# Simulated Annealing Without Rejected Moves

JONATHAN W. GREENE AND KENNETH J. SUPOWIT

*Abstract*—**The customary need for rejection of candidate moves in simulated annealing is eliminated by biasing the selection of moves according to their effect on the cost function. An efficient implementation can offer a significant speed-up, though with increased memory. The logic partitioning problem is used as an illustration; both simple moves and pairwise interchanges are considered.**

## I. Introduction

SIMULATED annealing is a technique for finding a minimum or near-minimum in a function of many variables. It has proved useful for a wide range of optimization problems in computer design and other fields [3], [5], [7], [9], [12], [14]. This is true despite the long run times required. For example, many hours or even a day of CPU time are routinely devoted to the use of simulated annealing for integrated circuit placement [12].

This paper gives a general method of improving the speed of simulated annealing. We begin with a short review [9].

### *1.1. Simulated Annealing*

Optimization problems are often tackled by the iterative improvement method. Iterative improvement starts at some initial state, or set of values for the variables. Then successive small changes in the state are made in such a way that the cost function is decreased at each step. The moves from one state to another are effected by some simple procedure, for instance, modifying one of the variables while leaving all others unchanged. This process continues until there are no moves which reduce the cost function. At this point we have found a minimum. Often, however, it is a local rather than a global minimum. Doing iterative improvement from several initial states and selecting the best of the resulting local minima is one response to this problem. A more effective method is suggested by an analogy between the search for a minimum in a cost function, and the physical process by which a material changes state while minimizing its energy.

When a material is crystallized from the liquid phase, it must be cooled slowly if it is to assume its highly ordered, lowest energy state. At each temperature during this annealing process, the material is in equilibrium: the likelihood of its being in a given state is governed by the Boltzmann distribution for that temperature. As the temperature decreases, the distribution becomes concentrated on the lower energy states until, when the temperature finally reaches zero, only the minimum energy state(s) have nonzero probability. However, if the cooling is too rapid, the material does not have time to reach equilibrium. Instead, various defects become frozen into the structure.

Because iterative improvement forbids changes of state which increase the cost function, it is much like rapidly quenching a physical system to zero temperature. The local minima encountered in iterative improvement are analogous to the metastable states obtained after rapid cooling. Simulated annealing is a variation of the iterative improvement approach in which, as in physical annealing, moves that increase the cost function are permitted under the control of a temperature parameter.

### *1.2. The Metropolis Method*

Although simulated annealing is motivated by an analogy to a physical process, its implementation can be described precisely in terms of Markov chains [13]. This we now proceed to do.

A simulated annealing scheme to minimize cost function $F(x)$ over states $x$ in a finite set $S$ is defined by (1) a procedure for selecting a next state $x'$ from the current state $x$, and (2) an annealing schedule.

The customary procedure for selecting the next state is the Metropolis method [11], described as follows. Let $S_x$ be the set of states $y \in S$ reachable in exactly one move from $x$. Each move must be reversible, i.e., $y \in S_x \Rightarrow x \in S_y$. The number of possible moves, $N = |S_x|$, must be the same from any state $x$. Furthermore, it must be possible to reach any state in $S$ from any other in some number of moves. For each temperature $T$, define a function

$$a_T(\Delta) = \min \{1, \exp(-\Delta/T)\}.$$

Choose a candidate move at random; each move has probability $1/N$ of being selected. Let $y$ be the state that would result from making this move, and evaluate $\Delta = F(y) - F(x)$. With probability $a_T(\Delta)$ accept the move, setting $x' = y$. Otherwise, reject the move and set $x' = x$. The resulting sequence of states is a Markov process with transition function

$$p_T(x'|x) = \begin{cases} \frac{1}{N} a_T(F(x') - F(x)), & x' \in S_x \\ 1 - \sum_{y \in S_x} \frac{1}{N} a_T(F(y) - F(x)), & x' = x \\ 0, & \text{otherwise.} \end{cases}$$

Manuscript received October 8, 1984; revised August 12, 1985.

J. W. Greene was with Hewlett-Packard Laboratories, Palo Alto, CA 94304. He is now with Systems Research Laboratory, LSI Logic Corporation, Palo Alto, CA 94303-0857.

K. J. Supowit was with Hewlett-Packard Laboratories, Palo Alto, CA 94304. He is now with the Department of Computer Science, Princeton University, Princeton, NJ 08544.

IEEE Log Number 8405890.

If $T > 0$ this process is irreducible; that is, for any two states $x, y \in S$, there is a nonzero probability of eventually entering state $y$ given that the process is in state $x$. Defining $\pi_T$ as the stationary (or equilibrium) distribution of this process, we find that

$$\pi_T(x) = \frac{\exp(-F(x)/T)}{\sum_{y \in S} \exp(-F(y)/T)}$$

since this distribution satisfies the *detailed balance equation*[1]

$$\pi_T(x)\, p_T(x'|x) = \pi_T(x')\, p_T(x|x'), \qquad \forall\, x, x' \in S.$$

Note that $\pi_T$ has the form of a Boltzmann distribution.

The annealing schedule $[T_0, n_0], \cdots, [T_m, n_m]$ lists the sequence of decreasing temperatures and the desired number of accepted moves at each temperature. The procedure begins with some arbitrary initial state. Then for each temperature $T_j$ in the schedule, Markov transitions are executed according to $p_{T_j}(x'|x)$ until $n_j$ moves are accepted. The selection of a good annealing schedule is an area of active research (see [4] and [3, theorem B]). However, certain empirically determined guidelines have been proposed [15], [7]. One can choose the initial state randomly, and begin annealing at a sufficiently high temperature that most moves are accepted. Alternatively, a fast problem-specific heuristic is used to find an initial state with a moderately small cost function value, and annealing begins at a lower temperature.

We offer the following explanation for the use of $a_T(F(y) - F(x))$, rather than some other function $b_T(x, y)$, as the acceptance probability for a move from $x$ to $y$. Given any function $b_T$ for which the detailed balance equation is satisfied with the above $\pi_T$, it is easily shown that

$$\forall\, x \in S \quad \forall\, y \in S_x \quad b_T(x, y) \leq a_T(F(x) - F(y)).$$

Thus of all such functions, the use of $a_T(F(x) - F(y))$ results in the highest rate of acceptance of moves. Nevertheless, it is possible to use any number of other functions to determine the acceptance probability. The new method described in Section 1.3 can exploit this fully. Also, in Section 3.1 we find it helpful to use an acceptance probability function composed of two factors.

### *1.3. The Rejectionless Method*

Although the Metropolis method is simple, effective and easily programmed, it has one major drawback: at low temperatures, the running time is quite high because many candidates are rejected before each move to a different state. For example, when partitioning sparse graphs with about 1000 vertices Johnson [7] has found it useful to lower the temperature until only about one move per hundred is accepted. Furthermore, if heuristic methods are used to construct a good initial state, annealing begins at a low temperature and so there is a low acceptance rate all along. This low acceptance rate is necessary because the Metropolis method maintains no information about the effect of the possible moves on the cost function. It must choose uniformly from the $N$ possible moves, and then reject most choices.

The alternative we suggest is to keep a list of the effects of each possible move on the cost function and use this information to bias the selection of moves. The procedure, which we call the *rejectionless* method, is described as follows.

For each possible move $i$, $1 \leq i \leq N$, store a value $w_i = a_T(\Delta_i)$, where $\Delta_i$ is the change in the cost function that would result from the move and $a_T$ is as defined above. Choose move $i$ with probability

$$\frac{w_i}{\sum_{j=1}^{N} w_j}.$$

Then make the move, and update the $w_i$ values accordingly.

The sequence of states generated by the rejectionless method is probabilistically equivalent to the sequence of states generated by the Metropolis method, if the repetition of the current state is omitted each time a move is rejected. This is shown as follows. Let $\alpha_{xT}$ be the probability that the Metropolis method accepts the chosen candidate move when in state $x$ at temperature $T$:

$$\alpha_{xT} = \frac{1}{N} \sum_{j=1}^{N} a_T(\Delta_j).$$

Then the probability that the Metropolis method makes move $i$ from current state $x$ to some different state after some number of rejections is

$$\sum_{k=0}^{\infty} (1 - \alpha_{xT})^k \frac{1}{N} a_T(\Delta_i) = \frac{(1/N)\, a_T(\Delta_i)}{\alpha_{xT}} = \frac{w_i}{\sum_{j=1}^{N} w_j}$$

which is exactly the probability of choosing move $i$ under the rejectionless method.

As an aside, note that the stationary distribution $\pi_T$ of the Markov process defined by the rejectionless method is not of the form of a Boltzmann distribution. However if we weight $\pi_T(x)$ by the expected number $R_{xT}$ of repetitions of state $x$ at temperature $T$ due to rejections under the Metropolis method, we do obtain a Boltzmann distribution. That is:

$$\frac{\pi_T(x) \cdot R_{xT}}{\sum_{y \in S} \pi_T(y)\, R_{yT}} = \frac{\exp(-F(x)/T)}{\sum_{y \in S} \exp(-F(y)/T)}$$

where

$$R_{xT} = \sum_{k=1}^{\infty} k(1 - \alpha_{xT})^{k-1}\, \alpha_{xT} = 1/\alpha_{xT}.$$

[1]Detailed balance, also known as reversibility, is a sufficient but not necessary condition for stationarity when the Markov process is irreducible ([13, theorem 4.7.2]). Intuitively, detailed balance means that the likelihood of any transition equals that of the opposite transition.

The rejectionless method, implemented as described in Section II, has two advantages over the Metropolis method:

1) It has a running time per move independent of the acceptance ratio $\alpha$. By comparison, the Metropolis method runs in time proportional to $1/\alpha$, the number of candidates generated. Hence when the temperature is sufficiently low that $\alpha$ is below a certain value (which we call the *crossover point* and which is problem dependent), the new method runs faster than the Metropolis. We experimentally determined the crossover point for a specific problem, namely logic partitioning (see Section 3.2).

2) Another possible advantage of the rejectionless method is that it permits the use of acceptance probability functions other than $a_T$, without regard to the resulting lower acceptance rate. Furthermore, it is not even necessary that the function be less than or equal to one, since it is not used as a probability, but only to determine the weights $w_i$. For example, the function $a'_T(\Delta) = \exp(-\Delta/2T)$ can be used without the need for normalization. The use of $a'_T$ or other novel functions might increase the rate of convergence to the Boltzmann distribution and thus permit more rapid annealing schedules. Our experiments offer some support for this hypothesis (see Section 3.3).

Biased selection of moves has been suggested previously as a replacement for the Metropolis method in the latter's original application: simulation of physical systems [1]. However, the algorithm presented there is inefficient when there are more than a few possible values for the $w_i$, as there are in many optimization problems.

At each temperature, the rejectionless method involves solving a special case of what we call the dynamic weighted selection problem. In Section II, this problem is defined and an algorithm to solve it is presented and analyzed. In Section III, we discuss the application of the rejectionless method (using the algorithm of Section II) to logic partitioning, and present experimental results. Section IV shows how to cut the memory required by the rejectionless method for a certain type of move known as interchanges. Conclusions and open problems are presented in Section V.

## II. The Dynamic Weighted Selection Problem

In the *dynamic weighted selection problem*, an initial vector $\overline{W} = (w_1, w_2, \cdots, w_N)$ of real numbers is given. Then, repeatedly, the following two steps occur:

1) A number $X \in [0, 1)$ is input. The least integer $k$ such that $\Sigma_{j=1}^{k} w_j > X \cdot \Sigma_{j=1}^{N} w_j$ is output.
2) A list of $z$ pairs $(i_1, w'_{i_1}), (i_2, w'_{i_2}), \cdots, (i_z, w'_{i_z})$, is input, where $i_j$ is an integer in $[1, N]$ and $w'_{i_j}$ is a real number. The data structure is updated by replacing $w_{i_j}$ by $w'_{i_j}$, for each $j$, $1 \le j \le z$.

The rejectionless method for simulated annealing involves solving a dynamic weighted selection problem, where at each iteration $X$ is chosen at random, uniformly from [0, 1).

We present an algorithm to solve the dynamic weighted selection problem and analyze its worst-case time complexity; that is, we put an upper bound on the number of steps it requires per iteration.

We note in passing that for the *static* weighted selection problem in which $z$ is always 0, if $X$ is chosen uniformly over [0, 1) then the fastest expected-time algorithm uses Huffman coding (see [2, pp. 52–55]).

### 2.1. The Tree Algorithm and Its Analysis

Our algorithm, which we call the *tree algorithm*, maintains a complete $t$-ary tree, (that is, a rooted tree in which all nodes have 0 or $t$ ordered sons and all leaves are on one of two consecutive levels). We say how to choose $t$ later. $N$ of the leaves correspond to the integers 1, 2, $\cdots$, $N$. Stored at each node $v$ is a value $u(v)$: if $v$ is the leaf corresponding to $i$ then $u(v) = w_i$. Other leaves $v$ have $u(v) = 0$ and are dummies, present only to make the tree complete. Each internal node $v$ has $u(v) = \Sigma_{j=1}^{t} u(\text{son}_j(v))$, where $\text{son}_j(v)$ denotes the $j$th son of $v$. In other words, $u(v)$ is the sum of the values of all leaves descendant from $v$.

At each iteration, the algorithm starts at the root node of the tree. It examines the current node's sons to find the one whose subtree contains the leaf corresponding to the integer $k$ defined in (1) above. It does this by comparing the sons' $u$ values to a cutoff value $Y$. This is repeated at the selected son until the proper leaf is reached. The $u$ values are then updated for the next iteration. The detailed algorithm is as follows:

```
[[Select an element and output it]]
  v ← the root of the tree;
  Y ← X · u(v);
  WHILE v is not a leaf DO
    BEGIN
      j ← 1;
      WHILE u(son_j(v)) ≤ Y DO
        BEGIN
          Y ← Y − u(son_j(v));
          j ← j + 1
        END;
      v ← son_j(v);
    END;
  Output the number k corresponding to v;
[[Update the tree for the next iteration]]
  FOR j ← 1 TO z DO
    BEGIN
      [[Record the effect of the change (i_j, w'_{i_j})]]
        v ← the leaf corresponding to i_j;
        FOR each ancestor v' of v DO
          u(v') ← u(v') + (w'_{i_j} − u(v));
        u(v) ← w'_{i_j}
    END;
```

Clearly the tree algorithm is correct in the sense that it outputs the value of $k$ specified in the problem definition.

To analyze the time complexity of the tree algorithm,

note that there are $\lceil \log_t N \rceil + 1$ levels in the tree. Therefore the time required is $O(t \log_t N)$ to select an element and $O(z \log_t N)$ to update the tree. Thus, setting $t = z$, the total time per iteration is $O((z/\log z) \log N)$.

### 2.2. Implementation Details

We note the following ways to significantly speed up the tree algorithm (although its running time is still $\Theta((z/\log z) \log N)$):

1) the $t$-ary tree should be stored in an array (see [10, p. 401]);
2) instead of choosing $t$ as precisely $z$, it may be better to choose some other value for $t$ by means of (say) binary search.

Both of the above points were incorporated into a program that we used to generate the data described in the next section. We note three more possible improvements that we did not implement, but that may indeed improve the rate of growth of the expected time of the algorithm:

1) The updating of the tree can be done level-by-level, using a FIFO queue. That is, the leaves whose values are to be changed are first put in the queue. Then nodes are iteratively dequeued, their $u$ values are updated, and their fathers enqueued if not already in the queue. Thus each node to be updated is visited only once, resulting in a substantial savings if the $z$ changes tend to cluster among neighboring leaves.

2) The selection of an element is expedited when the sons of each node are examined in decreasing order of their $u$ values. Therefore it might be helpful to periodically reorder the leaves according to their time-averaged $u$ values and then to reinitialize the tree.

3) Instead of updating the tree completely after each iteration, it may be faster to update only some nodes, and then do local updating as necessary during selection.

## III. An Application: Logic Partitioning

In the logic partitioning problem, one is given a circuit of cells and nets. Each net connects two or more cells. The cells must be partitioned into two subsets $L$ and $R$ of approximately equal size so as to minimize the number of nets connected to at least one cell in each subset. (This is the special case of the hypergraph partitioning problem in which the hypergraph represents a logic circuit). Empirical studies [7] have found that simulated annealing compares favorably with other known heuristics for this problem.

### 3.1. Adapting the Method to Logic Partitioning

A scheme for partitioning by simulated annealing is as follows. A state consists of an assignment of each cell to $L$ or $R$. Moves are made by changing the assignment of a single cell. Let $N$ denote the number of cells, which is also the number of possible moves. We use a cost function that is the sum of two terms

$$F(x) = F^C(x) + F^I(x)$$

The connectivity term $F^C(x)$ is the number of nets connected to at least one cell in each subset. The imbalance term $F^I(x) = c(|L|^2 + |R|^2)$ penalizes excessive imbalance between subsets $L$ and $R$; $c$ is a problem-dependent constant. Let reassignment of cell $i$ correspond to move $i$ from state $x$ to $x'$. Then $\Delta_i = F(x') - F(x)$ is the change in the cost function that the move would cause. It is convenient to express $\Delta_i$ as the sum of two terms:

$$\Delta_i^C = F^C(x') - F^C(x)$$

and

$$\Delta_i^I = F^I(x') - F^I(x)$$

If we use the tree algorithm directly, then for each move $i$ it is necessary to store and maintain the value $w_i = a_T(\Delta_i)$. Note that when move $i$ occurs, for each cell $j$ sharing a net with cell $i$ the corresponding value $w_j$ may require updating because of a change in $\Delta_j^C$. There are generally only a small number of such cells. However, every move also changes $|L|$ and $|R|$. This in turn changes *every* $\Delta_j^I$, $1 \le j \le N$, making it prohibitively time consuming to update all affected $w_i$ values. More precisely, the parameter $z$ (introduced in Section II) could be as large as $N$; hence the time per move would be $\Theta(N)$.

To avoid this difficulty, observe that $\Delta_i^I$ takes one value $\Delta_L^I$ for all reassignments from $L$ to $R$, and another value $\Delta_R^I$ for all reassignments from $R$ to $L$. This suggests the following solution to the updating problem. Rather than use $a_T(\Delta_i)$ for the acceptance probability, as in the original Metropolis method, it is possible to use an acceptance probability composed of two factors: $a_T(\Delta_i^C)\, a_T(\Delta_i^I)$.

(When this acceptance probability is used in the Metropolis method, the resulting Markov process has transition function:

$$p_T(x' \mid x) = \begin{cases} \dfrac{1}{N} a_T(F^C(x') - F^C(x)) \cdot a_T(F^I(x') - F^I(x)), & x' \in S_x \\ 1 - \displaystyle\sum_{y \in S_x} \frac{1}{N} a_T(F^C(y) - F^C(x)) \cdot a_T(F^I(y) - F^I(x)), & x' = x \\ 0, \quad \text{otherwise.} \end{cases}$$

It is easily seen that the detailed equation is still satisfied by taking $\pi_T$ to be a Boltzmann distribution.)

We extend the tree algorithm as follows to take advantage of this factorization of the acceptance probability. We use two trees, a left and a right, each with $N$ leaves. The $i$th leaf in the left tree is assigned $w_i = a_T(\Delta_i^C)$ if cell $i$ is in $L$, and zero otherwise. The $i$th leaf in the right tree is assigned $w_i = a_T(\Delta_i^C)$ if cell $i$ is in $R$, and zero otherwise. The two trees are each maintained as described in Section II. Let us also maintain two additional values: $w_L = a_T(\Delta_L^I)$ and $w_R = a_T(\Delta_R^I)$. Let $u_L$ and $u_R$ be, respectively,

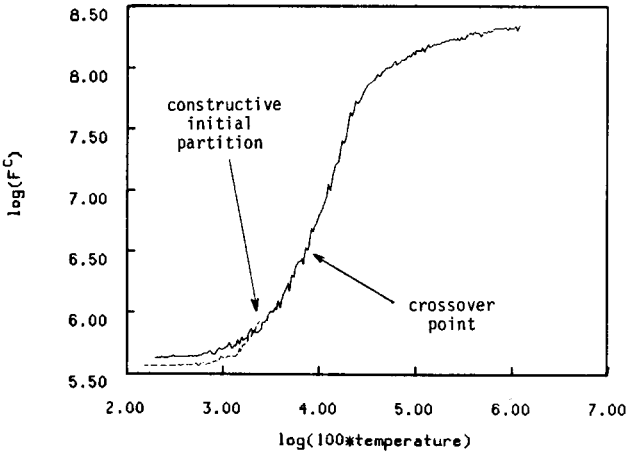



Fig. 1. Decline in cost during simulated annealing: from a random partition (solid line) and from a constructive initial partition (dashed line).

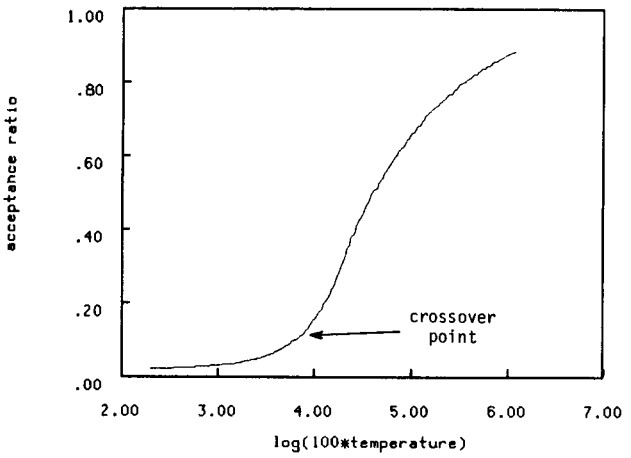



Fig. 2. Acceptance ratio as a function of temperature.

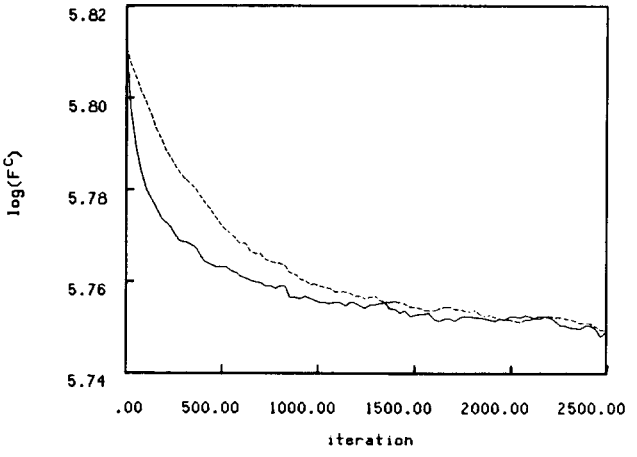



Fig. 3. Comparison of two acceptance probability functions, $a_T$ (dashed line) and $a'_T$ (solid line).

the $u$ values of the roots of the left and right tree. To select a move we start at the root of the left tree with probability $u_L w_L/(u_L w_L + u_R w_R)$ and the root of the right tree otherwise. Then we descend the chosen tree in the manner described in Section II. If the $i$th leaf is selected, move $i$ is taken by reassigning cell $i$. The likelihood of selecting move $i$ is $a_T(\Delta_i)/\Sigma_{j=1}^{N} a_T(\Delta_j)$ as desired, and the updating has been greatly simplified. In particular, the number of leaves that require updating is

$$z \leq |\{j \neq i: i \text{ and } j \text{ share a net}\}| + 2.$$

### 3.2. *Experimental Results: Crossover Point*

In order to determine its utility, the above algorithm was used to partition a microprocessor design with 7596 cells, 7023 nets, and a total of 21 920 net-cell connections. Two experiments were done: annealing from a randomly chosen initial partition, and annealing from a partition generated by a constructive heuristic. The latter partition was found by applying a linear placement heuristic (see Appendix for details) and bisecting the resulting placement.

At high temperatures, the Metropolis method was used. When the acceptance ratio reached the crossover point, the rejectionless method was employed. The factored acceptance probability function was used throughout. Similar efficient programming techniques, such as tabulation of the exponential function, were used for each method. The annealing parameters[2] were determined empirically.

Results typical of those obtained are shown in Figs. 1 and 2. Fig. 1 is a log–log plot of $F^C$, the connectivity term of the cost function, as a function of temperature. The crossover point was an acceptance ratio of roughly 11 percent, which occurred at a temperature of about 0.48. With the initial configuration generated by the constructive heuristic, it was possible to begin annealing well below the crossover point, and obtain results as good as or better than annealing from a random initial partition. Fig. 2 shows the relationship between temperature and acceptance ratio. (This was determined empirically when the Metropolis method was in use, and estimated by computing $\alpha_{xT}$ when the rejectionless method was in use.) The final acceptance ratio was 2.2 percent, for which the rejectionless method runs at five times the speed of the Metropolis method.

Identical experiments were also carried out on two smaller circuits (801 and 1196 cells, respectively). For each of these, the crossover point was about 13 percent. When using the rejectionless method, we also observed that $z$, the number of leaf nodes per move that required updating, averaged between 5 and 7 at different temperatures.

### 3.3. *Experimental Results: Alternate Acceptance Probability Function*

The functions $a_T(\Delta) = \min\{1, \exp(-\Delta/T)\}$ and $a'_T(\Delta) = \exp(-\Delta/2T)$ were compared as follows. A starting state $x_0$ was obtained by slowly annealing to temperature $T = 0.30$. From state $x_0$ we ran many iterations at temperature $T = 0.28$ using $a_T$, and observed the decline in the cost function $F^C$. We repeated this procedure by resetting the state to $x_0$, but this time using $a'_T$.

Fig. 3 shows the decline of $F^C$ averaged over 800 trials

[2]Cost function constant: $c = 10^{-4}$.
Final imbalance: $\|L| - |R\| < (0.01)N$.
Annealing Schedule: $T_{j+1} = (0.98)T_j$, $n_j = (10)N$.

using $a_T$ (dashed line) and $a'_T$ (solid line). Note that the approach to equilibrium is initially more rapid when $a'_T$ is used.

## IV. Interchanges

For certain problems, annealing seems to be accelerated by the use of moves more complex than those mentioned above. In logic partitioning, for instance, a move between states can be made by interchanging two cells from different subsets, instead of simply reassigning a single cell. Interchanges have been used in iterative improvement heuristics for the logic partitioning problem (e.g., [6]), and in application of simulated annealing to placement problems [9], [12].

As before, we illustrate with the logic partitioning problem. Suppose there are $N$ cells and $|L| = |R| = N/2$. Interchanges preserve this exact balance. Thus there are always $N^2/4$ possible interchanges from which to choose. The Metropolis method is employed in the usual way by selecting one of these interchanges as a candidate move and accepting it with probability $a_T(\Delta)$, where $\Delta$ is the change in the cost function due to the interchange. The rejectionless method can also be used in a straightforward way. However, since a value must be stored for each possible move, it would appear that treating interchanges rather than simple reassignment increases the memory requirement from $\Theta(N)$ to $\Theta(N^2)$. We present a modification of the rejectionless method described in Section 3.1 that accommodates interchanges with only $\Theta(N)$ memory.

Before presenting the modified method, some notation is required. The set of accessible states $S$ is restricted to only those partitions with $|L| = |R|$. Because the imbalance cost $F^I(x)$ is constant for all $x \in S$, we neglect $F^I$ and let $F(x) = F^C(x)$. Let $x'$ be the state obtained from state $x \in S$ by reassigning cell $i$ to the opposite subset. (Note that $x' \notin S$; some other cell must be reassigned in the opposite direction to obtain another state in $S$.) Let $\Delta_i(x) = F(x') - F(x)$, the change in (connectivity) cost that would be caused by reassigning cell $i$ when in state $x$.

The modified rejectionless method uses two trees, each with $N$ leaves, as in Section 3.1. If $x$ is the current state, the $i$th leaf in the left (right) tree is assigned $w_i = a'_T(\Delta_i(x))$ if cell $i$ is in $L$ ($R$) and 0 otherwise. We first select a cell $i$ from the left tree with probability $w_i / \Sigma_{l=1}^{N} w_l$ and then select a cell $j$ from the right tree with probability $w_j / \Sigma_{r=1}^{N} w_r$. The chosen cells are interchanged and then both trees are updated. Thus the move consisting of an interchange of cells $i \in L$ and $j \in R$ is chosen with probability

$$\frac{a'_T(\Delta_i)}{\sum_{l \in L} a'_T(\Delta_l)} \frac{a'_T(\Delta_j)}{\sum_{r \in R} a'_T(\Delta_r)} = \frac{a'_T(\Delta_i + \Delta_j)}{\sum_{l \in L} \sum_{r \in R} a'_T(\Delta_l + \Delta_r)}.$$

(The dependence of the $\Delta$'s on $x$ is omitted for clarity.)

The paradoxical thing about this method is that the two cells are selected independently, even though the change in $F$ due to their interchange cannot be expressed as the sum of the changes that would result if either cell alone were reassigned. In fact, the modified method does make some interchanges with a different probability than the straightforward method would. However, the reverse interchanges are made with a probability differing by exactly the same amount, so that the proper stationary distribution is obtained. We prove this by exhibiting the equivalent Metropolis implementation and showing that it satisfies the detailed balance equation.

Following the argument of Section 1.3, it is easily seen that the equivalent Metropolis implementation accepts an interchange of cells $i$ and $j$ with probability

$$\frac{a'_T(\Delta_i(x) + \Delta_j(x))}{\gamma_T}$$

where $x$ is the current state and

$$\gamma_T = \max_{x \in S} \max_{\substack{l \in L \\ r \in R}} a'_T(\Delta_l(x) + \Delta_r(x)).$$

(In practice, this implementation would not be used because the large normalization constant $\gamma_T$ results in a low acceptance ratio.) We show as follows that the detailed balance equation is satisfied with the usual Boltzmann distribution. Since the value of the cost function at a certain state is independent of how we got there, we have this

*Proposition:* Let states $x, y \in S$ differ only by an interchange of cells $i$ and $j$. Then $F(x) + \Delta_i(x) = F(y) + \Delta_j(y)$ and $F(x) + \Delta_j(x) = F(y) + \Delta_i(y)$.

Detailed balance is satisfied because

$$\begin{aligned}
&\exp(-F(x)/T) \cdot \frac{a'_T(\Delta_i(x) + \Delta_j(x))}{\gamma_T} \\
&\quad = \frac{1}{\gamma_T} \exp\left(\frac{1}{T}(-F(x) - \Delta_i(x)/2 - \Delta_j(x)/2)\right) \\
&\quad = \frac{1}{\gamma_T} \exp\left(\frac{1}{T}(-F(y) - \Delta_j(y)/2 - \Delta_i(y)/2)\right) \\
&\quad = \exp(-F(y)/T) \cdot \frac{a'_T(\Delta_j(y) + \Delta_i(y))}{\gamma_T}
\end{aligned}$$

The second equality follows from the proposition. Thus this Metropolis implementation, and hence the equivalent modified rejectionless implementation, both have the proper stationary distribution.

Considering interchanges instead of reassignments doubles the computation time per iteration of the Metropolis method (using acceptance function $a_T(\Delta)$). With the above modifications, the rejectionless method likewise takes about twice as long to do an interchange as a reassignment. Thus the ratio between the times for the two methods is nearly unchanged, and one expects a crossover point about the same as that found in Section 3.2. This is indeed true experimentally. However, because the acceptance ratio for interchanges is approximately the square of the acceptance ratio for reassignments, the relative speed-up for interchanges is the square of that found in Section 3.2, or about 25–30 times at the final temperature.

## Conclusions

We have presented a new algorithm for simulated annealing, whose speed does not depend on the acceptance ratio (and, therefore, does not depend on the temperature). On the other hand, the Metropolis method requires time proportional to the inverse of the acceptance ratio. Therefore, we recommend using the Metropolis method until the acceptance ratio reaches the crossover point and then switching to the rejectionless method. When a constructive heuristic is used to generate an initial configuration, it is often possible to start annealing below the crossover point and therefore one should use only the rejectionless method. The rejectionless method achieves its speed-up (at the lower temperatures) over the Metropolis method at the cost of a greater memory requirement.

For the logic partitioning problem, we found that a constructive heuristic indeed allowed annealing to start below the crossover point, while attaining results as good or better than with a random (high-temperature) start. A speed improvement of five times for simple moves and 25–30 times for interchanges was obtained at the final temperature. The memory requirement of the rejectionless method has the same rate of growth as that of the Metropolis method—linear in the number of cells; this is true for both simple moves and interchanges.

The rejectionless method also makes possible the use of new acceptance probability functions. In the case of logic partitioning, at least one such new function seems to give more rapid convergence to the Boltzmann distribution.

Among the open questions arising here are the following.

1) What is the complexity of the dynamic weighted selection problem? Can the $\Theta((z/\log z) \log N)$ bound be improved on?

2) Is it possible to select moves according to slightly inaccurate probabilities without significantly affecting the result? If so, then only intermittent or partial updating of the tree data structure may be necessary, thus permitting a further increase in speed.

3) What is the lowest acceptance ratio $\alpha_{\min}$ at which it is useful to do simulated annealing, for various problems? How does $\alpha_{\min}$ shrink with the problem size?

## Appendix
## A Linear Placement Heuristic

The *linear placement problem* is to place the cells of a logic circuit on a line so as to minimize the maximum density. More precisely, let $C$ be the set of cells and let $n = |C|$. The task is to find an ordering $c_1, c_2, \cdots, c_n$ of the cells that minimizes $\max \{\text{cut}(i): 1 \leq i < n\}$, where $\text{cut}(i)$ denotes the number of nets having at least one terminal on a cell in $\{c_1, c_2, \cdots, c_i\}$ and at least one on a cell in $\{c_{i+1}, c_{i+2}, \cdots, c_n\}$.

The linear placement heuristic used to construct an initial partition in the experiments described in Section 3.2 is as follows. At all times, $C$ is partitioned into three sets: *IN*, *FRONTIER*, and *OUT*. *IN* is the set of cells already given numbers. *FRONTIER* is the set of cells not in *IN* connected by at least one net to at least one cell in *IN*. *OUT* is the set $C - (IN \cup FRONTIER)$. We say that a net is *incident* on a cell $c$ if it has at least one terminal on $c$. We say that a net is *split* if it is incident on some cell in *IN* and on some cell in *FRONTIER* $\cup$ *OUT*. The algorithm is:

```
OUT ← C; IN ← φ; FRONTIER ← φ; [[thus, initially no nets are split]]
FOR j ← 1 TO n DO
  BEGIN
    IF FRONTIER ≠ φ
      THEN BEGIN
             c_i ← a least cost element of FRONTIER;
             FRONTIER ← FRONTIER − {c_i}
           END
      ELSE BEGIN
             c_i ← a least cost element of OUT;
             OUT ← OUT − {c_i}
           END;
    IN → IN ∪ {c_i};
    FOR each c ∈ OUT such that there is a net incident on both c and c_i DO
      BEGIN
        FRONTIER ← FRONTIER ∪ {c_i};
        OUT ← OUT − {c_i}
      END
  END
```

In this algorithm, by the *cost* of a cell $c$ in *FRONTIER* $\cup$ *OUT* we mean

(the number of nonsplit signals incident on $c$)

$$-\beta \cdot \sum_{j \geq 1} \frac{f(c, j)}{j}$$

where $\beta$ is a constant, and $f(c, j)$ is the number of split signals incident on exactly $j$ cells of *FRONTIER* $\cup$ *OUT* including $c$. Intuitively, the cost represents the damage caused by moving $c$ to the *IN* set, and consists of two components:

1) a penalty for splitting currently nonsplit nets;
2) a reward for bringing another cell of a split net into the *IN* set. The reward is weighted so that the smaller the number $j$ of its cells remaining to be brought "in," the greater the reward.

The parameter $\beta$ is simply a way to adjust the relative importance of the two components. We chose the value $\beta = 5$ when constructing the initial partition.

This heuristic is a modification of that presented in [8].

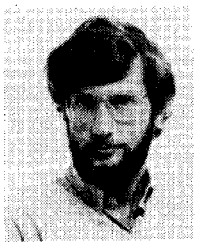
## Acknowledgment

The authors thank N. Campbell for his many helpful comments.

## References

[1] A. B. Bortz, M. H. Kalos, and J. L. Lebowitz, "A new algorithm for Monte Carlo simulation of ising spin systems," *J. Comput. Phys.*, vol. 17, pp. 10–18, 1975.
[2] R. G. Gallager, *Information Theory and Reliable Communication*. New York: Wiley, 1968.
[3] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721–741, Nov. 1984.
[4] B. Hajek, "Cooling schedules for optimal annealing," Jan. 1985, unpublished.
[5] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley, "Boltzmann machines: Constraint satisfaction networks that learn," Carnegie-Mellon Univ., Tech. Rep. CMU-CS-84-119, May 1984.
[6] T. Ishiga, T. Kozawa, and S. Sato, "A logic partitioning procedure by interchanging clusters," in *Proc. 12th Design Automation Conf.*, 1975, pp. 369–377.
[7] D. S. Johnson, L. McGeoch, C. Aragon, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation," unpublished.
[8] S. Kang, "Linear ordering and application to placement," in *Proc. 20th Design Automation Conf.*, 1983, pp. 457–464.
[9] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.
[10] D. E. Knuth, *The Art of Computer Programming*, vol. 1, *Fundamental Algorithms*. Reading, MA: Addison-Wesley, 1968.
[11] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, June 1953.
[12] F. Romeo, C. Sechen, and A. Sangiovanni-Vincentelli, "Simulated annealing research at Berkeley, in *Proc. Int. Conf. Computer Design*, (Port Chester, NY), Oct. 1984, pp. 652–657.
[13] S. M. Ross, *Stochastic Processes*. New York: Wiley 1983.
[14] M. P. Vecchi and S. Kirkpatrick, "Global wiring by simulated annealing," *IEEE Trans. Computer-Aided Des.*, vol. CAD-2, no. 4, pp. 215–222, Oct. 1983.
[15] S. White, "Concepts of scale in simulated annealing," in *Proc. Int. Conf. Computer Design* (Port Chester, NY), Oct. 1984, pp. 646–6541.

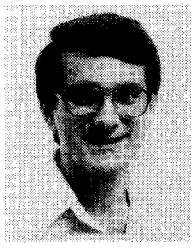
*



**Jonathan W. Greene** was born in Long Island, NY, in 1957. He received the Sc.B. degree in biology from Brown University, Providence, RI, in 1979, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1980 and 1983.

From 1983 to 1984 he was a member of the Design Automation Department at Hewlett-Packard Laboratories, and since then has been with the LSI Logic Corporation Systems Research Laboratory, both in Palo Alto, CA.

*



**Kenneth J. Supowit** received the A.B. degree in linguistics from Cornell University, Ithaca, NY, in 1978, and the Ph.D. in computer science from the University of Illinois at Urbana-Champaign in 1981, in the area of computational geometry.

From 1981 to 1984 he was a member of the Design Automation Department at Hewlett-Packard Laboratories in Palo Alto, CA. He is currently an Assistant Professor of computer science at Princeton University, Princeton, NJ. His research interests include design automation and analysis of algorithms.