly generated solution such that all L_M transitions are accepted. The observed μ and σ for the chain will approximate E_∞ and σ_∞. If $c(i_{min})$ is the minimum-cost value seen over the course of the Markov chain, γ_∞ can be approximated by:

$$\gamma_\infty \approx \frac{\mu_{chain} - c(i_{min})}{\sigma_{chain}} \tag{20}$$

Tables 1-3 show results for approximating the offset of each problem instance in our test suite using table values [3]. Results for each instance are averaged over 20 runs. As can be seen in the table, computed $z_{\rho/2}$ values match very closely with observed offsets. The variance seen in the offset values generated for any particular problem instance is generally quite high. However, in practice, this did not affect the ability of the TSSA systems to converge to the same quality solutions as the corresponding standard SA algorithms for the same problem. This can be seen in the TSSA results presented in Section 4.

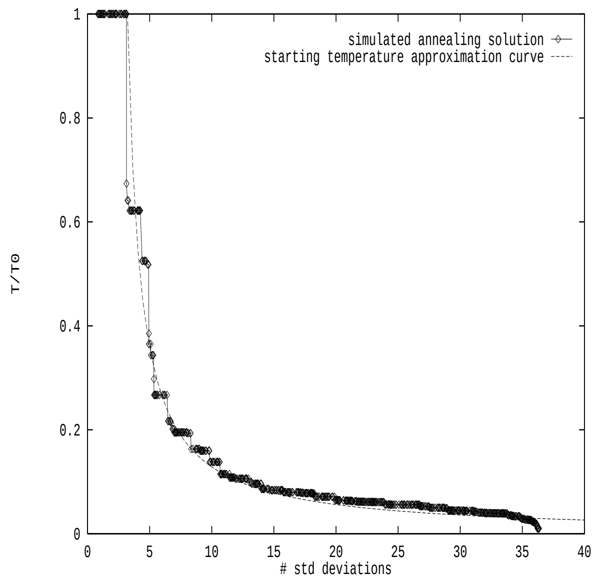
Based on the above discussion, our method can be summarized in the following steps:

- Execute the heuristic to obtain $c(i_{best})$.
- Obtain values for E_∞, σ_∞, and γ.
- Use $c(i_{best})$, E_∞, σ_∞, and γ in Equation 7 to obtain the starting temperature approximation t_{app}.
- Set $t = t_{app}$ and begin the SA phase.

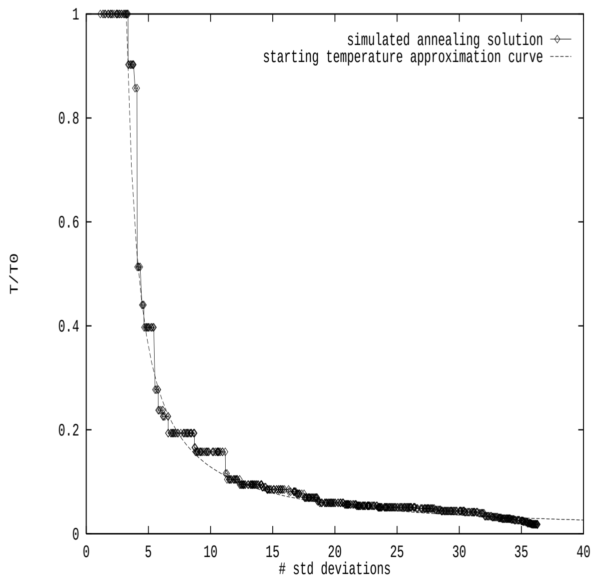
As can be seen in Figures 4 and 5, our method produces approximations that are quite close to actual SA temperatures associated with the best-seen solution for different problem/schedule combinations. Figure 4 shows our approximation curve plotted against actual SA solution curves for the Primary1 VLSI-NPP instance. Figure 5 shows our approximation curve plotted against actual SA solution curves for the 318-city TSP instance. For both figures, plots (a) and (b) concern respectively the logarithmic and classic schedules. The experimental results presented in the next section indicate that in practice there is a significant time reduction seen for TSSA systems incorporating the above methodology over standard SA with no loss in solution quality.

4 TSSA EXPERIMENTAL RESULTS

Results are presented for three different NP-hard combinatorial optimization problems, namely the VLSI network partitioning (VLSI-NPP), rectilinear Steiner minimal tree (RSMT), and traveling salesperson (TSP) problems. A short description of each problem is given in the corresponding



(a)



(b)

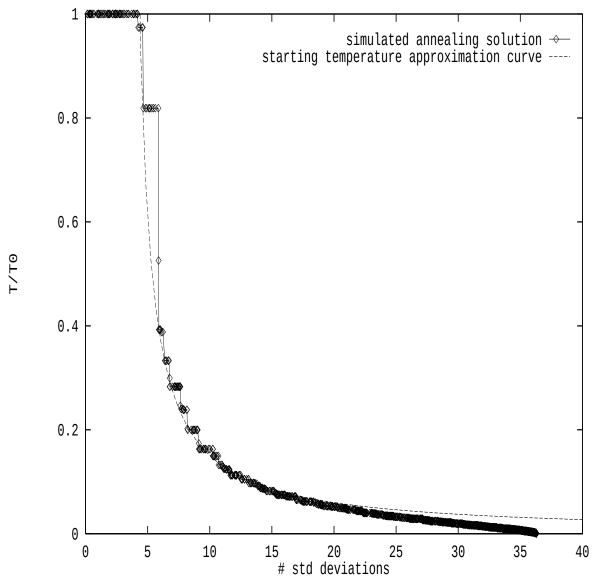
Figure 4: Starting temperature approximation curve vs. actual SA runs for Primary1 VLSI-NPP instance concerning (a) the logarithmic schedule; and (b) the classic schedule.

subsection. Each problem is solved with two different cooling schedules—a classic schedule [17] and a logarithmic schedule [2]. All TSSA systems discussed here are implemented in the C/C++ programming language using the Gnu g++™ compiler. All test runs were executed on a Sun Sparc-Server™ 10/51 operating under SunOS™ 4.1.3 (UNIX™) with 128 MB of RAM. All results are averaged over 10 runs.

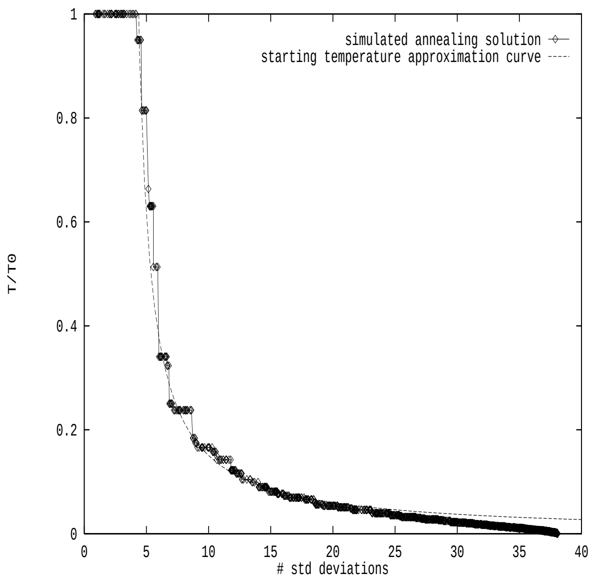
4.1 VLSI Network Partitioning Problem

The VLSI-NPP is a graph partitioning problem (GPP) [14, 16]. The input to the VLSI-NPP consists of a set of VLSI circuit elements, or *cells*, connected by a set of *nets*. Each cell has an associated positive-valued area. A net is a set of at least two cells that is to be electrically interconnected. The goal of the VLSI-NPP is to divide the cells into two blocks so as to minimize the number of nets that have cells in both blocks under the constraint that the sums of the areas of the cells in each block are approximately equal. The difference in area between the two blocks is used as a penalty term. Hence, the objective function to be minimized for the VLSI-NPP is

$$c(i) = |E_{cut}| + \lambda \cdot \left(\sum_{a_i \in A} area(a_i) - \sum_{b_i \in B} area(b_i) \right)^2 \tag{21}$$



(a)



(b)

Figure 5: Starting temperature approximation curve vs. actual SA runs for 318-city TSP instance concerning (a) the logarithmic schedule; and (b) the classic schedule.

where E_{cut} is the number of nets with cells in both blocks and λ is a weighting constant. For our VLSI-NPP implementation, $\lambda = 0.02$.

The TSSA VLSI-NPP system incorporates the Fiduccia and Mattheyses (F-M) heuristic [7]. The F-M heuristic is a generalization of the Kernighan-Lin graph bipartitioning heuristic [16]. The heuristic was selected due to its fast running times and quality of solution. The complexity of the algorithm is linear in the total number of pins, where a *pin* is an interconnection point on a cell for a particular net. We chose to only use one pass of the algorithm, since the majority of improvement takes place during the first pass. Experimental results show that SA improves solutions generated by one pass of F-M by an average of 15% relative to E_∞ in terms of standard deviation units σ_∞.

Experimental data used for evaluating the TSSA VLSI-NPP system is made up of SIGDA standard cell benchmark circuits Primary1 (833 cells, 904 nets) and Primary2 (3014 cells, 3029 nets) [26] as well as four randomly generated networks with average edge degrees similar to the benchmark circuits. The randomly generated networks range in size from 100 nets with 100 cells to 10000 nets with 10000 cells. The results are given in Tables 4 and 5 respectively for the logarithmic and classic schedules. As can be seen from the tables, significant speedup is observed in the TSSA systems incorporating our method of starting temperature determination over standard SA with a minimal difference in solution quality. The average speed-up was approximately 56% and 33%

Data instance (cells)	SA CPU time (sec)	SA nets cut	TSSA CPU time (sec)	TSSA nets cut	% CPU time decrease
100	0.69	18.8	0.44	18.8	36.2
500	7.30	118.0	3.84	119.3	47.4
833	18.97	82.1	6.52	82.6	65.6
1500	39.56	328.8	15.91	327.2	59.8
3014	145.41	222.6	48.67	228.3	66.5
10000	984.57	2066.9	386.83	2060.4	60.7

Table 4: Results for the TSSA VLSI-NPP system using a logarithmic schedule.

Data instance (cells)	SA CPU time (sec)	SA tour length	TSSA CPU time (sec)	TSSA tour length	% CPU time decrease
100	0.43	23.2	0.33	20.1	23.2
500	3.18	140.7	2.27	141.3	28.6
833	5.92	108.4	4.05	107.5	31.6
1500	12.91	398.0	7.91	400.4	38.5
3014	29.59	381.1	21.03	381.0	35.5
10000	140.06	2611.2	84.48	2616.4	39.7

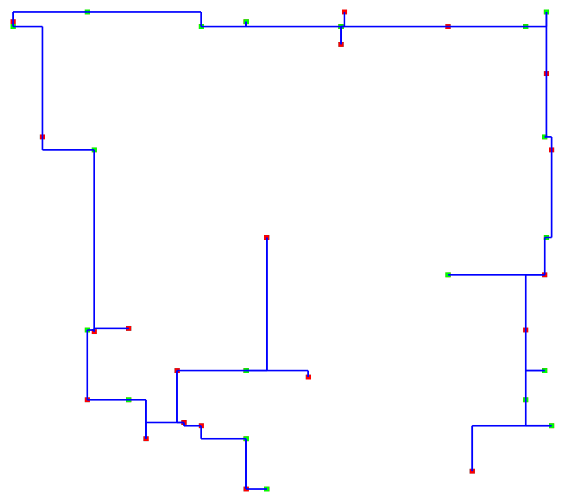
Table 5: Results for a TSSA VLSI-NPP system using a classic schedule.

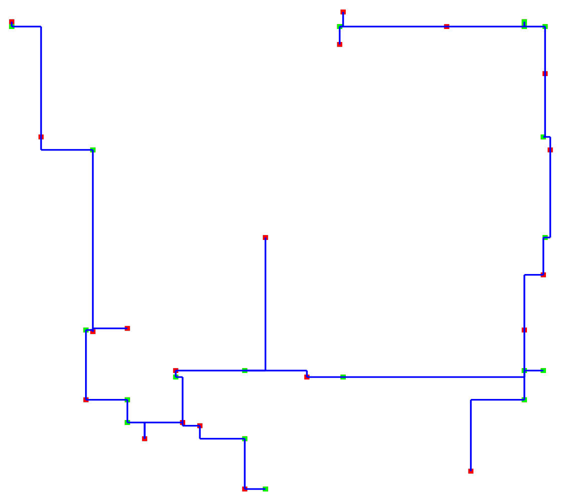
respectively for the logarithmic and classic schedules.

4.2 Rectilinear Steiner Minimal Tree Problem

The input to the RSMT problem consists of a set of *n* points in a plane, called *terminals*. The goal of the RSMT problem is to connect the terminals with horizontal and vertical line segments such that the sum of the lengths of the segments is minimized. The connected terminals should form an acyclic tree such that all of the terminals serve as endpoints to various segments. Additional points, called *Steiner points*, can also be used to connect the terminals. Using a result of Hanan [12], we can restrict a search for an optimal solution to Steiner point locations that lie on a grid imposed by the terminals. This grid defines at most O(n^2) possible Steiner locations. Hanan's result also allows us to limit the actual number of Steiner points to at most n-1. The objective function is simply

$$c(i) = \sum_{e_i \in E} length(e_i) \tag{22}$$


(a) temperature = 225.76; cost = 4512


(b) temperature = 60.74; cost = 3929

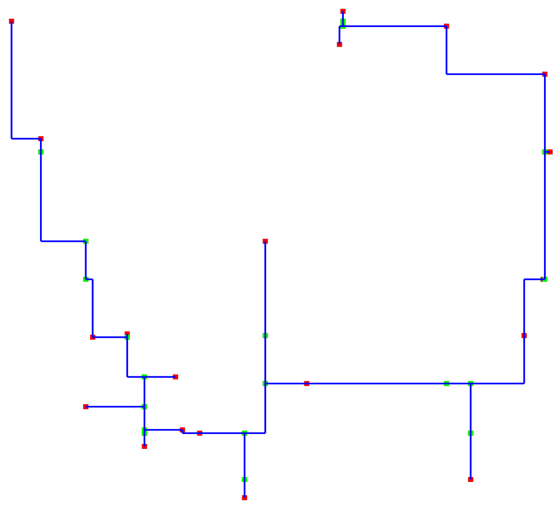

(c) temperature = 0.91; cost = 3584

Figure 6: Evolution of TSSA best-seen solution for 20-terminal RSMT instance. (a) random initial solution; (b) solution after first-stage heuristic; and (c) solution after SA phase.

where E is the set of all edges in the tree.

The heuristic chosen for the first phase of the TSSA RSMT system is based on Kruskal's minimum-spanning tree algorithm [18]. Kruskal's algorithm is first run to obtain the minimum-spanning tree for the n terminals using no Steiner points. Then up to n - 1 Steiner points are added in a greedy fashion from the set of possible Steiner locations to form the initial solution for the SA phase. SA improves solutions produced by our variation of Kruskal's algorithm by an average of 10% relative to E_∞ in terms of standard deviation units σ_∞. A graphical example of the evolution

Data instance (terminals)	SA CPU time (sec)	SA tree length	TSSA CPU time (sec)	TSSA tree length	% CPU time decrease
9	4.81	1554.4	3.82	1554.4	20.6
11	19.57	2822.0	12.37	2822.0	36.8
13	11.40	1950.6	4.33	1949.6	62.0
16	47.36	3008.4	19.42	3002.4	59.0
20	782.82	304.9	543.87	304.5	30.5
30	5155.76	359.3	3691.14	359.3	28.4

Table 6: Results for a TSSA RSMT system using a logarithmic schedule.

Data instance (terminals)	SA CPU time (sec)	SA tree length	TSSA CPU time (sec)	TSSA tree length	% CPU time decrease
9	5.20	1554.2	4.13	1554.0	20.6
11	13.31	2822.0	9.20	2822.0	30.9
13	9.91	1946.8	6.63	1942.6	33.1
16	31.45	2999.6	20.43	2998.8	35.0
20	251.69	305.4	175.79	304.7	30.2
30	1276.37	360.2	898.23	359.9	29.6

Table 7: Results for a TSSA RSMT system using a classic schedule.

of a best-seen solution for the TSSA RSMT system is shown in Figure 6.

Experimental data used for evaluating the TSSA RSMT system consists of four of the larger nets from the SIGDA benchmark circuits Primary2 as well as randomly generated 20 and 30 terminal networks. The nets taken from Primary2 range in size from 9 terminals to 16 terminals. The placements of the nets from Primary2 are intermediate solutions generated by a local placement and routing package. The results are shown in Tables 6 and 7. As is the case with the previous two problems, significant speedup is noted for the TSSA RSMT system over standard SA with no loss in solution quality The average speedup was approximately 40% and 30% respectively for the logarithmic and classic schedules.

4.3 Traveling Salesperson Problem

The input to the TSP consists of a symmetric $n \times n$ distance matrix d, representing distances between n cities. The goal is to find a minimum-length tour that visits each city exactly once while

Data instance (cities)	SA CPU time (sec)	SA tour length	TSSA CPU time (sec)	TSSA tour length	% CPU time decrease
20	0.53	258.6	0.33	254.3	37.7
42	4.58	704.6	2.43	703.7	46.9
50	6.15	240.3	2.98	236.4	51.5
57	10.21	13075.3	4.73	13133.0	53.7
100	47.74	316.0	21.53	307.3	54.9
318	1715.18	42943.7	703.22	42835.0	59.0

Table 8: Results for the TSSA TSP system using a logarithmic schedule.

Data instance (cities)	SA CPU time (sec)	SA tour length	TSSA CPU time (sec)	TSSA tour length	% CPU time decrease
20	0.21	256.6	0.16	255.1	23.8
42	1.26	705.4	0.81	704.8	35.7
50	1.59	236.3	0.97	238.0	39.0
57	2.30	13164.6	1.29	13106.5	43.9
100	7.43	326.8	3.69	320.8	50.3
318	123.11	43347.7	41.00	43360.8	66.7

Table 9: Results for a TSSA TSP system using a classic schedule.

terminating at the city of origin. The objective function to be minimized is

$$c(i) = \sum_{j}^{n-1} d_{j,j+1} + d_{n,1} \tag{23}$$

For the heuristic phase of the TSSA TSP system, the Croes heuristic [5] is used. Our experimental results show that solutions produced by our variation of the Croes algorithm are on average 10% closer to E_∞ in terms of standard deviation units σ_∞ than final SA solutions.

Experimental data used for evaluating the TSSA TSP system consists of the following instances: the 20 city problem of Croes [5]; the 42 city problem of Dantzig, Fulkerson, and Johnson [6]; a randomly generated 50 city problem; the 57 city problem of Karg and Thompson [18]; a randomly generated 100 city problem; and the 318 city problem of Lin and Kernighan [22]. The results are given in Tables 8 and 9. Again, significant speedup is noted over standard SA with no loss in solution quality. The average speedup was approximately 51% and 44% respectively for the

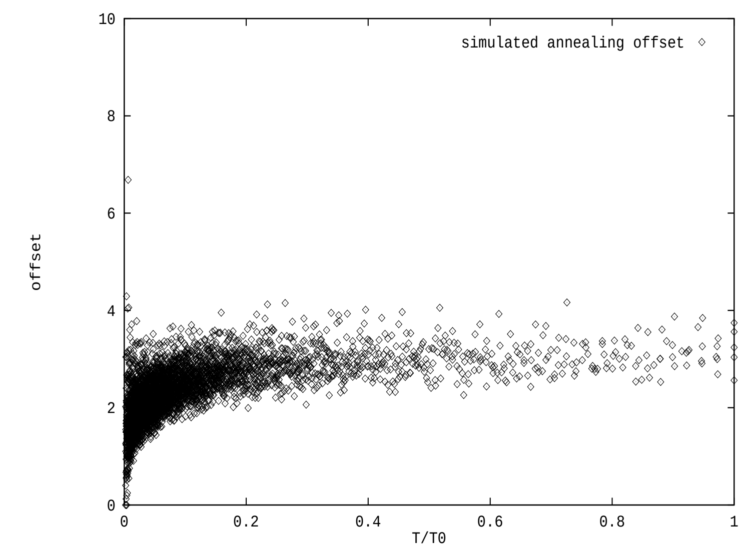


Figure 7: Evolution of the offset γ_k for the 100-city TSP instance.

logarithmic and classic schedules.

In summary, the average running time speedup for TSSA over SA for all problem instances in the test suite is approximately 50% when using a logarithmic cooling schedule and approximately 35% when using a classic cooling schedule. Both figures are clearly significant.

5 AN ALTERNATIVE STOP CRITERION

During the initial testing of our methodology, it became apparent that the classic schedule was performing a significant percentage of its total Boltzmann trials after the algorithm had already converged to its final solution. However, this proved not to be the case for the logarithmic schedule. This prompted the examination of possible new stop criteria for the classic schedule. The standard stop criterion for the classic schedule specifies to terminate the algorithm when three consecutive Markov chains end in the same value for the cost function. Since this stop criterion presents little overhead to the algorithm, one of the goals for any new stop criterion was to similarly keep any additional overhead to a minimum. We chose to focus on the behavior of the offset, as specified by Equation 12, to serve as the basis for a new stop criterion.

As discussed in Section 3, the offset remains essentially constant over the majority of the algorithm, converging quickly to zero at temperatures close to that which corresponds to the optimal value for the cost function. This behavior is shown graphically in Figure 7. The figure shows the evolution of the offset for a number of SA runs on the 100-city TSP instance. The above mentioned behavior is clearly exhibited. This behavior remains consistent across all problem/formulation combinations in our test suite.

Incorporating the offset into a new stop criterion introduces very little overhead. The average cost over each Markov chain as well as the minimum-cost value seen over each Markov chain must be tracked. This adds a single assignment statement and a single conditional statement to the inner loop of the SA algorithm. We chose a parameterized approach—specifically, the new stop criterion dictates that the algorithm be terminated at some temperature t_k when the following relation holds:

$$\frac{\mu_k - c(i_{kmin})}{\sigma_k} < \Theta \tag{24}$$

where Θ is a small user-defined constant. Θ can be optimized for the problem being solved. However, for our tests we chose a value that worked well for all three problems, specifically $\Theta = 0.0001$.

Results for the classic schedule incorporating the new stop criterion are given in Tables 10-12. As can be seen in the tables, there is generally a significant reduction in the number of Boltzmann trials performed after the final solution is found for all three of the test problems. This translated into an average 5-10% further reduction in computation time as compared to TSSA using the original stop criterion. The length of the Markov chains at each temperature and the computational cost of each Boltzmann trial determine the absolute reduction in CPU time. The VLSI-NPP system showed the least improvement due to its having by far the least computationally expensive Boltzmann trials. The RSMT system showed the greatest improvement due to its very computationally expensive Boltzmann trials, despite the fact that it has the shortest Markov chain length.

6 CONCLUSIONS

We propose a TSSA method with a more formal basis for determining the temperature at which to begin the low temperature SA phase. We have tested our method on three important optimization problems using both classic and logarithmic schedules. The results have been consistently very good. On average for a SA algorithm using a logarithmic cooling schedule the running time is cut in half; while for a SA algorithm using a classic schedule, the running time is reduced by one-third. Equally important is that there is on average no loss in solution quality. An alternative stop criterion for the classic schedule is also presented that further decreases the running times by another 5-10%.

7 ACKNOWLEDGEMENTS

The authors' work has been supported in part through National Science Foundation grants

cells	*old s.c. # trials*	*final solution trial #*	*new s.c. # trials*	*final solution trial #*	*% waste reduced*	*% time reduced*
100	4060	1456	2920	1525	53.6	2.7
500	42950	14132	26875	16330	63.4	7.6
833	89131	37941	87633	37167	1.4	0.3
1500	170025	65021	160526	63540	7.6	2.2
3014	346305	169226	334855	181062	13.2	3.1
10000	1246500	497268	1176000	495963	9.2	3.3

Table 10: Comparison of classic and newly proposed stop criteria for VLSI-NPP TSSA system.

terminals	*old s.c. # trials*	*final solution trial #*	*new s.c. # trials*	*final solution trial #*	*% waste reduced*	*% time reduced*
9	3229	2233	2751	2087	33.3	11.9
11	5548	2240	5167	2547	20.8	8.6
13	2094	1230	1470	1132	60.9	12.8
16	3872	1943	3013	1831	38.7	13.9
20	32870	27653	30856	27207	30.1	5.9
30	52264	48197	50239	48487	10.2	3.4

Table 11: Comparison of classic and newly proposed stop criteria for RSMT TSSA system.

cities	*old s.c. # trials*	*final solution trial #*	*new s.c. # trials*	*final solution trial #*	*% waste reduced*	*% time reduced*
20	12873	11040	11895	10976	49.9	4.4
42	66748	50880	57610	51216	59.7	8.8
50	76328	67011	74845	67728	23.6	3.3
57	131890	95595	109838	94883	58.8	11.0
100	398475	365822	390699	378412	62.4	7.1
318	5611534	4744012	5264806	4693109	34.1	4.7

Table 12: Comparison of classic and newly proposed stop criteria for RSMT TSSA system.

MIP-9107717 and CDA-8922545. Their support is greatly appreciated.

8 REFERENCES

[1] E.H.L. Aarts, J.H.M. Korst, and P.J.M. van Laarhoven, "Solving Traveling Salesman Problems by Simulated Annealing," *J. Stat. Phys.*, vol. 50, 187-206, 1988.

[2] E.H.L. Aarts and P.J.M. van Laarhoven, "A New Polynomial-Time Cooling Schedule," *Proc. IEEE ICCAD-85*, Santa Clara, CA, 206-208, 1985.

[3] W.H. Beyer, Ed., *CRC Standard Mathematical Tables and Formulae*, CRC Press, Boca Raton, FL, 1991.

[4] V. Cerny, "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm," *J. Optimization Thry. and Appl.*, vol. 45, 41-51, 1985.

[5] G.A. Croes, "A Method for Solving Traveling-Salesman Problems," *Operations Research*, vol. 5, 791-812, 1958.

[6] G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson, "Solution of a Large Scale Traveling-Salesman Problem," *Operations Research*, vol. 2, 393-410, 1954.

[7] C.M. Fiduccia and R.M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *Proc. 19th ACM/IEEE DAC*, Las Vegas, NV, 241-247, 1985.

[8] J.W. Greene and K.J. Supowit, "Simulated Annealing Without Rejected Moves," *IEEE Trans. CADICS*, vol. 5, 221-228, 1986.

[9] L.K. Grover, "A New Simulated Annealing Algorithm for Standard Cell Placement," *Proc. IEEE ICCAD-86*, Santa Clara, CA, 378-380, 1986.

[10] L.K. Grover, "Standard Cell Placement Using Simulated Sintering," *Proc. 24th ACM/IEEE DAC*, Miami Beach, FL, 56-59, 1987.

[11] B. Hajek, "A Tutorial Survey of Theory and Applications of Simulated Annealing," *Proc. 24th IEEE Conf. Decision and Control*, Ft. Lauderdale, FL, 755-760, 1985.

[12] M. Hanan, "On Steiner's Problem With Rectilinear Distance," *SIAM J. Appl. Math.*, vol. 14, 255-265, 1966.

[13] M.D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An Efficient General Cooling Schedule for Simulated Annealing," *Proc. IEEE ICCAD-86*, Santa Clara, CA, 381-384, 1986.

[14] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part 1, Graph Partitioning," *Operations Research*,

vol. 37, no. 6, 865-892, 1989.

[15] R.L. Karg and G.L. Thompson, “A Heuristic Approach to Solving Traveling-Salesman Problems,” *Management Science*, vol. 10, 225-247, 1964.

[16] B.W. Kernighan and S. Lin, “An Efficient Heuristic Procedure for Partitioning Graphs,” *Bell Sys. Tech. J.*, vol. 49, 291-307, 1970.

[17] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, “Optimization by Simulated Annealing,” *Science*, vol. 220, 45-54, 1983.

[18] J.B. Kruskal, “On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem,” *Proc. American Math. Soc.*, vol. 7, 48-50, 1956.

[19] P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, Reidel Publishing, Dordrecht, Netherlands, 1987.

[20] J. Lam and J.-M. Delosme, “Performance of a New Annealing Schedule,” *Proc. 25th ACM/IEEE DAC*, Anaheim, CA, 306-311, 1988.

[21] J. Lam and J.-M. Delosme, “Simulated Annealing: A Fast Heuristic for Some Generic Layout Problems,” *Proc. IEEE ICCAD-88*, Santa Clara, CA, 510-513, 1988.

[22] S. Lin and B.W. Kernighan, “An Effective Heuristic Algorithm for the Traveling Salesman Problem,” *Operations Research*, vol. 21, 498-516, 1973.

[23] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, “Equation of State Calculations by Fast Computing Machines,” *J. Chem. Phys.*, vol. 21, 1087-1092, 1953.

[24] R.H.J.M. Otten and L.P.P.P. van Ginneken, “Stop Criterion in Simulated Annealing,” *Proc. IEEE ICCD*, Rye Brook, NY, 549-552, 1988.

[25] R.H.J.M. Otten and L.P.P.P. van Ginneken, *The Annealing Algorithm*, Kluwer Academic Publishers, Boston, MA, 1989.

[26] B. Preas, “Benchmarks for Cell-Based Layout Systems,” *Proc. 24th ACM/IEEE DAC*, Miami Beach, FL, 319-320, 1987.

[27] F. Romeo and A. Sangiovanni-Vincentelli, “Probabilistic Hill Climbing Algorithms: Properties and Algorithms,” *Proc. 1985 Chapel Hill Conf. on VLSI*, Chapel Hill, NC, 393-417, 1985.

[28] J.S. Rose, W.M. Snelgrove, and Z.G. Vranesic, “Parallel Standard Cell Placement Algorithms with Quality Equivalent to Simulated Annealing,” *IEEE Trans. CADICS*, vol. 7, 387-396, 1988.

[29] J.S. Rose, W. Klebsch, and J. Wolf, “Temperature Measurement and Equilibrium Dynamics

of Simulated Annealing Placements," *IEEE Trans. CADICS*, vol. 9, 253-259, 1990.

[30] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *IEEE J. Solid-State Circuits*, vol. 20, 510-522, 1985.

[31] S.R. White, "Concepts of Scale in Simulated Annealing," *Proc. IEEE ICCD*, Port Chester, NY, 646-651, 1984.

[32] D.F. Wong, H.W. Leong, and C.L. Liu, *Simulated Annealing for VLSI Design*, Kluwer Academic Publishers, Boston, MA, 1988.