A Guided Monte Carlo Search Algorithm for Global Optimization of Multidimensional Functions[†]

Rupika Delgoda[‡] and James Douglas Pulfer*

Department of Biological Sciences, University of Warwick, Coventry, CV4 7AL UK

Received November 25, 1997

The high efficiency of the Monte Carlo optimization algorithm developed by Pulfer and Waine[14] is due to the discovery of a novel sampler that combines randomized guided step sizes with a random direction search strategy. We modified this algorithm to use a preset number of optimally sequenced steps to bound the randomly chosen step length. This has the effect of both spanning the response surface rapidly and escaping local optima efficiently. Coupled to changes in both sampling strategy and termination criteria, the resulting guided Monte Carlo (GMC) numerical search algorithm is shown to solve the global optimization problem effectively. Fifteen multidimensional benchmark test functions having differing characteristics such as numerous local optima or very sharp optima, very shallow optima, variables with differing influence over the function, and high dimensionality, were used to test the efficacy of the GMC algorithm. It was successful in solving them all, with a majority converging 100% of the time out of 1000 independent runs within highly competitive computer processing times when compared to contemporary efficient algorithms. For example, when the highly intractable five dimensional shekel function was solved by Fagiuoli et al.'s[20] sampling algorithm, it required 2514 function evaluations (f.e.) and 7 shekels of computer time to find the global optimum with a success rate of 900 out of 1000 independent runs, whereas the GMC algorithm needed only 519 f.e. and 1.66 shekels to achieve the same accuracy with the same probability of success. A multirun routine has also been incorporated into the GMC algorithm to enable users to repeatedly test the response surface to achieve almost 100% certainty. Also, the GMC algorithm successfully solved the 100 dimensional cosine mixture test function, known to have numerous shallow local optima and one global optimum. This indicates its potential to solve practical problems such as those associated with protein configuration analysis.

1. INTRODUCTION

An optimization task must vary a set of independent parameters, often subject to constraints, in such a way that the value of the dependent output function is either maximized or minimized, i.e., the objective function is forced to assume an optimal value.[1]

Problems in optimization occur in all areas of mathematics, applied science, engineering, economics, medicine, and statistics. Apart from its major role in decision-making processes, it is often called upon to solve critical subproblems within other generalized numerical algorithms. The situation is so common that the very existence of an optimizing step may be taken for granted and pass unremarked.

The objective function is a mathematical relationship expressing an output or result in terms of system factors and/or responses. It may not always be available. The design of the objective function itself then becomes another type of optimization problem.[2] Models are developed to analyze and understand complex phenomena or to express all responses in a desired manner. In this context, optimization is used to determine the model that best represents a measured or experimental reality. Throughout this paper, the term optimization will be used in association with computing an optimal solution of the objective function and, since the aim is to devise solution techniques, the existence of a function in standard form will be assumed.

1.1. Problem Formulation. An unconstrained[3] optimization problem in real space can always be expressed as follows: given a real valued objective function $f(\mathbf{x})$ defined on the number space R^n where $\mathbf{x}$ is an n-dimensional vector, find a vector $\mathbf{x}^*$ such that $f(\mathbf{x}^*)$ is always less than or equal the functional value at any other point in the space, i.e.

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \ \forall \ \mathbf{x} \in R^n$$

or equivalently,

$$f^* = \min f(\mathbf{x})$$

If there are predefined constraints on the variables such as, for example, in a monetary problem where the currency variables must all be zero or positive, then the problem alters to that of finding

$$f^* = \min\{f(\mathbf{x})|\mathbf{x} \in D\}$$

where D is the domain of interest.

* To whom all correspondence should be addressed. E-mail: mssed@csv.warwick.ac.uk fax: +44 1203 523568.

† Paper presented at the American Association for the Advancement of Science Annual Meeting & Science Innovation Expo, AMSIE'95, Atlanta, Georgia, February 16−21, 1995.

‡ Present address: University of Oxford, Department of Pharmacology, Mansfield Road, Oxford OX1 3QT UK. E-mail: rupika.delgoda@linacre.oxford.ac.uk.

1.2. The Global Optimization Problem. The minimum found, $\mathbf{x}^*$, as defined above, is the global minimum, i.e., the value of the vector $\mathbf{x}$ that gives the lowest functional value within the region of interest. However, within the domain, there could exist a second type of minimum termed a local minimum[3] $\underline{\mathbf{x}}$ where there exists a neighborhood B of $\underline{\mathbf{x}}$ with

$$f(\underline{\mathbf{x}}) \leq f(\mathbf{x})\ \forall\ \mathbf{x} \in B$$

Objective functions of real applications usually have several minima; however, the common aim is to quickly and reliably detect the global one. This is especially true for technical design problems, economic analysis, and in many areas of scientific investigation.[4] The difficult problem of developing a numerical solution that will efficiently distinguish between a plethora of local minima and the global one is known as the global optimization problem (GOP).[3] Because of its difficulty and importance, numerous algorithms have been developed in an attempt to solve it.[3,4] The difficulties arise from several factors, many of which act in concert to compound the complexity of the solution. For example, if the parameters are experimentally determined, then each is subject to error. This must be taken into account when establishing the bounds of the global solution. If the function experiences about the same degree of sensitivity to many of these parameters and if the function happens to have shallow, wide minima near the global optima, then the solution may oscillate between one of several sites that are quite far apart. This leads to an inherent instability, making it exceedingly difficult to pinpoint with any certainty where the correct global optimum point might be.

1.3. Difficulties in Global Optimization. 1. **The Size of the Variable Space**. Many of the functions that need optimization are multidimensional. If an algorithm cannot handle a response surface influenced by more than one variable, then its use will be very limited.

2. The Continuity of the Response Surface. All methods which use derivatives require a smooth, continuous function with finite derivatives[4] up to the degree specified by the optimization method. Singularities at any point will rule out the use of such methods.

3. The Presence of Local Optima. (a) The function may possess more than one local optima and often many more. The global minimum could be nested within a plethora of local minima, thereby making its detection extremely difficult.

(b) The nature of the local optima will also affect certain algorithms. Although, some are able to find global optima that are shallow in nature, they may not be capable of detecting ones that are very sharp or *vice versa*.

4. The Sensitivity of the Objective Function to Each Variable.[5] Some methods can be severely hampered if the function is overly sensitive to one variable. The optimization tends to spend most time getting that parameter right while ignoring others.

5. Time and Convergence Rate. The algorithm must be able to handle all above features within reasonable computer time. Hand-in-hand with that goes the ability to ensure convergence. The sampling of the surface should be done so that within reasonable time there is some certainty of success.

The influence that these factors have on each other make the GOP a formidable task. Having an algorithm to solve one problem may be a hindrance in another. For example, the discontinuity problem has been overcome by stochastic methods. However, they do not necessarily guarantee convergence within a finite search time.

1.4. Global Optimization Methods. Traditionally, algorithms attempting to solve the GOP have been broadly classified as either deterministic or probabilistic.

Purely deterministic algorithms attempt to guarantee that a neighborhood of the global minimum will be located. The two commonly used classes of such algorithms are conjugate gradient methods,[6] as typified by the Fletcher–Reeves and Polak–Ribiere algorithms, and quasi-Newton methods, such as Fletcher–Powell or Brieman and Cutler.[7] Both utilize either first and/or second derivative information. While this guarantee of success is undoubtedly an asset,[8] deterministic methods are restricted to classes of functions which are continuous and differentiable up to the order required. Also, optimizers which require the first and/or second derivative may suffer from numerical inaccuracy and will usually be slowed by this requirement.

The alternative approach is to design algorithms which use stochastic methods to predict the probability of locating a global minimum within a defined region. These usually do not require a priori information about the topology of the function or its derivative(s) and are amenable to multidimensional functions. Ranging from the classical pure random search methods, many sophisticated probabilistic methods, such as the clustering technique of Rinnoy Kan et al.,[3] have been devised. However, a common criticism of all such methods is that they are slow to converge. This arises from the attempt to *guarantee* results. Thus, a tradeoff occurs between speed of convergence and probability of success.

Of all such methods, simulated annealing[9] (SA) has shown the most promise to date in avoiding being trapped in local optima and, hence, it grants some assurance of convergence. Local minima are overcome by the “hill-climbing” capacity of SA when combined with the Metropolis criterion. The practice is to sequentially move from one position to another and accept all moves that give lower functional values and some that generate higher ones. This latter option allows the search to escape local minima by climbing uphill. An uphill move is accepted only if the Metropolis criterion is satisfied, i.e., if the probability p

$$p = p_{\mathrm{T}}(\text{new move})/p_{\mathrm{T}}(\text{previous move}) = \exp\{-\Delta E/kT\}$$

is greater than a random number q generated on the interval [0, 1]. Here ΔE is the difference between the new functional evaluation and its previous one, i.e.

$$\Delta E = E(\text{new}) - E(\text{previous})$$

Various versions of SA[10,11] change the proportion of acceptance of uphill moves and step sizes as the search proceeds to speed up the algorithm, using large step sizes at the beginning of the search and small step sizes to avoid missing the global optimum at the end, and to make the acceptance criterion more efficient when moving through local minima. Recently, simulated annealing has been criticized[12] on the grounds that the Metropolis criterion may trap the search in a local minimum due to the precision limit of the computer. Apart from the possibility of trapping the search in a local minima, the hill climbing routine is relatively inefficient because it has no mechanism to avoid unnecessary encounters with numerous local minima. Also, the use of variable step sizes has given rise to certain inefficiencies whereby the routine tends to revisit local optima too often or the steps are made too small near the global point. Should these drawbacks be alleviated, then the way will be opened for developing the ideal probabilistic algorithm.

Just such a modified Monte Carlo optimization algorithm has been developed by Pulfer and Waine.[14] By using a real valued functional guide to help limit the random step size, together with random directional cosines when sampling the multidimensional response surface, the search was able to effectively sample the range space and escape local optima without recourse to the hill climbing effort. The result was an algorithm which rapidly moved to the region of the global optimum in relatively few steps. To improve on this algorithm, a better termination criteria was sought to bring it to a more efficient stop and to increase the probability of success. Also, because it was unclear as to why the randomized guided step sizes were proving to be such an efficient sampler of the response surface, a numerical study of that phenomenon was initiated. The result of this inquiry led to the possibility that only 21 step size guides need be employed when sampling the space!

2. AIM

The ultimate aim for developing an algorithm is to use it to solve a practical problem of interest. Yet for the purpose of investigating and testing an algorithm, it is difficult to use applied functions because they usually prove to be expensive to evaluate and secondly are not freely available to all. Thus, to get around these barriers, an artificial set of test functions has been designed that will simulate all the features of any real problem. The most popular test functions are those given in Dixon and Szegö.[13] Well over 30 algorithms[14,15] from the late 1970s to the present have been tested and compared using them. Accordingly, it was decided that these functions should be used for evaluating the GMC algorithm along with several other widely accepted functions. All functions were selected to ensure as comprehensive and fair a test as possible. Since test functions are designed to incorporate as many problematic features as possible and, in so doing, magnify versions of the problems faced in real applications, they ought to be a good judge of the capabilities of the algorithm.

3. THE GUIDED MONTE CARLO (GMC) ALGORITHM

The guided Monte Carlo (GMC) algorithm is equipped with a strategy that quickly spans the entire space to be certain that the global optimum has not been overlooked yet thoroughly and inexpensively samples the regions of most promise as it turns up the global point. The search begins by setting all of the working variables $x(i)$ in the vector $\mathbf{x} = (x(1), x(2),..x(n))$ to their starting coordinates that can be either arbitrary if no prior information is available about them or to greater degrees of closeness to the global optimum point depending on the level of knowledge about the system. These $x(i)$ are used to calculate the value of the objective function, $f(\mathbf{x})$, labeled fx for convenience.

Table 1. The Step Size Guides

number of the frustration counter (*fcount*)	step magnitude (arbitrary units) *step*(*fcount*)	number of the frustration counter (*fcount*)	step magnitude (arbitrary units) *step*(*fcount*)
0	90.0	11	1.40625
1	90.0	12	0.703125
2	90.0	13	0.3515625
3	90.0	14	0.1757812
4	45.0	15	0.0878906
5	22.5	16	0.0439453
6	11.25	17	0.0219726
7	5.625	18	0.0021972
8	2.8125	19	0.0002197
9	1.40625	20	0.0000219
10	1.40625		

A solution vector b is defined where the $b(i)$ are initially unknown and, at the beginning of each run in the search, are set to a small perturbation near $x(i)$

$$b(i) = x(i) + \delta$$

where δ is a random number selected from the range [0, 1]. During the search, whenever fx becomes less than fb, the better position is saved by setting $b(i)$ to $x(i)$ and fb to fx. Lastly, at the end of the search, the best $b(i)$ is reported.

New search coordinates are generated by randomly stepping away from the current best position. At each move of the search procedure, if the $x(i)$ selected do not lead to an improvement in fx, then this frustration is recorded in *fcount*. The maximum possible magnitude of the next move will partly depend on the state of the search, dictated by the frustration level of *fcount*, where *fcount* selects a particular guide value, as shown in Table 1. Hence at early stages, large numbers are used to help randomly sample a large area rapidly, while later on sampling will occur quite close to the current position. The direction of the step is also randomized in a way that is equivalent to using direction cosines.[14] Once a random step magnitude rx and direction cx have been calculated, they can be used to find the new positions of all the search parameters $x(i)$ relative to the current best $b(i)$ from the equation

$$x(i) = b(i) + rx^*cx$$

where the randomized step size rx is a random number selected from the range [0, *step*(*fcount*)] and cx from the range [−1, 1], i.e.

$$rx = rnd()^*step(fcount)$$

$$cx = rnd()^*2 - 1$$

When the search has encountered 21 frustrated moves ($fcount > 20$), it indicates that the problem at hand cannot be solved by simple changes in the step magnitude. To help alleviate this and to come to terms with the varying sensitivity of the function toward individual variables, sampling of each parameter begins. Thus, the *istep* counter is incremented from zero, indicating that only one variable at a time will change, in this case the first, $x(1)$, leaving all others at the best positon achieved so far in the search. When it transpires that 21 frustrated moves is encountered, once more, *istep* is incremented again and the next variable in sequence is

Table 2. Test Functions Employed

identifying symbols	function	optimizing parameters
A[17]	$TC = 20 + 0.8x_1 + 0.8x_2 + 0.022x_1x_2 - 0.015x_1^2 - 0.015x_2^2$	max (100,100)
B[10]	$\cos X_1X_2 = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3$	min (0,0)
C[10]	$\cos X_1 + X_2 = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1 + 4\pi x_2) + 0.3$	min (0,0)
D[10]	$\cos_n = \sum_{i=1}^{n}\{ix_i^2 - [(i+2)/10]\cos((i+2)\pi x_i) + (i+2)/10\}$, $[-1,10]$	$\{\min \forall x_i = 0\}$
E[14]	$\mathrm{SHEKEL}_n = \sum_{i=1}^{n} -\{1/[(a_i - x_i)^2 + c_i]\}$, $c_i = 10^{2-i}$, $[0,12]$	$\{\min \forall x_i = 2i\}$
F[7]	$\exp_n = \exp(-0.5\sum_{i=1}^{n}\{x_i^2\})$, $[-1,1]$	$\{\max \forall x_i = 0\}$
G[7]	$\mathrm{BC\_cos}_n = \sum_{i=1}^{n}\{0.1\cos(5\pi x_i) - x_i^2\}$, $[-1,1]$	$\{\max \forall x_i = 0\}$
H[15]	$\mathrm{Ras}_n = \sum_{i=1}^{n}\{x_i^2 - \cos(18x_i)\}$, $[-1,10]$	$\{\min \forall x_i = 0\}$
I[24]	$T1_n = \sum_{i=1}^{n}\{x_i^4 - 16x_i^2 + 5x_i\}$, $[-3,3]$	$\{\min \forall x_i = -2.904\}$
J[13]	$S_m = \sum_{i=1}^{m} -\{1/[(x - a_i)^T(x - a_i) + c_i]\}$, $[0, 10]$	
K[13]	$BR = x_2 - (5.1/4\pi^2)x_1^2 + (5/\pi)x_1 - 6)^2 + 10(1 - (1/8\pi))\cos x_1 + 10$, $-4 < x_1 < 10$ $2 < x_2 < 13$	
L[13]	$H_n = -\sum_{i=1}^{m} c_i \exp(-\sum_{j=1}^{n} a_{ij}(x_i - p_{ij})^2)$, $0 < x_i < 1$	
M[13]	$GP = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$, $-2 \le x_i \le 2$	
N[8]	$HUMP = [4 - 2.1x_1^2 + (x_1^4/3)]x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$, $-1 \le x_i \le 1$	
		min (0.08983, −0.7126)
		min (−0.08983, 0.7126)
O[8]	$T2 = 0.5x_1^2 + 0.5[1 - \cos(2x_1)] + x_2^2$	min (0,0)

independently searched. At the end of a whole set of variables being independently shaken, the termination parameter *iroc* is incremented, signaling the end of a set of variables being rocked. The search resumes, simultaneously changing of all variables once more as before. This routine iterates until *iroc* reaches 3, at which time the search resumes, simultaneously changing all variables except that in this case, as soon as a frustrated move is encountered, the search will assume that it is at the global minimum and termination occurs.

3.1. The Algorithm. This algorithm is set up to find a global mimimum.

1. Initialize $\mathbf{x}$ and set $\mathbf{b}$ to $\mathbf{x} + \delta$ where δ is an n-dimensional vector of random numbers selected from the range [0, 1]. Set $f(\mathbf{b})$ or *fb* to an arbitrarily large number (effectively infinity).

2. Calculate $f(\mathbf{x})$; i.e., *fx*.

3. Compare *fx* and *fb*.

(a) if $fx \le fb$, then

(i) set $b(i) = x(i)\ \forall\ i$;

(ii) set $fb = fx$;

(iii) set frustration counter, *fcount* to 0;

(iv) go to 5.

(b) if $fx > fb$, then increase *fcount* by 1, i.e.,
$fcount = fcount + 1$

4. Check all frustration (*fcount*), parameter (*istep*), and termination (*iroc*) markers:

(a) if $fcount > 20$ and *iroc* is 3, then terminate;

(b) if $fcount > 20$, then

(i) $istep = istep + 1$, i.e., change only the *istep* variable;

(ii) $fcount = 0$.

(c) if $istep > n$, then a complete set of variables have been independently sampled and

(i) $iroc = iroc + 1$;

(ii) $istep = 0$, to simultaneously search all variables once more;

(iii) $fcount = 0$.

5. Generate new coordinates for the next iteration.

(a) if $istep = 0$, then search all variables simultaneously as

$x(i) = b(i) + rx{*}cx\ \forall\ i$, where $rx{*}cx$ is the same for each i and calculated by

$rx = rnd()*step(fcount)$, rx randomly selected from $[0, step(fcount)]$

$cx = rnd()*2 - 1$, cx from the range $[-1, 1]$.

(b) if $istep > 0$, then search only $x(istep)$, freezing the rest in their best positions: $x(istep) = b(istep) + rx{*}cx$, where rx and cx are calculated each time x(*istep*) is called.

6. Go to 2.

7. Continue until condition 4(a) is met, then

(a) if *runs* > a preset number defined by the user, then calculate the average of those $b(i)$ for each run that fall within a user-defined precision and report this as the global optimum and go to 1.; otherwise

(b) report the best averaged $b(i)$ and terminate the search.

3.2. Discussion of the Algorithm. There are three essential features of this algorithm that distinguish it from the Pulfer−Waine[14] optimizer: the use of only 21 fixed guides to calculate the randomized step sizes; the strategic use of *fcount* to limit and sandwich simultaneous and individual variable searches; and the use of the same $rx{*}cx$ value when positioning all the $x(i)$ during the simultaneous search phase.

3.2.1. The Guide Function Values. Both the values and the sequence of the guide set resulted from an exhaustive numerical study of the 15 functions listed in Table 2. Several features can be spotted in the guide set shown in Table 1. First, at the beginning, the largest guide number is repeated four times. The fact that it is nearly equal to the 12th number of the Fibonacci sequence may be important. Then the values are drastically reduced by halving them after each frustrated attempt until they reach the neighborhood of $\sqrt{2}$, at which point that value is maintained for three moves. After that, the halving process resumes until the frustration counter reaches 17, at which time, the final three guide numbers each undergo a dramatic reduction by a factor of 10. The variation in size from the largest value to the eighteenth is $(1/2)^{12}$. This severe partitioning of size is one reason why the guide values are so effective at spanning the space. The last three numbers are there mainly to improve the solution in the immediate neighborhood of the optimal point or to refine the position of a variable that contributes significantly to a function value.

The first four numbers are thought to be large and the same because that is the arrangement which will most quickly move through the whole of the range space. Of the four, on a statistical basis, two will be given positive random number assignments and two negative, and, of the two positive (or negative), one will be in the right half of the space and the other in the left. This course partitioning is sufficient to ensure that little effort is devoted to a complete search. Then, too, the three numbers clustered at $\sqrt{2}$ are thought to be significant because the Fibonacci sequence has three integer values between 1 and 2 with an average size similar to $\sqrt{2}$. The similarity of the sequence to Fibonacci's may be important because it can be readily shown that for a well-behaved concave function, the Fibonacci search will be the most efficient at locating the minimum point.[25]

Apart from these theoretical considerations, there are several practical reasons for the way the step guide sequence was developed. First, the total number of frustrated moves allowed in the set was chosen to be 21 because this number maintains a favorable balance between precision and computational effort. Termination criteria of various versions of simulated annealing are also based on between 20 and 30 successive frustrated moves (although one uses 50!). These SA criteria have been used to demonstrate that such levels of frustration are usually ample when seeking to gain a more favorable position in reasonable time. Second, the search needed to be spear-headed by several large numbers so that reasonably large random step sizes can be achieved in order to improve the chance of stepping over many unfavorable regions while at the same time sampling them thoroughly. Third, by successively halving the numbers, the values reduce drastically and quickly, and thus allow a rapid spanning of the response surface. The maintenance of the values around $\sqrt{2}$ was first found by Pulfer and Waine[14] and turns out to be crucial when solving the Rastrigin function.[15] Since there are ~50 minima arranged in a lattice configuration within a couple of units of the optimal point, a successful solution requires repetitive numbers in this region. Also, after the first four large numbers, the successive halving of the values is a trait required or found efficient by almost all the test functions analyzed. Under such a sequence, almost all converged 100% of the time in a minimum number of moves, with the lone exception of the shekel function, which showed a success rate of ~90%. Fourth, by the time the guide numbers reach ~0.1, the search has been frustrated 16 times. On the assumption that large- to medium-size numbers have not worked, the sequence then changes to much smaller values. It turns out that these numbers are critical for very small regions of attraction such as are found in the shekel function or for obtaining accurate results.

As mentioned, large numbers were placed at the top of the sequence with the hope of covering as much of the surface as possible. Small step sizes were not introduced there because in some cases it results in early stagnation if very sharp local optima are present near the global point. For example, small steps at the early stages are detrimental when solving the Ras_2 and T1 functions even though the shekel family prefer them.

Since different functions showed various needs at different stages of their search, an attempt was made to design a procedure that could offer appropriate step sizes as the search evolved. This would incorporate an intelligence feature based on prior experience or prior identification of the topology of the function. A learning strategy, based on prior steps that the search had made use of, was tried out. However, this technique proved unfavorable, at least for the type of Monte Carlo search carried out here, because the contours of the test function can vary so rapidly that past learning proves to be of no use in predicting future requirements. Also, trial steps can be detrimental to some functions. Should, for example, small numbers be offered to the Ras_2 function early on, it would get stuck at one of the many sharp optima. There is no way of knowing what steps to try. Additionally, these trial steps consume time. It has been found to be far better to use a fixed general pattern which incorporates a quasi-intelligent guide set capable of changing its sampling ability to some degree. The current algorithm appears to possess that feature, since it offers steps of a given size at a given level of frustration in such a way that it efficiently satisfies the solution requirements of a wide diversity of test functions.

One vital finding of these investigations is that since the step size values are only a guide and randomization produces the actual step magnitude, other guides can also be used as long as the general pattern and range compared to the full range space is maintained. For example, we have found that a modified Fibonacci sequence works reasonably well. Also, when we fitted these guide numbers to several widely differing functions using a simplex optimization, we got values that performed quite well which were significantly different from those reported here.

3.2.2. Sampling Strategy. It was noted that for functions like the shekel family, where the global optimum lies within a region of attraction that is quite small, simultaneous but random fluctuations of all the variables at the same time is very detrimental. Because step magnitudes can vary significantly between each parameter, the small region of attraction can easily be missed. This realization led to a sampling strategy that uses the same random step size and direction for all the variables when simultaneously varying them (*istep* = 0). Then, when each variable is being shaken with the others frozen in their best positions (*istep* > 0), a new randomized step size and direction can be used each time a new position is sought.

This strategy is permissible because each variable is independent and thus, when presented with an independently picked step size and direction cosine, makes no distinction as to whether that pick might be better or worse than the next during its simultaneous positioning in the search for the global optimum (*istep* = 0). In this way, the search exerts a degree of control over the movement of the variables in different step sizes and directions so that it does not focus on only the most sensitive parameters when *istep* = 0, yet couples the search with enough independence to ensure that all of the variables, including the slackest, have a chance to move to their global optimum position (*istep* > 0). To a degree, similar sampling is seen in the constrained global optimization algorithm of Altschuler et al.[18]

3.2.3. Termination Criteria. Termination is based on two markers that are linked: *fcount* and *iroc*. A strong signal that the sampling strategy must be changed or terminated occurs whenever 21 frustrated moves are encountered in succession. Initially, all the variables are simultaneously varied and it takes quite some time before *fcount* exceeds 20. While this is happening, the search usually moves to the most promising neighborhood of the global optimum. Then, to fine tune the individual variables, each is rocked independently, only stopping when *fcount* > 20 in each case; then, back to a simultaneous search to be sure that any coupled parameters have a chance to move relative to each other to a better optimal point, and so it goes. When this process has been repeated three times (*iroc* = 3), the stopping criteria is triggered and the process terminates when the last simultaneous search encounters 21 frustrated moves. Termination takes place only after the simultaneous search because of the possibility that the movement of one or more variables may govern how they all interact with each other and that, in turn, could affect the global optimum of the function. If termination occurred after the set of variables had been shaken one by one, it would be the same as finding the partial optima with respect to each. The higher the number of iterations permitted by *iroc*, the greater the chance of success. However, the penalty to be paid is an increase in computational effort. To strike a balance between the two, *iroc* was set to three.[14] This was found to combine a thorough search with least computational effort.

3.2.4. The Multi-Run Option. In some instances, especially for the shekel functions, a further modification was incorporated into the algorithm to improve the chance of success. This involves the execution of three or more independent runs of the routine, with each subsequent run beginning near where the previous one ended. The results of each run are stored for later comparison and, if acceptably close together, are included into an average value which will be the best global optimum position for each variable. Every variable is compared with corresponding ones from all the other runs. Should any one of them fall outside a user-defined tolerance limit, it is discarded and the user is informed that this is being done. Hence, only trials that have comparisons within the required tolerance limit will be accepted and averaged to find the best global optimum position. The routine is based on the assumption that the outliers are in a distinct minority and accepts only variables from each run that are comparable. Furthermore, the routine assumes that those that have a high precision are also those near the global point. Use of stringent tolerance limits and more runs per optimization can increase precision.

4. PERFORMANCE ANALYSIS OF THE ALGORITHM

4.1. The Single-Run Routine. A summary of the results of applying the single run routine to 15 different test functions spread over some 28 different dimensionalities is presented in Table 3. The relatively low numbers of function evaluations (f.e.) and times (shekels) coupled with the high degrees of precision in all cases shows the general applicability of the method.

Looking over the 15 different functions evaluated, it can be seen that the method can handle multiple optima, as in the cosine family ($\cos_n$, BC_$\cos_n$, and Ras_n); very sharp optima, as in the shekel family; shallow optima, as with the TC function; varying influences of one or many variables, as tested by the modified shekel function SHEKEL_5; and high dimensionality, as found in the $\cos_{20}$ and $\cos_{100}$ cases. Its notable ability to handle highly intractable functions has been amply demonstrated by its success in solving the shekel and Hartman functions, whereas its ability to solve $\cos_{20}$ and $\cos_{100}$ functions speak eloquently of its potential to solve high dimensionality problems.

Looking at the number of function evaluations required for both the $\cos_n$ and BC_$\cos_n$ families reveals that the increase in f.e. with dimensionality (N) is not quite linear. If it were, f.e. would be approximately of the order of N. However, it is consistently below that: the proportion of $\cos_{n+m}/\cos_n$ for (n,m) = (1,1), (1,19), (1,99), (2,2), (4,16), and (20,80) are all well below the ratio of $(n + m)/n$, as is the case for BC_$\cos_n$.

The computer processing time (CPU) followed very commendable trends as well. If the time required to calculate a high-dimensional function like $\cos_{100}$ is taken as an averaged benchmark for calculating the time required to find one variable inside a general calculation then the time ratio tau (τ) where

$$\tau = \text{shekel time/dimension}$$

can serve as an index of efficiency for comparison. Thus, for $\cos_{100}$, τ is 6.14, and for $\cos_{20}$, τ is 1.42. In general, the τ ratio either rises slowly with increased dimensionality or remains relatively flat as opposed to moving sharply higher. This is a highly desirable trend for an algorithm to display and is in exactly the opposite direction of most others. For example, a comparison across dimensions for the upper bound algorithm of Brieman and Cutler[7] reveals that the increase in both CPU and number of vertices is roughly exponential.

From the standpoint of both time and numbers of function evaluations, it appears that the GMC algorithm has overcome the curse of dimensionality, i.e., the exponential growth of trial points with dimension.

The starting coordinates in Table 3 have been arbitrarily chosen so that the algorithm is severely tested over considerable ranges on the response surface. In addition, each function was evaluated from various starting points and, for all except the Goldstein−Price[13] function, the performance is unaffected. For example, for symmetric functions like the cosine family, starting from (10,10,...10) or (−10,−10,...−10) required both the same amount of time (2.01 for $\cos_4$ in both cases) and virtually the same number of function evaluations (877 as opposed to 875 for $\cos_4$). For highly asymmetric functions such as the shekel function S_5, starting all variables at 0 or 5 or 10 had little influence on f.e. (667, 661, and 655, respectively) or time (2.11, 2.08, and 2.07, respectively). We also looked at starting all coordinates for the cosine family at 100 rather than 10 and, surprisingly, the times and f.e. were not much different (894 f.e. for $\cos_4$ cf. 877). The difference is hardly worth noting. This observation adds weight to the expectation that the GMC algorithm will be successful when handling real applications.

The only function that showed a significant sensitivity to the starting coordinates was the Goldstein and Price[13] test. This function was considered difficult by Brieman and Cutler.[7] The only way the algorithm could be made insensitive to starting coordinates in this instance was by

Table 3. Average Performance of the Single-Run Routine

$f(x)$[a]	accuracy (S.F.)	function evaluations	time shekels[b]	precision[c] A	precision[c] B	efficiency, sps[d]
S_5	4	661	2.08	936	936	318
S_7	4	776	1.66	934	934	467
S_{10}	4	778	2.12	935	935	367
BR	3	1160	1.25	—[e]	924	928
H_3	4	1129	2.69	1000	999	420
H_6	4	2111	6.04	1000	1000	350
GP	4	830	5.08	996	992	163
$\cos_1$	6	452	0.39	1000	1000	1159
$\cos_2$	6	597	0.81	1000	1000	737
$\cos_3$	6	743	1.36	1000	1000	546
$\cos_4$	5	877	2.01	1000	1000	436
$\cos_{20}$	3	2910	28.33	1000	996	103
$\cos_{100}$	3	12894	614.27	1000	983[f]	21
BC_$\cos_n$:						
$n = 1$	6	377	0.28	1000	1000	1346
$n = 2$	6	492	0.55	1000	1000	895
$n = 3$	5	576	0.97	1000	1000	594
$n = 4$	5	681	1.27	1000	1000	536
$n = 5$	5	781	1.74	1000	1000	449
$n = 6$	5	880	2.28	1000	998	386
B	5	605	0.67	995	995	903
C	4	618	0.51	986	971	1212
E	4	1075	1.08	1000	1000	995
Ras_2	5	559	0.60	955	923	932
HUMP	3	1043	0.80	1000	1000	1304
$T1$	3	369	0.30	983	974	1230
$T2$	4	603	0.46	1000	1000	1311
Exp_4	5	536	0.47	1000	1000	1140
TC	3	272	0.14	1000	1000	1943

[a] Starting coordinates for each function: $S_5, S_7, S_{10} = (5,..\ 5)$; BR = (0,0); $H_3, H_6 = (0,..0)$; GP = (1.0, 0); $\cos_n = (10,..10)$; BC_$\cos_n = (10,..10)$; $\cos X_1X_2 = (10,10)$; $\cos X_1 + X_2 = (10,10)$; SHEKEL$_n = (5,..5)$; $\mathrm{Ras}_2 = (10,10)$; HUMP = (10,10); T1 = (10,10); T2 = (10,10); $\exp_4 = (0.5,..0.5)$; TC = (0,0). [b] Shekel time is calculated by dividing the actual time spent on the calculation by the time required to evaluate 1000 S_5 functions at (4,4,4,4). [c] Column A has the average correct number of variables out of 1000 variables evaluated, while column B contains the number of times all variables converged to the global optima. A variable was considered to be an outlier if 0.2 units or more away from its correct position. [d] Efficiency is measured in f.e./shekel. [e] Because the Branin function has three global minima, outlier detection was calculated with reference to any of the three it found. [f] Even if one out of the 100 variables was 0.2 units away, the whole run was considered an outlier. Only 17 such instances were encountered in 1000 runs.

changing the sampling strategy to make the search for each variable completely random during the simultaneous search (*istep* = 0).

Column 4 of Table 3 shows that of the 27 tests, 14 converged all the time to the global optima, five had over 99% convergence, and the rest (eight) had over 92% convergence. Clearly, the chance of missing the global optimum on a random run is very low, and a fair degree of reliability can be placed in the algorithm.

GMC found Ras_2 to be the most difficult to optimize. On the interval [1,−1], there are ~50 quite steep local optima. The best way to overcome them is by allowing the search to iterate for some time, and it was for this reason that the multiple-run option was built into the algorithm. It assumed that in three independent runs, where each run has a 0.9 probability of success, that the probability of seeing all three runs fall outside the convergence criterion will be $(0.1)^3$ or ~1 in 1000. Thus, by doing three runs and taking the lowest, the chance of missing the global minimum is small. With increasing numbers of runs, the chances of locating the global point increases to very nearly a dead certainty. As shown in Table 4, when multiple runs are performed on functions that had <100% precision, considerable improvement can occur. For the shekel family, $\cos X_1 + X_2$ and Branin, three runs were sufficient to obtain convergence in every case, whereas for the more difficult Ras_2, $T1$, and $\cos X_1X_2$, three runs increased the proportion to >99%. When the number of runs was increased, all functions converged using the same algorithm.

Table 4. Multi-Run Routine[a] for Intractable[b] Functions

$f(x)$	accuracy	f.e.	time	precision[c]	efficiency
S_5	4	1937	6.10	1000	317
S_7	4	2118	4.53	1000	467
S_{10}	4	2119	5.79	1000	367
BR	6	2237	2.41	1000	926
$\cos X_1X_2$	6	1817	2.02	997	900
$\cos X_1 + X_2$	6	2017	1.64	1000	1226
Ras_2	6	1671	1.78	995	940
T1	3	1047	0.86	996	1217

[a] The number of *runs* selected was 3. [b] Functions selected had a precision of <1000 in Table 3. [c] Number of operations with all variables correct out of 1000 independent optimizations.

When solving a problem for the first time, the multiple-run routine can be quite useful as it ensures convergence while at the same time isolating the most troublesome function variables. Later on, as confidence with the optimization problem grows, this dependency may be reduced or eliminated, lowering computer costs by about two-thirds because the three-run option is approximately three times slower than the one-run option. This is not particularly surprising since the search is independent of starting coordinates. As such, even though each run starts from near

Table 5. Comparison with Other Methods

method	test[a]	S_5	S_7	S_{10}	H_3	H_6	BR	GP
Price[19]	f.e.	3800	4900	4400	2400	7600	1800	2500
	time	14	20	20	8	46	4	3
Fagio[20]	f.e.	2514	2519	2518	2060	10970	6012	1076
	time	7	9	13	18.5[b]	381[b]	19.7[b]	4.82
	p	0.9	0.9	0.9	0.9	0.9	0.9	0.9
de Baise[23]	f.e.	620	788	1160	732	807	597	378
	time	23	20	30	16	21	14	15
Torn[22]	f.e.	3679	3606	3874	2584	3447	1558	2499
	time	10	13	15	8	16	4	4
Branin[23]	f.e.	5500	5020	4860	—	—	—	—
	time	9	8.5	9.5	—	—	—	—
Torn[15]	f.e.	950	1017	2224	363	627	164	165
	time	122	160	170	99	161	27.5	25.5
Timmer[3]	f.e.	404	432	564	197	487	206	148
	time	1	1	2	0.5	2	0.25	0.15
	p[c]	1.00	0.75	1.00	1.00	1.00	1.00	1.00
Brooks[16]	f.e.	696	557	520	455	522	236	292
	time	2.1	1.7	1.9	1.2	2.0	0.4	0.6
	p	0.58	0.47	0.47	0.78	1.0	1.0	1.0
Breiman[7]	f.e.	—[d]	—[d]	—[d]	2575	—[d]	269	—[e]
	time				62.9		1.5	
GMC	f.e.	661	776	778	1148	2111	1160	830
	time	2.08	1.66	2.12	2.73	6.04	1.25	5.08
	p[f]	0.94	0.93	0.94	$\sim$1	1.00	0.92	0.99

[a] f.e. counts the number of times the function has to be evaluated; the time is given in shekels; p gives the probability of convergence. [b] Calculated for 0.9 probability using the shekel time given. [c] Only four independent optimizations were performed in each case. [d] Problem too large to handle [e] Did not converge in 10 000 iterations. [f] Results reported for 1000 independent optimizations.

where the previous one ended, it still takes about the same number of function evaluations. Having such independence gives an additional chance of success (law of independent probability) and allows for increased reliability.

4.2. Comparison with Other Methods. Using f.e., time, and precision as efficiency measures, GMC compares favorably with other methods, as shown in Table 5. For some functions, such as those found within the shekel grouping, it is thought to be the best algorithm. Timmer's clustering algorithm requires less f.e. and time, but the reliability of the results are uncertain. Only four runs were performed on each function, leaving doubt as to the reproducibility of the results. Indeed, S_7 does not converge to the global optimum point in one of these four runs, lending credence to such doubts. GMC, on the other hand, detects the global optima over 930 times in any 1000 independent searches for all shekel functions. Of the algorithms reporting reliability, GMC outperforms both Brooks et al.[16] and Fagio et al.[20] The de Baise algorithm requires >20 shekels of time to solve these functions, whereas the P algorithm of Törn requires >100 shekels even though they are comparable to GMC in terms of f.e. These obviously have high overhead times, which indicate extensive auxiliary calculations and, more importantly, that they will not scale well to higher dimensionality. Also, GMC is simpler to implement. It does not have any storage memory requirements in the single-run phase. All other algorithms are surpassed by GMC in both f.e. and time. For Hartman, Branin, and GP functions, GMC outperforms all of the other algorithms in either f.e. or time or both except for the generalized simulated annealing (GSA) algorithm of Brooks and Verdini.

In the interests of brevity, several other comparisons have not been included because in the vast majority of cases the GMC algorithm is considerably better either in terms of f.e. or time or both. For the record, the only two algorithmic methods which we could find that would outperform GMC on specific functions, other than those already shown in Table 5 were Bohachevsky's[10] variable-step-size generalized simulated annealing (VSGSA) operating on cos X_1X_2 and cos $X_1 + X_2$, where they reported 400 f.e. for each (no times were given), whereas we took 598 and 614 f.e. with times of 0.66 and 0.50 shekels to achieve the same degree of accuracy. The other case of note was the Brieman and Cutler algorithm[7] operating on the camel hump function. They report 336 f.e. and 0.58 shekels of time, whereas we took 1043 and 0.8 shekels of time. However, as noted previously, their algorithm is of limited applicability for many test functions and fails for moderate to high dimensionality and, as such, it is difficult to pass judgment on GMC's failures at very low dimensions.

5. CONCLUSIONS

Using only one routine with no parametric changes, the guided Monte Carlo algorithm has successfully solved 15 different types of functions considered by the literature to be difficult, with a probability of success >0.99 for most and 0.92 for the rest.

The total number of function evaluations increases with dimensionality at a rate that is less than linear for the test cases considered. The time required per optimal parameter position found either decreases with increasing dimensionality or remains relatively flat. It is possible that GMC has solved the curse of dimensionality.

ACKNOWLEDGMENT

The authors thank the Research and Publication Committee of the University of Papua New Guinea for continued funding both in terms of provision of computer facilities and software. Also, we thank the Chemistry Departments of the University of Saskatchewan and Warwick for their support while this work was being developed. R.D. expressly thanks her family for enduring encouragement.

REFERENCES AND NOTES

(1) Gill, P. E.; Murray, W.; Wright, M. H. *Practical Optimization*, Academic: London, 1981; p 1.

(2) Deming, S. N. Multiple-Criteria Optimization. *J. Chromatogr*. **1991**, *550*, 15−25.

(3) Rinnoy Kan, A. H. G.; Timmer, G. T. Stochastic Global Optimization Methods. Part I & Part II. *Math. Program* **1987**, *39*, 27−56.

(4) Archetti, F.; Frontini, F. In *Towards Global Optimization 2*; Dixon, L. C. W., Szegö, G. P., Eds.; North-Holland: Amsterdam, 1978; p 179.

(5) Hibbert, D. B. Genetic Algorithms in Chemistry. *Chemom. Intell. Lab. Sys*. **1993**, *19*, 277−293.

(6) Press: W. H.; Teukolosky, S. A.; Vetterling, W. T.; Flannery, B. P. *Numerical Recipes in FORTRAN (The Art of Scientific Computing)*, Second Edition; Cambridge University: Cambridge, 1992; p 388.

(7) Breiman, L.; Cutler, A. A Deterministic Algorithm for Global Optimization. *Math. Program* **1993**, *58*, 179−199.

(8) Cetin, B. C.; Barhen, J.; Burdick, J. W. Terminal Repeller Unconstrained Subenergy Tunneling (TRUST) for Fast Global Optimization. *J. Optim. Theor. Appl*. **1993**, *77(1)*, 97−126.

(9) Kirkpatrick, S.; Gelatt, C. D., Jr.; Vecchi, M. P. Optimization by Simulated Annealing. *Science* **1983**, *220(4598)*, 671−680.

(10) Bohachevsky, I. O.; Johnson, M. E.; Stein, M. L. Generalized Simulated Annealing for Function Optimization. *Technometrics*. **1986**, *28(3)*, 209−217.

(11) Sutter, J. M.; Kalivas, J. H. Convergence of Generalized Simulated Annealing with Variable Step Size with Application toward Parameter Estimations of Linear and Nonlinear Models. *Anal. Chem*. **1991**, *63*, 2383−2386.

(12) Stiles, G. S. The Effect of Numerical Precision Upon Simulated Annealing. *Phys. Lett. A* **1994**, *185*, 253−261.

(13) Dixon, L. C. W.; Szegö, G. P. In *Towards Global Optimization 2*; Dixon, L. C. W., Szegö, G. P., Eds.; North-Holland: Amsterdam, 1978; p 1.

(14) Pulfer, J. D.; Waine, C. An Efficient Monte Carlo Approach to Optimization. *J. Chem. Inf. Comput. Sci.* **1998**, *38*, 791−797.

(15) Törn, A A.; Zilinskas, A. In *Lecture Notes in Computer Science (Global Optimization)*; Goos, G., Hartmanis, J., Eds.; Springer-Verlag: Berlin, 1989; Vol. 350.

(16) Brooks, D. G.; Verdini, W. A. Computational Experience with Generalized Simulated Annealing Over Continuous Variables. *Am. J. Math. Management Sci*. **1988**, *8(3, 4)*, 425−449.

(17) Kalivas, J. H. Optimization Using Variations of Simulated Annealing. *Chemom. Intell. Lab. Sys.* **1992**, *15*, 1−12.

(18) Altshuler, E. L.; Williams, T. J.; Ratna, E. R.; Dowla, F.; Wooten, F. Method of Constrained Global Optimization. *Phys. Rev. Lett*. **1994**, *72(17)*, 2671−2674.

(19) Price, W. L. In *Towards Global Optimization 2*; Dixon, L. C. W., Szegö, G. P., Eds.; North-Holland: Amsterdam, 1978; p 71.

(20) Fagiuoli, E.; Pianca, P.; Zecchin, M. In *Towards Global Optimization 2*; Dixon, L. C. W., Szegö, G. P., Eds.; North-Holland: Amsterdam, 1978; p 103.

(21) de Baise, L.; Frontini, F. In *Towards Global Optimization 2*; Dixon, L. C. W., Szegö, G. P., Eds.; North-Holland: Amsterdam, 1978; p 85.

(22) Törn, A. A. In *Towards Global Optimization 2*; Dixon, L. C. W., Szegö, G. P., Eds.; North-Holland: Amsterdam, 1978; p 49.

(23) Gomulka, J. Two Implementations of Branin's Method. In *Towards Global Optimization 2*; Dixon, L. C. W., Szegö, G. P., Eds.; North-Holland: Amsterdam, 1978; p 151.

(24) Szu, H.; Hartley, R. Fast Simulated Annealing. *Phys. Lett. A*. **1987**, *122(3, 4)*, 157−162.

(25) Wilde, D. J.; Beightler, C. S. *Foundations of Optimization*; Prentice Hall: Englewood Cliffs, NJ, 1967.

CI9701042