Journal of Global Optimization **5:** 267–276, 1994.

Topographical Global Optimization Using Pre-Sampled Points

AIMO TÖRN and SAMI VIITANEN
Åbo Akademi University, Computer Science Department, FIN-20520 Åbo, Finland

(Received: 2 April 1993; accepted:22 September 1993)

Abstract. A method for global minimization of a function $f(x)$, $x \in A \subset R^n$ by using presampled global points in A is presented. The global points are obtained by uniform sampling, discarding points too near an already accepted point to obtain a very uniform covering. The accepted points and their nearest-neighbours matrix are stored on a file. When optimzing a given function these pre-sampled points and the matrix are read from file. Then the function value of each point is computed and its k nearest neighbours that have larger function values are marked. The points for which all its neighbours are marked are extracted as promising starting points for local minimizations. Results from a parallel implementation are presented. The working of a sequential version in Fortran is illustrated.

Key words: Global optimization, topography graph, parallel algorithms.

1. Introduction

The global minimum and minimizer of a function $f(x)$ in $A \subset R^n$ is to be determined. It is assumed that the problem is essentially unconstrained, i.e., that the global minimum of f is attained in the interior of A and has a basin with positive measure.

The method to be presented is a probabilistic method and may be classified as belonging to the so-called clustering methods. In these (1) basins are determined by sampling points at random, (2) points are concentrated to possible basins, (3) these concentrations are identified (the concentrated points are clustered), and (4) their corresponding local minima are determined by using some local optimization algorithm. The techniques for concentrating are: either leaving out unpromising points [1,3] or using a few steps of local optimization to bring the points nearer to the local minima "attracting" them [3]. A review of clustering methods can be found in [6].

The way to identify possible basins used here is based on using topographical information about the objective function represented as a directed graph [5]. The graph is connecting neighbouring points to each other by directed arcs pointing towards points of larger function values. If such a graph is covering enough it would be sufficient to start a local optimizer from each local minimum in the graph, i.e., from a node with no incoming arc.

The basic topographical global optimatization method consists of the following conceptual steps:

Uniform random sampling of points in the region of interest, A, is used to explore the function in each subregion of A as well as possible. Ideally the sampling should be such that at least one of the sampled points lies in the basin of the global minimum.

The second conceptual step is to determine a subset S of the accepted sampled points such that the points in this set represent the basins of all local minima "detected" in the sampling step. This is done by constructing the so called *topograph* which contains topographical information of the objective function represented by the accepted points. The topograph is a directed graph with directed arcs connecting the accepted sampled points on a k-nearest-neighbours basis, where the direction of an arc is towards a point with a larger function value. The minima in the graph, i.e., those points that have no better neighbours are part of this topographical information. We let these minima be the set S. This is motivated by the observation that for an increasing number of accepted points, these points would better and better approximate all local minima of any reasonably smooth objective function. The set S may be approximated by the set of points whose all neighbours have larger function values [5].

The third conceptual step is to determine the local minima represented by the set S. This is done by using the points in S (or a subset thereof) as starting points for local minimizations. The best point so determined is then given as the result of the global optimization effort.

Successful experiments with a sequential and a parallel realization of topographical global optimization are reported in [5]. In the parallel version the task of determining nearest neighbours, the objective function evaluations task and the local minimizations task were all parallelized.

This paper explores the following refinement. The topograph used to find graph minima is mainly dependent on the sampled points. The function values of the points will only affect the direction of the arcs. This means that given a number of points covering the region of interest the undirected topograph could be computed once and for all and stored for later use. The points could then either be loaded from a file or be regenerated for the function evaluations needed to obtain the complete topograph with directed arcs.

2. The Topographical Algorithm

The algorithm consists of two parts, "the sampler" and "the evaluator". The sampler constructs the undirected topograph and the evaluator reads an undirected topograph, constructs the topograph and performs the optimization.

2.1. THE SAMPLER

When sampling one attempts to distribute the points as evenly as possible across the search area. The method for sampling presented here is by no means the only one, and any other method known to produce a very uniform covering could be used.

Due to the fact that the search area is not known in advance in this case, the unit hypercube was chosen as an appropriate substitute. In order to force the sampled points to cover the region better than what would be the case for uniform random sampling, a threshold value is defined. Any sampled point falling within this threshold distance to an existing point is discarded. For a given N and the right choice of a threshold value a very uniform covering of the unit hypercube by N points is obtained. The sampling itself is done with quite a simple algorithm, presented below:

1. WHILE accepted < N DO
2. Sample a new point
3. Check its distance to all previously accepted points
4. IF nearest neighbour closer than δ THEN reject point
   ELSE accepted:= accepted + 1

In step 3 the Euclidean distance between points is used. The random number generator used in step 2 is an implementation of the Combined Tausworthe Random number generator by Tezuka and L'Ecuyer [2]. It has a cycle of 2^{60} and is tested to distribute points uniformly in hypercubes with dimension up to 15. This in conjunction with discarding points that come too close to an already accepted point should guarantee a very even distribution of the points. The coordinates of the accepted points are stored in a matrix, the so called C-matrix.

Once N points have been accepted the sampling terminates and the undirected topograph is created. It consists merely of a matrix, the so called knn-matrix, containing the id:s (simply the indexes of the points in the C-matrix) of the k nearest neighbours to every point, sorted by distance. Finally the C-and knn-matrices are written on files from which they can be used by the evaluator.

2.2. CHOICE OF N

How should the sampling be terminated? When only few points are sampled the resulting graph is rather crude and the graph minima will only represent some of the local minima. For an increasing number of sampled points the graph will approximate the objective function better and better and for reasonably smooth functions it is expected that the graph minima will at some stage represent all essential local minima.

The stopping condition should thus be based on the information that all parts of the region of interest have been explored, i.e., that the sampled points well cover the region. The number of points that need to be sampled in order for the probability that at least one point will be in the basin L of the global minimum to be no less than p is dependent on the relative size of this set. The probability is also dependent on how uniformly the sampled points cover the region of interest — random sampling could require several times more points than some more uniform technique for the probability to be the same [4]. However, the size of L is normally not known and therefore this technique to determine the number of points to sample requires that an assumption about the size of L is made.

Our technique to specify a threshold distance and discard any sampled point within this distance to an existing point means that as the points cover the region better and better it will become increasingly difficult to find a point that will be accepted. Therefore a maximum number of successive discarded points could be specified. When this maximum is reached the sampling will be terminated.

All these stopping conditions determine the effort that will be applied in searching for a solution. For functions expensive to evaluate the effort will be measured in number of function evaluations. Normally the information about the optimization problem is insufficient to establish the relation of this number to the probability that a point in the basin of the global minimum is obtained. In a real application the stopping condition therefore in many cases would be given as the maximum number of affordable function evaluations (available resources). The resources available to solve a particular problem would be dependent on the savings expected in obtaining a good solution. In the presentation here we assume that the user will express his expectations by specifying N.

2.3. The Evaluator

The evaluator is the process that does the actual work. It starts with the matrices created by the sampler and attempts to find the global minimum for a given function within a given search area. This is achieved by the following algorithm:

1. Read the C-and knn-matrices
2. Scale the C-matrix to the actual search area
3. Calculate the function values for all N points
4. Identify the minima in the topograph
5. Start local minimizations from a number of minima

Steps 1 and 3 require no further comments. In step 2 the scaling should preserve the hypercube in order not to distort the metrics. Step 4 consists of transforming the knn-matrix (the undirected topograph) into a directed topograph and then determining the minima in this graph.

The transformation is achieved by conceptually assigning signs to the knn-matrix. This can be illustrated by an example: Assume that point number 22 is the fifth nearest neighbour to point number 36. If now point number 22 has a lower function value than point number 36 then the element (36,5) in the knn-matrix will become -22 (unmarked), otherwise it will become +22 (marked). The signs therefore represent the directed arcs in the graph, a positive sign representing the "arrow head" of the arc, and a negative sign representing the "start" of the arc.

Finally the directed topograph is scanned for all rows that contain only arrow heads (marked points). These rows represent points for which all k neighbours have larger function values and thus are the minima in the topograph and may be used as starting points for local minimizations. In order for this procedure to be exactly correct it should additionally be checked that the point is not marked anywhere in the topograph. However, in [5] it is argumented that this simpler procedure with a slightly larger k does the same job.

The number of minima is of course heavily dependent on the value of k. For a small k the number of potential minima is large, however for $k = N$ the number will be 1 (the point with the lowest function value of all). It is typical in applications that the number of minima stabilizes and remains the same for a range of k-values.

The simplicity of the algorithm does have a few drawbacks. The assumption that a lower function value than the surrounding imply a local minimum is a very bold one. It holds only if the function is smooth within the surroundings inspected. Therefore the algorithm does not perform too well if the problem function is very ragged within small areas unless the raggedness means small perturbations on a smooth function. Also, the larger the search area the further away are the nearest neighbours. This means that for very large search areas the volume of the surroundings inspected becomes very large. This exaggerates the effect of even small raggedness and renders the algorithm useless for problems of this nature unless the number of points, N, is chosen very large. However, not only this algorithm will have difficulties in finding the global solution for such problems, most algorithms will.

2.4. The Parallel Transputer Implementation

The algorithm is implemented on the HATHI-2, a multi-transputer system located at Åbo Akademi University. The two parts of the algorithm are written as separate programs. The implementation exists in three different versions utilizing 1+8, 1+16 or 1+32 processors.

The sampler is completely sequential because it is not considered to be time critical. Some parallelism could be achieved by performing distance calculations in parallel. This can be done by distributing the already accepted points evenly among the available processors and then pipe-lining trial points through the processors. In this way every processor would compare a trial point only with a subset of all accepted points thus speeding up the comparisons, see [5]. But this would only be

TABLE I. Threshold distances for some values of N and n

N^n	2	3	4	5	6	7	8	9	10
100	0.088	0.216	0.361	0.495	0.623	0.745	0.864	0.978	1.088
200	0.060	0.166	0.285	0.410	0.522	0.630	0.738	0.840	0.940

useful in the case where N is large. Most testing was done with $k = 10$ and $N = 100$ or $N = 200$.

Some attempts were made to find a mathematical formula that would give an optimal threshold distance given N and n. By optimal we mean the maximal threshold distance for which N points could be obtained but $N + 1$ could not within feasible time. The attempts did not come out too well. Therefore a more basic "trial-and-error" method was used. Table I shows the values finally used. The number of points to be sampled in order to obtain a uniform cover of the hypercube with N points, $N = 100, 200$ for $n \in [1, 10]$ were in the range [19000,700 000]. The uniform cover means that in most cases it would be very expensive or impossible to place the $N + 1$:st point. The time to run the Sampler measured in 1000 evaluations of the Shekel 5 function (see [6]) were in the range [80, 1800]. From this it can be concluded that re-using rather than re-sampling points that cover the hypercube uniformly is profitable from an efficiency point of view. Other ways to produce the uniform cover could be used as well. However, because the sampling is a one time procedure the possible inefficiency of the method used here is not crucial.

The evaluator consists of two parts. The main program is run on one processor, the so called *root* processor, and performs steps 1, 2 and 4 of the algorithm presented above in sequence. The second part (steps 3 and 5) is performed by P ($P = 8$, 16, or 32) so called *slave* processors. These are connected in a one-way ring starting and ending with the root. The ring topology was chosen because its simplicity and because the fast communication between transputers results in only very little overhead even for long rings. The N points are distributed evenly among the P processors so that the function evaluations can be done in parallel. This parallelization of step 3 reduces the time required for the function evaluations by a factor of P.

Step 4 is the key element in the implementation of the topographical algorithm. The transformation of the undirected topograph to a directed one and the subsequent identification of the minima in this graph are merged to a single scan of the knn-matrix. During this scan the function values of the k neighbours to the current point are compared to the function value of the current point. If the function value of any of the k neighbours is lower than the function value of the current point, then the current point is considered not to be a minimum in the topograph.

The implementation actually identifies k different sets of minima, one for each value of the number of nearest neighbours in the range $[1, k]$. The program then chooses the cluster size which is optimal with regard to P, i.e., the one that gives

TABLE II. Function independent overhead (unit: 1 Shekel 5)

n	T1	(a)	(b)	(c)	T3	T1+T3
2	431	9	35	13	57	488
4	651	19	55	13	87	738
6	870	28	75	14	117	987
10	1275	47	116	15	178	1453

the largest number of local minima still smaller than or equal to P. This is done because in step 5 a maximum of P local minimizations can be done in parallel by having every slave perform one local minimization.

Because we are mainly interested in the global part of the algorithm, local minimization (step 5) was not considered to be very important (any suitable method would do). Nevertheless, to obtain a complete picture of the behavior of the algorithm, a local minimization algorithm using gradient evaluations was written. The next section presents the results.

2.5. Experimental Results

The speed of the algorithm depends on a number of factors but can be given roughly as $T = T1 + T2 + T3 + T4$. $T1$ here is the time it takes to read the C-matrix from the file, $T2$ is the time spent performing function evaluations ($T2 = N\alpha/p$, where α is the time it takes to perform one function evaluation), $T4$ is the time spent performing local minimizations (in reality the time it takes to perform the most timeconsuming of the selected local minimizations). $T3$ is all the remaining time and includes the times for scaling (a) and distributing (b) points, plus identifying local minima (c). The times of $T1$ and $T3$ are independent of the function optimized. The values of these for the implementation using 1+32 transputers and a range of n-values are given in Table II. The times are given using the unit: 1 Shekel 5 function evaluation.

The values for $T1$ are mean values. There are fluctuations ($\pm 25\%$) due to the unpredictability of the wait times for file accesses.

$T4$ which is the time spent on doing the longest local search depends partly on chance (i.e. how much work needs to be done to get from the starting point to the true local minimum), partly on the efficiency of the local search algorithm (for example how many function evaluations it needs to perform).

Results for some standard test functions [6] can be found in Table III. The function evaluations listed include also those needed for gradient evaluations during the local minimizations. The results for the three first test functions (1.13, 1.27, 2.07) could be compared with the results 1.9, 2.3, 2.6 obtained without using pre-sampled points. However, the total number of points sampled in this latter case was only a small fraction of those used in the pre-sampled case meaning that the

TABLE III. Results by using 1 + 32 processors for standard test functions (average over 20 runs)

	Test function f			
	RCOS	Shekel 5	Hartman 6	Griewank 10
Dimensionality of f	2	4	6	10
# of local minimizations	15	21	22	27
Function evaluation	122	98	168	283
$T4$ (unit: 10^3 Shekel 5)	0.11	0.22	0.80	5.30
T (unit: 10^3 Shekel 5)	1.13	1.27	2.07	7.10

points used in the latter case were only slightly more uniformly distributed than points sampled from the uniform distribution and thus not of the same quality as the pre-sampled points.

3. A Sequential Fortran Algorithm

Below the working of a Fortran implementation of the sequential algorithm is given for the problem RCOS. The code consists of three parts: the user interface part, the sampling part, and the minimization part. No local optimization is performed in this version. The user should write the code for computing function values. The user first asks the program either to sample or to minimize. The user is requested to give the name of a file giving problem information, i.e., n and the coordinates of the hypercube.

For sampling the following parameters should be submitted by the user: the number of global points to sample (N), the threshold distance (THRESH) for the sampling phase to avoid points near to each other, whether the sampling should be performed in the unit hypercube or other area, and two seeds for the random number generator. The sampling area should be a hypercube. On sampling the coordinates of each point accepted is printed together with the cumulative number of points discarded. For an example see the output below for $N = 100$ and THRESH = 0.087:

```
ID     DISCARDED   x1     x2     ...
 1                 0.55   0.42
 2         0       0.50   0.73
...
98      5050       0.68   0.62
99      8918       0.00   0.57
        18918
100     25180      0.29   1.00
```

It can be seen that in order to sample the requested 100 points 25180 points were discarded. When the sampling is finished the user must submit the name of the file to store the points and the C-matrix.

For optimizing the names of the problem information file and the file containing the points and the C-matrix must be given. The result is then computed and stored in a file named by the user. At the end of the optimization the ID of the local minima in the graph for $k = 1,2,\ldots,18$ are given together with the minima and minimizers for all local minima for $k = 18$, see below:

```
K    #MIN   MINIMA
 1    51    94 43 52 70 16 49 66 88 24 35 59 28 27 ...
 2    19    94 43 70 16 49 35 28  1 39 54 72 13 80 ...
 3     7    94 43 70 16 49 39 13
 4     5    94 43 70 16 49
 5     4    94 43 70 16
 6     3    94 43 70
...
18     3    94 43 70

THE 3 MINIMA FOR K=18

POINT 94     F(X)= 0.13661872E+01 X:
 9.762428     3.427323

POINT 43     F(X)= 0.18809843E+01 X:
 -3.226959    11.27771

POINT 70     F(X)= 0.31365552E+01 X:
 3.009627     0.7506922
```

From the output one can see how the number of minima stabilizes when the value of k grows. The three minima given as the result are near to the global minima of RCOS. The result above is typical for the given parameter values of N and THRESH.

On completion the result is stored in the user named file. The result consists of the problem information, the global points used, their function values and the information about the minima as illustrated above.

4. Conclusions and Discussion

Topographical global optimization using pre-sampled points is a method with almost no overhead. This is because the work to find a uniform covering of the region of interest, A, and the work to determine the nearest neighbour matrix giving the undirected topograph can be performed once and for all for a given number of dimensions n and global points N. For optimizing the stored points and the matrix representing the undirected topograph are read from file, the function values of the

points are calculated giving the directed topograph from which the local minima of the topograph are easily extracted, giving starting points for local minimizations.

In the parallel version of the algorithm the function evaluations can be performed in parallel as well as the local minimizations which means linear speed up for the function evaluations. Also in many cases the local minimizations can be made at the expense of just one local minimization. The size of the algorithms is very small (for the sequential algorithm in Fortran the optimization part excluding the subroutines for function evaluations and local minimization is only about 100 lines) which makes the algorithm easily available for experimenting and possible inclusion in other software.

Another way of sampling would be to sample in a smaller hypercube centered at the latest accepted point until a prescribed number of rejections occurs. Then sampling for a point in the hypercube H containing A is resumed and so on. The sampling is terminated when a prescribed number of rejections in H is achieved. Such a sampling is expected to be more efficient and would extend the applicability of the method to problems where A is any region of positive measure and thus to a subset of constrained problems.

References

1. R.Becker and G.Lago (1970) A global optimization algorithm, in: *Proceedings of the 8th Allerton Conference on Circuits and Systems Theory,* Monticello, Illinois, 312pp.
2. S.Tezuka and P.L'Ecuyer (1991) Efficient and Portable Combined Tausworthe Random Number Generators, *ACM Transactions on Modelling and Computer Simulation* **1** (2), 99–112.
3. A.Törn (1974) Global optimization as a combination of global and local search, PhD. Thesis, Åbo Akademi, HHÅA A:13, 65pp.
4. A.Törn (1978) A search-clustering approach to global optimization, in: *Towards Global Optimization* **2**, North-Holland, 49–62.
5. A.Törn and S. Viitanen (1992) Topographical Global Optimization, In: C. A. Floudas and P. M. Pardalos (eds.), *Recent Advances in Global Optimization,* Princeton University Press, 384–398.
6. A.Törn and A.Žilinskas (1989) *Global Optimization,* Lecture Notes in Computer Science 350, Springer-Verlag Berlin, 255pp.