Minimizing Multimodal Functions of Continuous Variables with the "Simulated Annealing" Algorithm

A. CORANA, M. MARCHESI, C. MARTINI, and S. RIDELLA
Istituto per i Circuiti Elettronici-C.N.R.

A new global optimization algorithm for functions of continuous variables is presented, derived from the "Simulated Annealing" algorithm recently introduced in combinatorial optimization.

The algorithm is essentially an iterative random search procedure with adaptive moves along the coordinate directions. It permits uphill moves under the control of a probabilistic criterion, thus tending to avoid the first local minima encountered.

The algorithm has been tested against the Nelder and Mead simplex method and against a version of Adaptive Random Search. The test functions were Rosenbrock valleys and multiminima functions in 2, 4, and 10 dimensions.

The new method proved to be more reliable than the others, being always able to find the optimum, or at least a point very close to it. It is quite costly in term of function evaluations, but its cost can be predicted in advance, depending only slightly on the starting point.

Categories and subject descriptors: G.1.6. [**Numerical Analysis**]: Optimization—*nonlinear programming*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*probabilistic algorithms (including Monte Carlo)*; G.4. [**Mathematics of Computing**]: Mathematical Software—*certification and testing*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Global optimization, stochastic optimization, test functions

1. INTRODUCTION

The problem of determining the position in n-space of the minimum of a given function of n variables has been tackled using nonlinear programming methods for many years, in practice since digital computers have been available.

If the cost function is unimodal in the domain of interest, one can choose among many good alternatives. Some of the available minimization algorithms (direct methods) involve only function evaluations, such as those of Rosenbrock [16], Hooke and Jeeves [7], and Nelder and Mead [1, 11]. Others also use the evaluation of the derivatives of the cost function [4, 5, 17] and are considered to be more efficient than direct methods; but they are also more complicated and

Authors' address: Istituto per i Circuiti Elettronici-C.N.R, via all' Opera Pia, 11-16145 Genova, Italy.

more inclined to terminate far from the minimum if the cost function is very ill-conditioned.

However, if the cost function is multimodal within the domain of interest, the number of available algorithms is reduced to very few. In this case, the algorithms quoted above tend to stop at the first minimum encountered, and cannot be used easily for finding the global one. Since a systematic search in the function domain, in nontrivial cases, requires so many function evaluations as to be impracticable, one must rely on a limited search, either systematic or random.

A simple and widely used technique is to generate a given number of different points inside the function domain, performing unimodal searches starting from each of them, and retaining the best result.

Other methods for random global optimization are reported in [2, 3, 9, 13, 14]. Under mild conditions on the test functions, these stochastic methods guarantee *asymptotic convergence to the global optimum as the number of sample points* increases. All of these techniques are efficient in the case of functions with a few local minima; but, in practice, many optimization problems deal with a large number of variables (up to tens or hundreds) and/or a very large number of local minima that is often an increasing function of the number of variables. In this situation, traditional methods offer low efficiency and limited reliability.

Recently, a global optimization algorithm called Simulated Annealing (SA) [8] has been proposed in the area of combinatorial optimization, that is, when the cost function is defined in a discrete domain. This method is reported to perform well in the presence of a very high number of variables (even tens of thousands) [8, 15, 18]. It is based on random evaluations of the cost function, in such a way that transitions out of a local minimum are possible. It does not guarantee, of course, to find the global minimum, but if the function has many good near-optimal solutions, it should find one. In particular, this method is able to discriminate between "gross behavior" of the function and finer "wrinkles." First, it reaches an area in the function domain where a global minimum should be present, following the gross behavior irrespectively of small local minima found on the way. It then develops finer details, finding a good, near-optimal local minimum, if not the global minimum itself.

In the present work we propose some modifications to this algorithm, in order to apply it to the optimization of functions defined in a continuous domain. It is worth noting that these functions do not need to be smooth or even continuous in their domain. The algorithm is characterized by the need for a number of function evaluations, usually about three orders-of-magnitude greater than that commonly required for a single run of unimodal algorithms. Nevertheless, in the case of very ill-conditioned cost functions with thousands or millions of local minima, such big computational effort leads to better results than the random or systematic use of a traditional optimization algorithm for as many times as it takes to make the total number of function evaluations equal to SA. Moreover, the cost of automatic computation is becoming lower and lower, and such computational tasks are becoming affordable.

In Section 3, we introduce some multimodal test functions that are difficult to minimize, having a number of local minima greater than 10^{10} in their domain of interest. In Section 4, our SA algorithm is tested against the Nelder and Mead simplex method and against the "Adaptive Random Search" stochastic method. The test functions used are both the traditional Rosenbrock valleys and the multimodal functions quoted above. The results obtained are critically analyzed, and suggestions are made for future research.

2. METHOD

Let $\mathbf{x}$ be a vector in R^n and $(x_1, x_2, \ldots, x_n)$ its components. Let $f(\mathbf{x})$ be the function to minimize and let $a_1 < x_1 < b_1, \ldots, a_n < x_n < b_n$ be its n variables, each ranging in a finite, continuous interval. f does not need to be continuous but it must be bounded.

Our SA algorithm is schematically shown in Figure 1. It proceeds iteratively: Starting from a given point $\mathbf{x}_0$, it generates a succession of points: $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_i, \ldots$ tending to the global minimum of the cost function. New candidate points are generated around the current point $\mathbf{x}_i$ applying random moves along each coordinate direction, in turn. The new coordinate values are uniformly distributed in intervals centered around the corresponding coordinate of $\mathbf{x}_i$. Half the size of these intervals along each coordinate is recorded in the step vector $\mathbf{v}$. If the point falls outside the definition domain of f, a new point is randomly generated until a point belonging to the definition domain is found. A candidate point $\mathbf{x}'$ is accepted or rejected according to the Metropolis criterion [10]:

If $\Delta f \leqslant 0$, then accept the new point: $\mathbf{x}_{i+1} = \mathbf{x}'$
else accept the new point with probability:
$p(\Delta f) = \exp(-\Delta f/T)$

where $\Delta f = f(\mathbf{x}') - f(\mathbf{x}_i)$ and T is a parameter called temperature.

At a fixed value of T the succession of points $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_i, \ldots$ is not downhill, except when $T = 0$. For values of T large compared to the mean value of $|f(\mathbf{x}_h) - f(\mathbf{x}_k)|$ ($\mathbf{x}_h$ and $\mathbf{x}_k$ are points randomly chosen inside the definition domain of f) almost all new points are accepted and the succession is a random sampling of f.

The SA algorithm starts at some "high" temperature T_0 given by the user. A sequence of points is then generated until a sort of "equilibrium" is approached; that is a sequence of points $\mathbf{x}_i$ whose average value of f reaches a stable value as i increases. During this phase the step vector $\mathbf{v}_m$ is periodically adjusted to better follow the function behavior. The best point reached is recorded as $\mathbf{x}_{\text{opt}}$.

After thermal equilibration, the temperature T is reduced and a new sequence of moves is made starting from $\mathbf{x}_{\text{opt}}$, until thermal equilibrium is reached again, and so on.

The process is stopped at a temperature low enough that no more useful improvement can be expected, according to a stopping criterion that we will describe later.

The SA optimization algorithm can be considered analogous to the physical process by which a material changes state while minimizing its energy [8, 18]. A slow, careful cooling brings the material to a highly ordered, crystalline state of lowest energy. A rapid cooling instead yields defects and glass-like intrusions inside the material.

From an optimization point of view, an iterative search accepting only new points with lowest function values is like rapidly quenching a physical system at

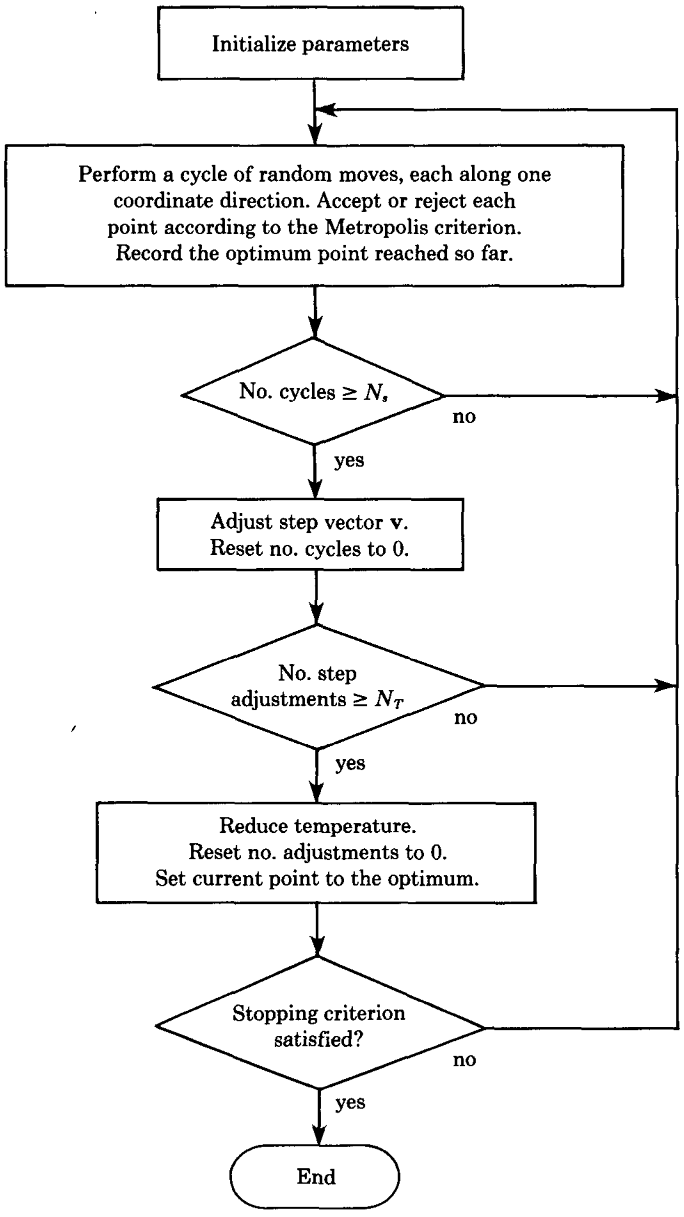

Initialize parameters
Perform a cycle of random moves, each along one coordinate direction. Accept or reject each point according to the Metropolis criterion. Record the optimum point reached so far.
No. cycles ≥ N_s
no
yes
Adjust step vector **v**. Reset no. cycles to 0.
No. step adjustments ≥ N_T
no
yes
Reduce temperature. Reset no. adjustments to 0. Set current point to the optimum.
Stopping criterion satisfied?
no
yes
End

Fig. 1. The SA minimization algorithm.

zero temperature: It is very likely to get stuck in a metastable, local minimum. On the contrary, SA permits uphill moves under the control of a temperature parameter. At higher temperature only the gross behavior of the cost function is relevant to the search. As temperature decreases, finer details can be developed to get a good final point. While the optimality of the final point cannot be guaranteed, the method is able to proceed toward better minima even in the presence of many local minima.

A detailed description of the algorithm follows.

Step 0 (*Initialization*)

Choose
- A starting point $\mathbf{x}_0$.
- A starting step vector $\mathbf{v}_0$.
- A starting temperature T_0.
- A terminating criterion ϵ and a number of successive temperature reductions to test for termination N_ϵ.
- A test for step variation N_S and a varying criterion $\mathbf{c}$.
- A test for temperature reduction N_T and a reduction coefficient r_T.

Set i, j, m, k to 0. i is the index denoting successive points, j denotes successive cycles along every direction, m describes successive step adjustments, and k covers successive temperature reductions.

Set h to 1. h is the index denoting the direction along which the trial point is generated, starting from the last accepted point.

Compute $f_0 = f(\mathbf{x}_0)$.
Set $\mathbf{x}_{\text{opt}} = \mathbf{x}_0$, $f_{\text{opt}} = f_0$.
Set $n_u = 0, \quad u = 1, \ldots, n$.
Set $f_u^* = f_0, \quad u = 0, -1, \ldots, -N_\epsilon + 1$.

Step 1

Starting from the point $\mathbf{x}_i$, generate a random point $\mathbf{x}'$ along the direction h:

$$\mathbf{x}' = \mathbf{x}_i + r v_{m_h} \mathbf{e}_h$$

where r is a random number generated in the range $[-1, 1]$ by a pseudorandom number generator; $\mathbf{e}_h$ is the vector of the hth coordinate direction; and v_{m_h} is the component of the step vector $\mathbf{v}_m$ along the same direction.

Step 2

If the hth coordinate of $\mathbf{x}'$ lies outside the definition domain of f, that is, if $x'_h < a_h$ or $x'_h > b_h$, then return to step 1.

Step 3

```
Compute $f' = f(\mathbf{x}')$.
If $f' \leqslant f_i$, then accept the new point:
    set $\mathbf{x}_{i+1} = \mathbf{x}'$,
    set $f_{i+1} = f'$,
    add 1 to $i$,
    add 1 to $n_h$;
    if $f' < f_{\text{opt}}$, then set
        $\mathbf{x}_{\text{opt}} = \mathbf{x}'$,
        $f_{\text{opt}} = f'$.
    endif;
```

else ($f' > f_i$) accept or reject the point with acceptance probability p (Metropolis move):

$$p = \exp\left(\frac{f_i - f'}{T_k}\right).$$

In practice, a pseudorandom number p' is generated in the range [0, 1] and is compared with p. If $p' < p$, the point is accepted, otherwise it is rejected.

In the case of acceptance:

```
set $\mathbf{x}_{i+1} = \mathbf{x}'$,
set $f_{i+1} = f'$,
add 1 to $i$,
add 1 to $n_h$.
```

Step 4

```
Add 1 to $h$.
    If $h \leq n$, then goto step 1;
else set $h$ to 1 and add 1 to $j$.
```

Step 5

```
If $j < N_S$, then goto step 1;
else update the step vector $\mathbf{v}_m$:
    for each direction $u$ the new step vector component $\mathbf{v}'_u$ is
```

$$v'_u = v_{m_u}\left(1 + c_u \frac{n_u/N_s - 0.6}{0.4}\right) \quad \text{if} \quad n_u > 0.6N_s,$$

$$v'_u = \frac{v_{m_u}}{1 + c_u \dfrac{0.4 - n_u/N_s}{0.4}} \quad \text{if} \quad n_u < 0.4N_s,$$

$$v'_u = v_{m_u} \quad \text{otherwise.}$$

```
Set $\mathbf{v}_{m+1} = \mathbf{v}'$,
set $j$ to 0,
set $n_u$ to 0,   $u = 1, \ldots, n$,
add 1 to $m$.
```

The aim of these variations in step length is to maintain the average percentage of accepted moves at about one-half of the total number of moves. The rather complicated formula used is discussed at the end of this chapter. The c_u parameter controls the step variation along each uth direction.

Step 6

```
If $m < N_T$, then go to step 1;
else, it is time to reduce the temperature $T_k$:
    set $T_{k+1} = r_T \cdot T_k$,
    set $f_k^* = f_i$,
    add 1 to $k$,
    set $m$ to 0.
```

It is worth noting that a temperature reduction occurs every $N_S \cdot N_T$ cycles of moves along every direction and after N_T step adjustments.

Step 7 (terminating criterion)

```
If:
    $|f_k^* - f_{k-u}^*| \leq \epsilon, \quad u = 1, \ldots, N_\epsilon$
    $f_k^* - f_{opt} \leq \epsilon$
then stop the search;
else:
    add 1 to $i$,
    set $\mathbf{x}_i = \mathbf{x}_{opt}$,
    set $f_i = f_{opt}$.
Go to step 1.
```

Reasonable values, found after some test optimizations, of the parameters that control the simulated annealing are

$$\begin{aligned} N_S &= 20. \\ N_T &= \max(100, 5 * n). \\ c_i &= 2, \quad i = 1, \ldots, n. \\ N_\epsilon &= 4. \\ r_T &= 0.85. \end{aligned}$$

Step Adjustments. In Monte Carlo simulations of fluids using the Metropolis approach, new configurations are generated, trying to maintain a 1:1 rate between accepted and rejected configurations [10]. A lower rate means that too many moves are rejected, thus wasting computational effort. A higher rate means that trial configurations are too close to the starting ones, thus having a small difference in energy compared to the temperature. This implies that the accepted configurations evolve too slowly, again wasting computational effort.

The same criterion is used by our SA algorithm. From an optimization point of view, a high number of accepted moves with respect to rejected ones means that the function is explored with too small steps. On the contrary, a high number of rejected moves means that new trial points are generated too far from the current point. A 1:1 rate between accepted and rejected moves means that the algorithm is following the "function behavior" well.

In our SA algorithm, trial points are generated along each coordinate direction in turn, independently from the other directions. A step vector $\mathbf{v}$ records the maximum increments possible along each direction and is adjusted every N_Sth move to maintain the 1:1 ratio quoted above.

For the sake of simplicity, let us focus on one coordinate direction, say u. The number of accepted moves along the u-axis since the last step adjustment is n_u. The ratio $n = n_u/N_S$ is therefore restricted to the interval [0, 1]. We define as $g(n)$ the coefficient to be multiplied by v_u in order to obtain the new step v_u' along the u-axis. If $n = 0.5$, the 1:1 ratio is verified as regards moves along the u direction, and v_u need not to be modified:

$$g(0.5) = 1.$$

If all the moves are accepted, $n = 1$. In this case the step length is multiplied by a factor c greater than 1:

$$g(1) = c.$$

For the sake of simplicity, and to preserve a certain degree of symmetry, in the case that all moves were rejected we decided to divide the step by the same factor c:

$$g(0) = 1/c.$$

Possible dependencies of g as a function of n are shown in Figures 2(a) and 2(b). The former is simply a piecewise straight line. The latter is a straight line for $n > 0.5$. In the interval $[0, 0.5]$, $g(n)$ is a hyperbola obtained by dividing 1 by the value of the corresponding straight line of Figure 2(a). We made some comparisons between these and other possible functions $g(n)$. Actually, the SA algorithm performance depends much more upon the value of c than on the function $g(n)$, provided that it is reasonably behaved and remembering that $g(0.5)$ must be equal to 1. Eventually, we decided to adopt the function shown in Figure 3, which corresponds to the formula given in Step 5 of the SA algorithm description, with the assumptions

$$n = \frac{n_u}{N_s}, \qquad c = 1 + c_u.$$

Using this formula, the step length is more stable than using the functions shown in Figure 2, and small deviations from the optimal 1:1 rate between accepted and rejected moves do not change the step length. Test runs performed in 2 dimensions showed that the minimum total number of function evaluations at each temperature needed to end in the global minimum is about 10 percent lower using the formula of Figure 3 than with the other choices. The starting temperature and the temperature reduction coefficient were the same for all tests. In 4 and 10 dimensions there was no significant difference between the formulas of Figures 2 and 3.

3. TEST FUNCTIONS

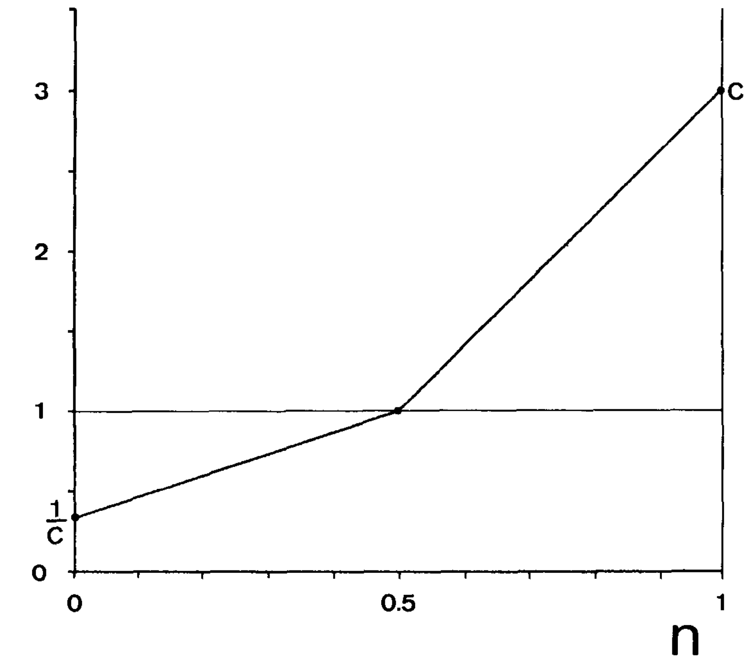
To test the simulated annealing algorithm described above for functions of continuous variables, we have used both classical, monotonic test functions and multiminima test functions appropriately constructed.

The first requisite of a global optimization algorithm is the capability to find the minimum of unimodal cost functions, although ill-conditioned and difficult to minimize. We have chosen the Rosenbrock classical test function [12, 16], in 2 and 4 dimensions:

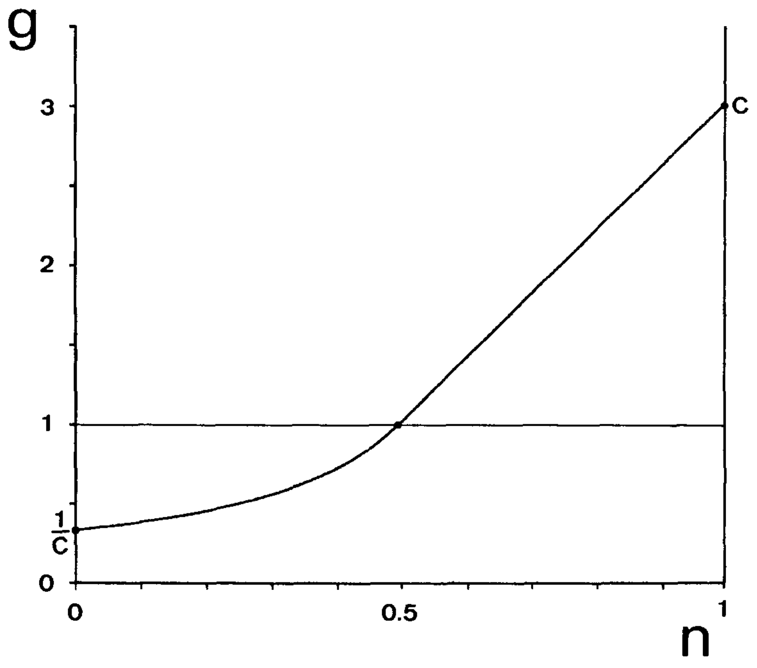
$$f_2(x_1, x_2) = 100(x_2 - x_1^2)^2 - (1 - x_1)^2,$$

$$f_4(x_1, x_2, x_3, x_4) = \sum_{k=1}^{3} 100(x_{k+1} - x_k^2)^2 - (1 - x_k)^2.$$

These curved valley functions are indeed a hard test for every minimization algorithm.



(a)



(b)

Fig. 2. Two possible multiplication factors $g(n)$.

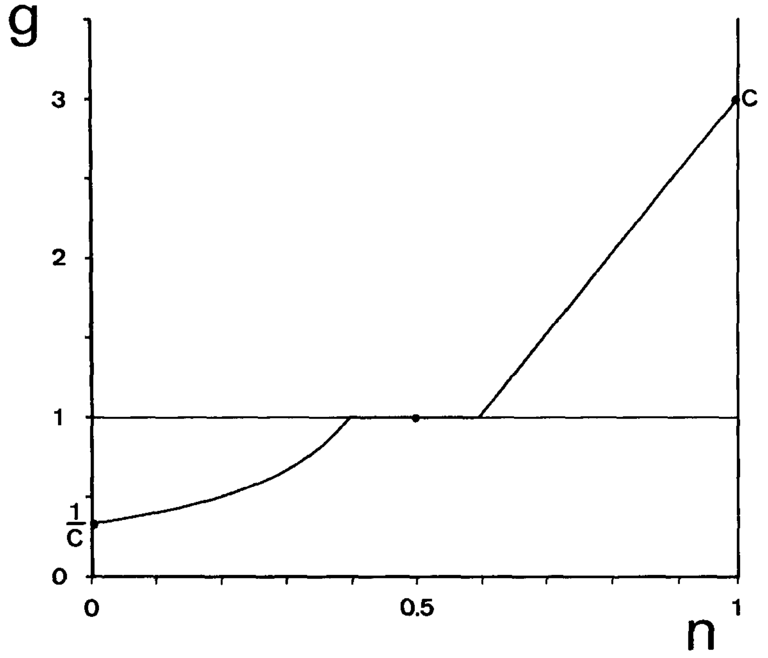


Fig. 3. The multiplication factor adopted in the SA algorithm presented.

Regarding multimodal functions, we have constructed a family of test functions q, defined in a limited domain, that are very simple to compute and contain a very high number of minima. These functions are obtained by defining a regular, rectangular grid in the R^n space and a set of open, nonoverlapping, rectangular subdomains, each centered around a node of the grid. The definition domain of the function $q_n(\mathbf{x})$ of n variables is a rectangular subdomain of R^n centered at the origin and including several nodes of the grid.

The function q_n is a paraboloid with axes parallel to the coordinate directions, except inside the open subdomains cited above, where it is constant with a value lower than the lowest value of q_n computed on the boundary of each subdomain. These subdomains are like a set of rectangular "holes," representing local minima of q_n and introducing strong discontinuities in the test function. The absolute minimum of the functions q lies in the origin and has value 0. The number of local minima depends on the dimension of the grid step along each axis and on the number of axes. The flatness of the function inside the domains, where it presents local minima, is also useful for lowering the function evaluation cost, since, once the function has been evaluated at a local minimum, subsequent evaluations relative to the same subdomain can be made at almost no extra cost. A formal definition of the generic test function q_n is the following.

Let D_f be the domain of definition of the function q_n in n-space:

$$D_f \equiv \{\mathbf{x} \in \mathbb{R}^n\colon -a_1 \leq x_1 \leq a_1, \ldots, -a_n \leq x_n \leq a_n;\ \mathbf{a} \in \mathbb{R}^n_+\}.$$

D_f is a rectangular subdomain of R^n, centered around the origin and whose width along each coordinate direction is determined by the corresponding component of the vector **a**.

Let D_m be the family of open, disjoint, rectangular subdomains of R^n contained in D_f and defined as

$$d_{k_1,\dots,k_n} \equiv \left\{ \mathbf{x} \in D_f: k_1 s_1 - t_1 < x_1 < k_1 s_1 + t_1, \dots, k_n s_n - t_n < x_n < k_n s_n + t_n; \right.$$
$$\left. k_1,\dots,k_n \in \mathbb{I};\ \mathbf{t}, \mathbf{s} \in \mathbb{R}^n_+;\ t_i < \frac{s_i}{2}, i = 1, \dots, n \right\},$$
$$D_m \equiv \bigcup_{k_1,\dots,k_n \in \mathbb{I}} d_{k_1,\dots,k_n} - d_{0,0,\dots,0}.$$

D_m is the open subset of D_f where the function q_n presents local minima. The vector **s** controls the grid steps along each axis, the grid points being the centers of these subdomains, while the vector **t** controls the size of these subdomains. The condition $t_i < s_i/2$ ensures that the subdomains are disjoint.

Let D_r be the closed subdomain of D_f complementary to D_m: $D_r \equiv D_f - D_m$. The test function $q_n(\mathbf{x})$ of n real variables is defined as

$$q_n(\mathbf{x}): D_f \to \mathbb{R}_+,$$

$$q_n(\mathbf{x}) \equiv \sum_{1}^{n}{}_i\, d_i x_i^2, \qquad \mathbf{x} \in D_r, \quad \mathbf{d} \in \mathbb{R}^n_+,$$
$$q_n(\mathbf{x}) \equiv c_r \sum_{1}^{n}{}_i\, d_i z_i^2, \qquad \mathbf{x} \in d_{k_1,\dots,k_n}, \quad (k_1, \dots, k_n) \neq 0,$$

where

$$z_i = \begin{cases} k_i s_i + t_i & \text{if } k_i < 0, \\ 0 & \text{if } k_i = 0, \\ k_i s_i - t_i & \text{if } k_i > 0. \end{cases}$$

The components of the vector **d** determine the steepness of the paraboloid along the axes, while the coefficient c_r controls the depth of local minima relative to the function values along the boundaries of the regions $d_{k_1,\dots,k_n}$. It is worth noting that the region $d_{0,\dots,0}$ belongs to the subdomain D_r, where q_n is equal to the paraboloid. This means that q_n is not constant around the origin, which is the unique global minimum of q_n.

The parameters that control the test function behavior are usually set to the following values:

$a_i = 10^4$,	$i = 1, \dots, n.$
$s_i = 0.2$,	$i = 1, \dots, n.$
$t_i = 0.05$,	$i = 1, \dots, n.$
$\mathbf{d} = (1, 1000)$	$n = 2.$
$\mathbf{d} = (1, 1000, 10, 100)$	$n = 4.$
$\mathbf{d} = (1, 1000, 10, 100, 1, 10, 100, 1000, 1, 10)$	$n = 10.$
$c_r = 0.15.$	

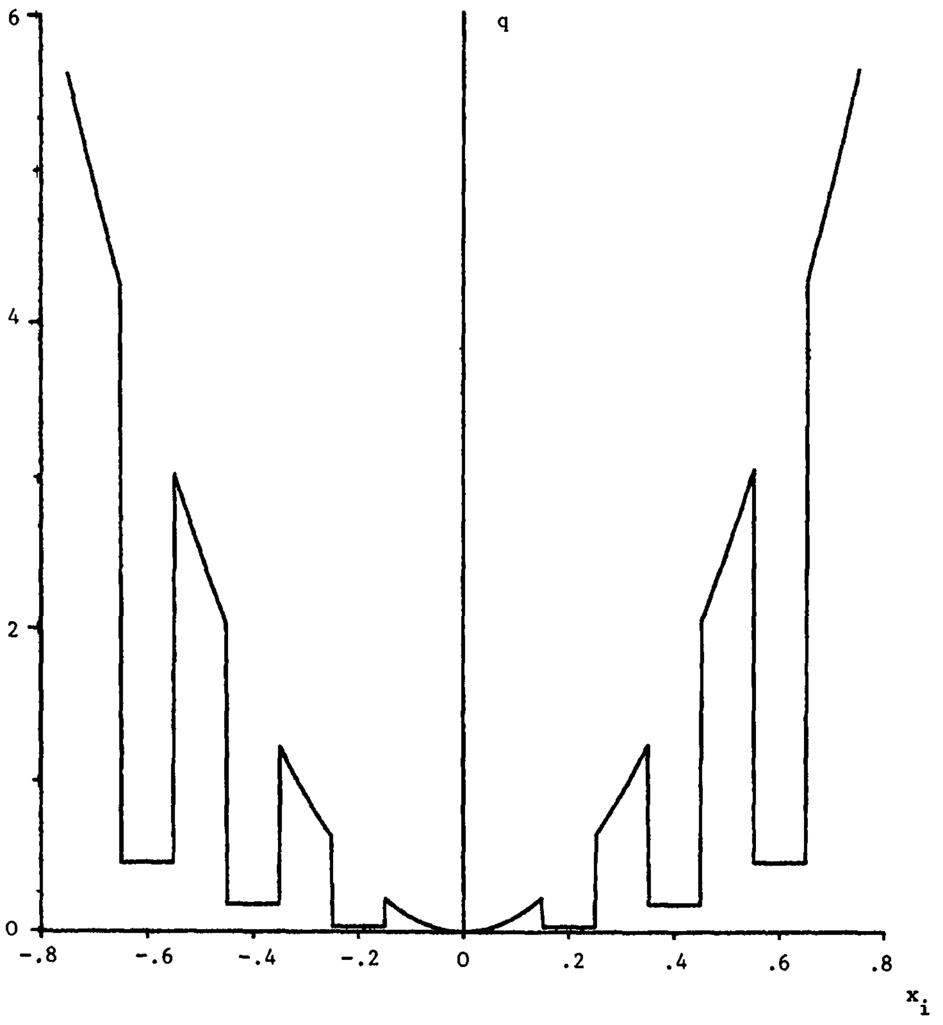


Fig. 4. A section of $q_n(x)$ along an axis i where $d_i = 10$.

The number of grid points along each direction is

$$n_g = 2\frac{a_i}{s_i}.$$

The total number of local minima of the test function $q_n(\mathbf{x})$ in its domain of definition is therefore $10^{5n} - 1$. For $n = 2$ we have 10^{10} minima and for $n = 4$ these minima are 10^{20}. A section of q_n along an axis where $d_i = 10$ is shown in Figure 4.

The number of local minima, their depth, and the strongly discontinuous behavior of the q functions are characteristics sufficient for quickly trapping any unimodal optimization algorithm at a local minimum and for severely testing any method for finding global minima.

4. TESTS AND RESULTS

We decided to test the effectiveness of the SA method against a well-known optimization algorithm, the Nelder and Mead simplex method [1, 11], and against a recently introduced general-purpose global optimizer using Adaptive Random Search (ARS) [9, 14].

The simplex method has been chosen owing to its efficiency and robustness compared with other direct-search methods. Moreover, the use of a gradient method would not yield any significant comparison, since the test functions defined in the previous chapter would immediately stop such algorithms at the first local minimum encountered.

The simplex method has proven to be very reliable. It is sometimes able to follow the gross behavior of the test functions despite their many local minima. This is also the reason why the local minima of our test functions are so deep: In some cases, a lesser depth does not prevent the simplex method from quickly finding the global minimum.

The ARS method has been implemented in the form proposed in [14]. It has been chosen because it is reported to be a simple, reliable, and efficient global optimizer. Moreover, a comparison with another global optimizer is needed to state the quality of the SA method.

The tests have been made starting from points randomly generated and quite far from the optimum, according to a test approach similar to the one proposed by Hillstrom [6]. The computer was a CDC Cyber 170/720 with 60-bit words. The tests took about 60 hours of computer time.

A first test of the SA method was made on the Rosenbrock functions in 2 and 4 dimensions. The admissible domains of the functions were $(2000, 2000)^{\oplus 2}$ and $(200, 200)^{\oplus 4}$, respectively. The SA algorithm and the comparison methods were started from different points randomly chosen and quite far from the origin. The starting temperatures of the SA were 50,000 and 10^7 for 2 and 4 dimensions, respectively. The results are reported in Table I.

As regards reliability, the SA never failed to reach the minimum of the functions. The simplex method in 4 dimensions stopped twice at a saddle point of the function. ARS seemed to be the least reliable of the three methods; in 2 dimensions it failed badly three times, and in 4 dimensions it stopped once at the saddle point mentioned above. It is worth noting that ARS, in these test runs on Rosenbrock functions, works with 10 different variances and not with 5, as suggested in [14] and used in the other test runs presented. This is because with 5 variances the algorithm failed in almost all runs on Rosenbrock functions.

The efficiency of the three algorithms has been measured by the number of function evaluations. The simplex method is by far the most efficient: On average, the SA takes from 500 to 1000 times more function evaluations than the simplex method. The number of evaluations needed by the ARS algorithm, on average, is half-way between the others. This number, however, greatly depends on the starting point and on the random number generator. In one case in 4 dimensions, the number of evaluations reached 29 million. This is due to the fact that the test functions are curved valleys with a bottom descending with a very mild slope. If ARS finds a point lying very close to the center of the bottom, it is possible to

Table I. A Comparison Between Three Optimization Algorithms on Rosenbrock Functions in 2 and 4 Dimensions

	Simplex			ARS			Simulated Annealing		
Starting point	Number of function evaluations	Final function value	Distance from optimum	Number of function evaluations	Final function value	Distance from optimum	Number of function evaluations	Final function value	Distance from optimum
2 Dimensions:									
1001, 1001	993	4.9E–10	4.07E–5	3411	1586.4	1508.	500001	1.8E–10	2.4E–6
1001, –999	276	7.4E–10	3.6E–5	131841	8.6E–9	2.08E–4	508001	2.6E–9	4.7E–5
–999, –999	730	2.7E–10	2.6E–5	15141	1.2E–8	2.6E–4	524001	1.2E–9	7.0E–4
–999, 1001	907	9.2E–10	2.72E–5	3802	583.2	631.5	484001	4.2E–8	2.0E–4
1443, 1	907	5.4E–11	7.2E–6	181280	4.7E–10	4.4E–5	492001	1.5E–8	2.3E–4
1, 1443	924	2.2E–10	3.1E–5	2629	1468.9	1393.	512001	1.6E–9	6.1E–4
1. 2, 1	161	2.4E–10	3.4E–4	6630	5.5E–7	1.6E–3	488001	2.0E–8	3.0E–4
4 Dimensions:									
101, 101, 101, 101	1869	3.70	2.1	519632	1.9E–6	2.8E–3	1288001	5.0E–7	1.2E–3
101, 101, 101, –99	784	5.46E–17	1E–9	194720	1.7E–6	2.7E–3	1328001	1.8E–7	6E–4
101, 101, –99, –99	973	9.8E–18	3.7E–9	183608	3.8E–6	3.9E–3	1264001	5.9E–7	1.3E–3
101, –99, –99, –99	1079	3.4E–17	4E–9	195902	2.3E–6	3E–3	1296001	7.4E–8	6.1E–5
–99, –99, –99, –99	859	8.3E–18	1.7E–9	190737	2.7E–6	3.3E–3	1304001	3.3E–7	1E–3
–99, 101, –99, 101	967	1.2E–17	2.4E–9	4172290	2.6E–6	3.3E–3	1280001	2.8E–7	1E–3
101, –99, 101, –99	870	6E–18	9.9E–8	53878	3.70	2.1	1272001	2.3E–7	3E–2
201, 0, 0, 0	1419	3.70	2.1	209415	1.1E–6	2.0E–3	1288001	7.5E–7	5E–4
1, 201, 1, 1,	1077	9.4E–18	1.9E–9	215116	1.2E–6	2.2E–3	1304001	4.6E–7	1.3E–3
1, 1, 1, 201	1265	3.9E–17	6.3E–9	29069006	2.2E–6	3E–3	1272001	5.2E–7	4.9E–4

proceed toward the minimum only by generating new points with a very small variance, thus very slowly.

The number of function evaluations needed to reach the minimum is quite constant in the SA, depending much more on the starting temperature rather than on the starting point. This fact is an obvious advantage, since one can forecast in advance the cost of an optimization run without imposing limits on the number of function evaluations. The starting temperature of the SA is quite different between 2 and 4 dimensions. These temperatures were found to empirically make the SA method able to find the minimum. If the starting temperatures had been set equal to the standard deviation of the test function in the domain of interest, as suggested by White [18], their values would have been about $2\ 10^{13}$ and $4\ 10^{13}$ in 2 and 4 dimensions, respectively. This would have resulted in an increase of the number of function evaluations of the order of 55 and 45 percent, respectively.

Summing up, the test runs performed on two very difficult functions show that the SA is the most reliable method among those under comparison. It is much more costly compared to the classical simplex method, but the number of function evaluations needed is quite constant and can be predetermined in advance.

A second test was made using our parabolic, multiminima, discontinuous test functions in 2, 4, and 10 dimensions. The SA algorithm was run ten times, starting from points located at a distance of about $1000\ \sqrt{n}$ from the origin, where n is the dimension of the cost function definition space; the admissible domains of the test functions were $(-10{,}000,\ 10{,}000)^{\oplus n}$. The results of these runs are reported in Table II. For each test function, the "s" and "t" parameters are shown, together with the starting temperature and the termination criterion of the SA algorithm. In 10 dimensions the "N_S" and "N_T" parameters of SA have been reduced with respect to the values reported in Section 2, resulting in a lower total number of function evaluations.

In 2 dimensions, SA never failed to find the global minimum at the origin. In 4 dimensions, in two runs out of ten, SA found a minimum value of 3.375E-3, which is the local minimum closest to the global one, and found the global minimum in the remaining eight runs. In 10 dimensions and ten test runs, SA always stopped at local minima valued 5.4E–4, which again are not the global minimum but are the local ones closest to the global minimum. The last two runs shown refer to the same starting point, with different seeds of the random number generator. In these test runs the number of function evaluations for each function is again practically independent of the starting point. Considering the difficulty of the test function chosen, this performance can be considered fully successful.

The other methods were run many more times, starting from points having components randomly generated inside the interval [−1000, 1000], up to a total number of function evaluations more than five times the average number of evaluations needed by SA to find the global minimum. In 10 dimensions the internal parameters of the simplex algorithm were modified relative to the original Nelder and Mead algorithm, according to the values reported in [1]. Using these values the simplex method behaved very well, also minimizing functions in 10 dimensions. The admissible domains of the test functions were $(-1000,\ 1000)^{\oplus n}$ for ARS, while the simplex method was run without constraints.

Table II. Results of Test Runs of SA Algorithm on Parabolic, Multiminima Functions in 2, 4, and 10 Dimensions

Starting point	Final function value	Function evaluation × 10^{-3}
$T_0 = 1E8$ $\epsilon = 1E-4$ $s = .2$ $t = .05$		
2 Dimensions:		
1000, 888	2.5E-8	684
−999, 1001	3.2E-9	680
−999, −889	4.0E-9	708
1001, −998	1.2E-9	696
1441, 3	3.2E-9	708
−10, −1410	4.0E-8	680
−1100, 850	1.2E-8	696
850, −1100	4.2E-10	656
4 Dimensions:		
−999, −999, −9999, −1000	2.6E-7	1440
999, 1000, 1001, −998	3.4E-3	1160
1000, −1000, 10000, −10000	8.7E-8	1464
−999, −999, −998, −1000	2.0E-7	1440
1000, 999, 999, 998	3.4E-7	1424
1000, −1000, −9999, 9999	2.5E-7	1416
1000, −1000, 998, 1000	3.4E-3	1176
0, 0, 1, 2001	4.0E-7	1408
1998, 3, 10, −13	4.6E-7	1408
1234, −1234, 560, −334	3.4E-7	1432

Starting point	Final function value	Function evaluation × 10^{-3}
$T_0 = 1E9$ $\epsilon = 1E-4$ $s = .1$ $t = .04$ $N_S = 15$ $N_T = 60$		
10 Dimensions:		
1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000	5.4E-4	1638
−1000, 1000, −1000, 1000, −1000, 1000, −1000, 1000, −1000, 1000	5.4E-4	1638
−999, −999, −999, −999, −999, −999, −999, −999, −999, −999	5.4E-4	1638
999, 999, 999, 999, 999, −999, −999, −999, −999, −999	5.4E-4	1548
−999, 1000, −999, 1000, −999, 1000, −999, 1000,−999, 1000	5.4E-4	1593
3000, 4, 20, 40, 120, −3, −6, 0, 0, 100	5.4E-4	1638
1000, −999, 1000, −999, 1000, −999, 1000, −999, 1000, −999	5.4E-4	1665
1000, −999, 1000, −999, 1000, −999, 1000, −999, 1000, −999	5.4E-4	1611

Table III shows the characteristics of the test functions used, together with the results of the minimizations. In 2 dimensions, the simplex method found the global minimum after 240 restarts and about 150,000 function evaluations, performing about four times better than SA. ARS spent five million evaluations without finding the global minimum. Its best point, however, was not too bad, considering the difficulty of the function and its average value in the considered domain.

In 4 dimensions, both algorithms were unable to find the minimum. In 10 dimensions, with a test function having parameter values $s_i = 0.1$, $t_i = 0.04$, $i = 1, \ldots, 10$, again both algorithms were unable to find the minimum. ARS found

Table III. Results of Test Runs of the Simplex and ARS Algorithms on Parabolic, Multiminima Functions in 2, 4, and 10 Dimensions

			Simplex			ARS		
Dimensions	$s.$	$t.$	Total number of evaluations	Number of runs	Best found value of F	Total number of evaluations	Number of runs	Best found value of F
2	.2	.05	150000	240	3.96E–10	5.E6	654	.084375
4	.2	.05	1E7	36121	278.7	1E7	940	192.6
10	.1	.04	2E7	27896	1265355	2E7	715	20332
10	.2	.05	1E7	6497	0.1549	2E7	6049	5222.5

a much better point than the simplex. In 10 dimensions, with a test function having local minima defined in a much smaller subset of the definition domain ($s_i = 0.2$, $t_i = 0.05$), the simplex method was able to find a very good point ($f = 0.1549$) after about ten million evaluations and 6500 restarts.

In all cases, except for the simplex method in 2 dimensions, SA proved much more reliable and efficient than the other algorithms.

5. CONCLUSIONS

We have presented a method for global minimization of functions of continuous variables based on simulated annealing. This method needs many function evaluations, but it is able to find the global minimum of test functions with an extremely high number of local minima. It seems unlikely that functions of this kind are easily found in real problems; nevertheless, SA can provide a very high reliability in the minimization of multimodal functions at high computational costs, linearly increasing with the number of dimensions of the problem.

Many problems, however, remain to be addressed and solved. The first one is the choice of the starting temperature T_0. In combinatorial SA, it has been suggested to use a value of the same order of magnitude of the standard deviation of the cost function in its domain of definition [18]. Dealing with continuous functions, this suggestion yields starting temperatures that are too high, thus wasting much computational effort. In fact, common cost functions of continuous variables often present terms varying with powers of the independent variables, thus leading to a rapid increase of the function toward the boundaries of its domain. This fact should not be significant for a good minimization algorithm, which should overcome in a few evaluations those parts of the function domain where it is varying steeply and monotonically. Moreover, this approach cannot be applied in the case of unconstrained minimization of continuous functions.

A better approach could be the monitoring of the function behavior as SA proceeds, using the incremental ratio between the average value of the cost function and of its square at the points accepted by the moves at a given temperature, as proposed by White [18]. This monitoring presupposes a Gaussian distribution of the function values in its definition domain, which again is not a common characteristic of continuous functions. Moreover, it has proven to be difficult to implement. More work is needed toward this goal, because a good monitoring of the minimization process can provide information about the goodness of the starting temperature, about the effectiveness of the number of function evaluations at each temperature, and about the time at which it is better to stop SA, perhaps starting a classical minimization algorithm. In this way it might be possible to save many function evaluations when the current point is close to the final minimum and the temperature is too low to allow escape from its region of attraction.

Further work is also needed to optimize the algorithm parameters, as we did for the simplex algorithm [1]. Such a study would be very costly, but many benefits would be gained.

Another open problem that does not appear with the cost functions used is the poor performance of the proposed algorithm when following multimodal cost functions having "valleys" not directed along the coordinate directions. This problem is due to the way new search points are generated. However, highly directional schemes might lose some of the flexibility of the random search procedure, as pointed out in [9].

The SA algorithm is commonly used in our Institute to address optimization problems not solved by conventional algorithms, and in the areas of electrical measurements and mathematical modeling. It is also used as a standard, to state the ability of traditional methods to solve particular classes of optimization problems.

Presently, work is proceeding on a version of the algorithm to run on parallel computers, thus reducing the major limitation of the method: its high computational cost.

REFERENCES

1. Barabino, G. P., Barabino, G. S., Bianco, B., and Marchesi, M. A study on the performances of simplex methods for function minimization. In *Proceedings of the IEEE International Conference on Circuits and Computers ICCC 80* IEEE, New York, 1980, 1150–1153.
2. Dixon, L. C. W., and Szego, G. P. (Eds.) *Toward Global Optimization.* North-Holland, Amsterdam, 1975.
3. Dixon, L. C. W., and Szego, G. P. (Eds.) *Toward Global Optimization 2.* North-Holland, Amsterdam, 1978.
4. Fletcher, R., and Powell, M. J. D. A rapidly convergent descent method for minimization. *Comput. J. 6* (1963), 163–168.
5. Fletcher, R., and Reeves, C. M. Function minimization by conjugate gradients. *Comput. J. 7* (1964), 149–154.
6. Hillstrom, K. E. A simulation test approach to the evaluation of nonlinear optimization algorithms. *ACM Trans. Math. Softw. 3*, 4 (Dec. 1977), 305–315.
7. Hooke, R., and Jeeves, T. A. Direct search solution of numerical and statistical problems. *J. ACM 7* (1969), 212–229.
8. Kirkpatrick, S., Gelatt, C. D., Jr., and Vecchi, M. P. Optimization by simulated annealing. *Science 220*, 4598 (May 1983), 671–680.
9. Masri, S. F., Bekey, G. A., and Safford, F. B. A global optimization algorithm using adaptive random search. *Appl. Math. Comput. 7* (1980), 353–375.
10. Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. Equation of state calculations by fast computing machines. *J. Chem. Phys. 21* (1953), 1087–1090.
11. Nelder, J. A., and Mead, R. A simplex method for function minimization. *Comput. J. 7* (1965), 308–313.
12. Oren, S. S. Self-scaling variable metric (SSVM) algorithms, part II: Implementation and experiments. *Manage. Sci. (Theor.) 20* (1974), 845–862.
13. Price, W. L. A controlled random search procedure for global optimization. *Comput. J. 20* (1977), 367–370.

14. PRONZATO, L., WALTER, E., VENOT, A., AND LEBRUCHEC, J. F. A general purpose global optimizer: Implementation and applications. *Math. Comput. Simul.* *26* (1984), 412–422.
15. ROMEO, F., SANGIOVANNI VINCENTELLI, A., AND SECHEN, C. Research on simulated annealing at Berkeley. In *Proceedings of the IEEE International Conference on Computer Design, ICCD 84*, IEEE New York, 1984, 652–657.
16. ROSENBROCK, H. H. An automatic method for finding the greatest or least value of a function. *Comput. J.* *3* (1960), 175–184.
17. SHAH, R. V., BUEHLER, R. J., AND KEMPTHORNE, O. Some algorithms for minimizing a function of several variables. *SIAM J.* *12* (1964), 74–92.
18. WHITE, S. R. Concepts of scale in simulated annealing. In *Proceedings of the IEEE International Conference on Computer Design, ICCD 84*, New York 1984, 646–651.

Received March 1986; revised January 1987; accepted April 1987