Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Physics Communications 159 (2004) 63–69

Computer Physics Communications

www.elsevier.com/locate/cpc

PANMIN: sequential and parallel global optimization procedures with a variety of options for the local search strategy ☆

F.V. Theos [a], I.E. Lagaris [a,*], D.G. Papageorgiou [b]

[a] *Department of Computer Science, University of Ioannina, P.O. Box 1186, Ioannina 45110, Greece*
[b] *Department of Materials Science and Engineering, University of Ioannina, P.O. Box 1186, Ioannina 45110, Greece*

Received 7 August 2003; accepted 1 November 2003

Abstract

We present two sequential and one parallel global optimization codes, that belong to the stochastic class, and an interface routine that enables the use of the Merlin/MCL environment as a non-interactive local optimizer. This interface proved extremely important, since it provides flexibility, effectiveness and robustness to the local search task that is in turn employed by the global procedures. We demonstrate the use of the parallel code to a molecular conformation problem.

Program summary

Title of program: PANMIN
Catalogue identifier: ADSU
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/ADSU
Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland
Computer for which the program is designed and others on which it has been tested: PANMIN is designed for UNIX machines. The parallel code runs on either shared memory architectures or on a distributed system. The code has been tested on a SUN Microsystems ENTERPRISE 450 with four CPUs, and on a 48-node cluster under Linux, with both the GNU g77 and the Portland group compilers. The parallel implementation is based on MPI and has been tested with LAM MPI and MPICH
Installation: University of Ioannina, Greece
Programming language used: Fortran-77
Memory required to execute with typical data: Approximately $O(n^2)$ words, where n is the number of variables
No. of bits in a word: 64
No. of processors used: 1 or many
Has the code been vectorised or parallelized?: Parallelized using MPI
No. of bytes in distributed program, including test data, etc.: 147163
No. of lines in distributed program, including the test data, etc.: 14366
Distribution format: gzipped tar file

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com.science/journal/00104655).
* Corresponding author.
E-mail address: lagaris@cs.uoi.gr (I.E. Lagaris).

doi:10.1016/j.cpc.2003.11.001

Nature of physical problem: A multitude of problems in science and engineering are often reduced to minimizing a function of many variables. There are instances that a local optimum does not correspond to the desired physical solution and hence the search for a better solution is required. Local optimization techniques can be trapped in any local minimum. Global Optimization is then the appropriate tool. For example, solving a non-linear system of equations via optimization, one may encounter many local minima that do not correspond to solutions, i.e. they are far from zero

Method of solution: PANMIN is a suite of programs for Global Optimization that take advantage of the Merlin/MCL optimization environment [1,2]. We offer implementations of two algorithms that belong to the stochastic class and use local searches either as intermediate steps or as solution refinement

Restrictions on the complexity of the problem: The only restriction is set by the available memory of the hardware configuration. The software can handle bound constrained problems. The Merlin Optimization environment must be installed. Availability of an MPI installation is necessary for executing the parallel code

Typical running time: Depending on the objective function

References: [1] D.G. Papageorgiou, I.N. Demetropoulos, I.E. Lagaris, Merlin-3.0. A multidimensional optimization environment, Comput. Phys. Commun. 109 (1998) 227–249.

[2] D.G. Papageorgiou, I.N. Demetropoulos, I.E. Lagaris, The Merlin Control Language for strategic optimization, Comput. Phys. Commun. 109 (1998) 250–275.

PACS: 02.60.Pn; 02.60.Ed; 07.05.Tp; 07.05.Mh

Keywords: Global and local optimization; Modeling; Curve-fitting; Neural network training

1. Introduction

Optimization has proved to be an invaluable tool in many scientific fields. Its strength comes in part from the fact that a plethora of diverse scientific as well as practical problems, can be reduced to optimization problems, and in part from the existence of robust and effective optimization methods. At this point we would like to make the distinction between local optimization and global optimization. A function may have more than one points where the optimality conditions are satisfied, i.e. many local minima. Any of these is an acceptable solution as far as local optimization is concerned. Among the local minima, the one with the lowest value, i.e. the global minimum, is the solution that global optimization seeks for. Note also that there may exist several global minima, i.e. minima at different positions with the same, globally lowest, function value. Various global optimization methods have been developed rather recently, many of them being of stochastic nature. These methods recover the global minimum with a probability tending to one, in the asymptotic limit. Deterministic methods that would guarantee the recovery of the global minimum, face at present various computational difficulties. We describe two methods that belong to the stochastic class and we present their implementation. The first aims to locate only one global minimum, while the second to collect all the existing local minima. Specifically we present a modification of the "Controlled Random Search" (CRS), originally introduced by Price [3], and the "Healed Topographical Multilevel Single Linkage" (HTMLSL) that is based on the Multi Level Single Linkage (MLSL) method of Kan and Timmer [6–8], with topographical modifications inspired from the articles by Ali and Storey [9] and by Törn and Viitanen [10]. The above two codes are sequential and do not take advantage of environments with multiple processors. Note also that CRS can hardly benefit from such a computational environment due to the nature of its algorithmic structure. HTMLSL lends itself to parallel processing and is reprogrammed using the Message Passing Interface (MPI) to take advantage of either a shared memory architecture or a distributed environment with interconnected computers.

For a review with detailed bibliography and comparisons among various methods, we refer to [11]. For the recent developments in the area of global optimization we refer to the article by Pardalos et al. [12]. Our implementation is based on the Merlin [1] optimization environment and its programming language MCL [2]. Merlin offers several local optimization algorithms and many useful tools. MCL is a programming language for implementing optimization strategies in the context of the Merlin environment.[1]

This article is organized as follows. In Section 2, we present the implemented algorithms and we make some relevant comments. In Section 3 we present `OPTIMA`, the routine that interfaces the Merlin package so that it can be used directly from a user's program as a local optimization routine. In Section 4 we lay out the documentation of the corresponding software.

In the user's manual, we apply the implemented methods to the standard "Six-hump Camel-Back" test function and we list the output of the test runs. In the examples section, we present a code that makes use of the subroutine OPTIMA to implement the simplistic multistart approach for global optimization, i.e. we generate a number of points at random and from each one we start a local search.

2. Algorithmic description

In the following the descriptions of the two algorithms are given. First in Section 2.1 we sketch the CRS, and in Section 2.2 the HTMLSL method. The parallel version of HTMLSL is described in Section 2.3.

2.1. Controlled random search

This is a modification of Price's [3] algorithm, similar but not identical to the one described in [4]. The method seeks for one global minimum in a given domain D. Here the feasible domain D is considered to be a rectangular hyperbox. The steps of the procedure are given (in a Fortran-like fashion) by:

Input data:

- M, an integer such that $M > N + 1$, where N is the space dimension. (Suggested value: $M = 25N$)
- ϵ, a small positive constant. (Suggested value: $\epsilon = 10^{-6}$)
- ω, a rather large positive constant. (Suggested value: $\omega = 1000$)

[1] We maintain the site http://merlin.cs.uoi.gr, where updated versions of the Merlin/MCL software and its manual can be found, as well as installation scripts, compiler specific issues, development info etc.

Step 0:

- Set $k = 0$. Form the initial set $S^k = \{x_1^k, x_2^k, \ldots, x_M^k\}$ by picking M points randomly from D
- Evaluate: $f_i^k = f(x_i^k)$ for $i = 1, 2, \ldots, M$

Step 1:

- $f_{\max}^k = \max\{f_i^k\}$, and let the corresponding point be denoted as $x_{\max}^k$. Similarly
- $f_{\min}^k = \min\{f_i^k\}$, and the corresponding point is denoted as $x_{\min}^k$
- IF $f_{\max}^k - f_{\min}^k \leqslant \epsilon$, polish $x_{\min}^k$ via a local search procedure and STOP

Step 2:

- Choose at random $N+1$ points $\{x_{i_0}^k, x_{i_1}^k, \ldots, x_{i_N}^k\}$ from S^k
- Calculate the weighted centroids:

$$c_w^k = \sum_{j=1}^{N} w_j^k x_{i_j}^k, \qquad f_w^k = \sum_{j=1}^{N} w_j^k f\left(x_{i_j}^k\right)$$

where:

$$w_j^k = \frac{n_j^k}{\sum_{j=1}^{N} n_j^k},$$

$$n_j^k = \frac{1}{f(x_{i_j}^k) - f_{\min}^k + \phi^k},$$

$$\phi^k = \omega \frac{(f_{\max}^k - f_{\min}^k)^2}{f_{\max}^0 - f_{\min}^0}$$

- Calculate a trial point $\bar{x}^k$ as:

$$\bar{x}^k = (x_{i_0}^k - c_w^k) \frac{f(x_{i_0}^k) - f_w^k}{f_{\max}^k - f_{\min}^k + \phi^k} + \Delta_w^k$$

- where $\Delta_w^k = 2c_w^k - x_{i_0}^k$ if $f_w^k \leqslant f(x_{i_0}^k)$ and $\Delta_w^k = 2x_{i_0}^k - c_w^k$ if $f_w^k > f(x_{i_0}^k)$
- IF $\bar{x}^k \notin D$ REPEAT step 2
- Compute $f(\bar{x}^k)$

Step 3:

- IF $f(\bar{x}^k) \geqslant f_{\max}^k$ THEN

- Calculate the success rate (the fraction of function evaluations that led to a new lower upper bound)
- IF success rate $> 50\%$, set $S^{k+1} = S^k$, $k = k+1$ and GOTO step 2
- Calculate $y^k = \frac{c_w^k + x_{i_N}^k}{2}$, compute $f_y = f(y^k)$
- IF $f_y \geqslant f_{\max}^k$, set $S^{k+1} = S^k$, $k = k+1$ and GOTO step 2
- Set $S^{k+1} = S^k \cup \{y^k\} - \{x_{\max}^k\}$, $k = k+1$ and GOTO step 1

• ENDIF

Step 4:

• Set $S^{k+1} = S^k \cup \{\bar{x}^k\} - \{x_{\max}^k\}$
• Increment: $k = k+1$ and GOTO step 1

2.1.1. *Comments*

The above algorithm (as quoted in [4]) has been designed for problems where the objective function is affected by the presence of noise and its gradient is not analytically available. Such problems, in the case of local optimization are treated with reasonable success by the irregular Simplex method [5]. The presented global algorithm is inspired in part by the tactic followed in that method. (Maintaining, i.e. a population of points and performing operations such as reflection with respect to a centroid, etc.). Note that if more than one global minima exist, this method will locate only one of them.

2.2. *Healed topographical multilevel single linkage*

In the present article we describe a stochastic method based on the MLSL algorithm [7], integrated with ideas from [9] and [10]. A healing technique along with a threshold on the number of iterations is used, to prevent premature termination at the early stages of the algorithm. The algorithm attempts to find all local minima of an objective function $f(x)$ inside a bounded set $S \subset R^n$, that are potentially global. These local minima are obtained by a local-search procedure, starting from suitably chosen points in a properly maintained sample. Stochastic algorithms in the framework of multistart suffer from the problem of recovering the same local minima repeatedly, a fact that diminishes their efficiency. MLSL is devised in such a way so as to avoid this undesirable repetition. Here again the bounded set $S \subset R^n$ is the rectangular hyperbox $[a_1, b_1] \otimes [a_2, b_2] \otimes \cdots \otimes [a_n, b_n]$.

At the kth iteration:

(1) Construct a sample by picking at random N points from S. At each point evaluate the objective function.
(2) Choose from the sample a subset of points to be used as start points for local searches.
(3) Perform a local search from each start point. If a new minimum is discovered store it.
(4) Determine whether to stop or not. If not, repeat from step 1.

From the stored local minima the one with the lowest value may be regarded to be the global minimum.

Steps 2 and 4 require further description. Steps 1 and 3 are straightforward, however a few comments may be helpful.

Step 1

The points are drawn from a uniform distribution. In [9] the Halton sequence is used instead, which in the asymptotic limit produces uniformly distributed points. This may reduce the number of function evaluations by a few percent. (A 15% reduction is reported in [9].) Maintaining asymptotically the uniform distribution is important, since the strong theoretical (asymptotic) results in [6] and [7], are only proved for the uniform distribution. In [10] uniform sampling is used. However if a new point is too close to another, is rejected in an attempt to cover S in a more evenly manner.

Step 2

This is the step that characterizes the method as “Topographical MLSL”. In this step we first add to the sample the already found (initially none) local minima. So the sample contains $N + w$ points, w being the number of the local minima found so far. For every point $r_i \in S$ we find its c closest neighbors b_{ij}, $j = 1, \ldots, c$. If $f(r_i) \leqslant f(b_{ij})$, $\forall j = 1, \ldots, c$, then the point r_i is called a graph minimum. The start points for the local searches are chosen from within the set of the graph minima. A point from that set is a start point as long as:

1. It is not a local minimum found earlier, and
2. There is no other point within a critical distance R, with a lower function value.

The critical distance is the one used and in the plain MLSL algorithm, depends on the iteration number k and is given by:

$$R_k = \frac{1}{\sqrt{\pi}}\left[\Gamma\left(1+\frac{n}{2}\right)m(S)\sigma\frac{\log(kN)}{kN}\right]^{1/n}, \tag{1}$$

where $m(S)$ is the Lebesgue measure of S, and σ is a user supplied positive parameter. If $\sigma > 4$ then, even if the sampling continues for ever, the total number of local searches has been proved to be finite with probability one. See [7] and [8].

Step 3

The local search is of key importance. It is invoked many times during the course toward the global minimum, and hence its efficiency and robustness influence the overall performance dramatically. Methods that require the Hessian matrix or the Gradient vector, cannot treat satisfactorily functions that are not continuously differentiable. On the other hand, methods that use only function values are comparatively inefficient when applied to smooth, continuously differentiable functions. Unfortunately there is no single method applicable to all cases as a panacea. Therefore the local procedure has to be either an intelligent system capable of making decisions as to which method is suitable to apply, or easily replaceable without the need to tamper with the source code. This is precisely the reason we have chosen the Merlin/MCL system in our implementation. Merlin offers many optimization algorithms. The information instructing which one to use, is read from a text file. This file can be easily edited each time accordingly and there is absolutely no need to change the source code. Alternatively, an intelligent minimization strategy can be coded in MCL, and Merlin can be instructed to execute it every time a local search is invoked. Such an example is provided in the user's manual. By using Merlin/MCL we satisfy the so crucial requirements of flexibility, efficiency and robustness.

Step 4

This is the final step, where a decision is made, based upon a Bayesian criterion, concerning the continuation or the termination of the iterations. A Bayesian estimate of the total number of local minimizers [6] is given by:

$$w_{\text{est}} = \frac{w(t-1)}{t-w-2}. \tag{2}$$

Similarly an estimate for the covered portion of S by the regions of attraction of the local minimizers already found is expressed as:

$$p_{\text{cov}} = 1 - \frac{w(w+1)}{t(t-1)} \tag{3}$$

where t in [6] stands for the total number of points used in the sample after a cut-off level reduction. Here the reduction is implicitly performed by selecting the start points from the set of the graph minima. Hence at iteration k we take:

$$t = \sum_{i=1}^{k}\big[L_i + \alpha_i(G_i - L_i)\big], \tag{4}$$

L_i, G_i stand for the number of start points and the number of graph minima at the ith iteration, and $\alpha_i \in (0, 1)$ is given by:

$$\alpha_i = \frac{1-\exp(-i/h)}{1+\exp(-i/h)}, \tag{5}$$

h being a positive healing parameter. Note that as the iteration count increases $\alpha_i \to 1$ (healing), and then the summand in Eq. (4) becomes equal to the number of graph minima. Two conditions must be satisfied in order to terminate the algorithm.

(1) $w_{\text{est}} < w + \frac{1}{2}$, or equivalently, $2w^2 + 3w + 2 < t$, and
(2) $p_{\text{cov}} > 1 - \epsilon$, or equivalently, $w(w+1) < \epsilon(t-1)t$,

ϵ being a small positive number. As it can be readily realized, healing, protects the algorithm from premature termination, by delaying the growth of the t-values for a number of initial iterations. As an additional control parameter, a threshold I_t on the minimum number of iterations is used. This forces the algorithm to iterate for at least I_t times.

2.3. Parallel HTMLSL

For heavy tasks parallelization is of great importance. For example the determination of the conformational structure of molecules can be an exceedingly time consuming application. The parts of the algorithm that are parallelized are the nearest neighbor procedures and the local searches.

We employ the "one master $+ n$ workers" scheme. Therefore one needs at least two processors to execute this version of the program. The workers perform two actions:
a) Determination of the nearest neighbors of a sample point
b) Local minimization of the objective function.

The master processor creates the random points inside the hyperbox, evaluates the objective function at every point and communicates this information to all workers. Each sample point is assigned to a worker for nearest neighbor determination. After this the master selects the start points. Each start point is then assigned to a worker for local optimization. In both of the above actions, load balancing is achieved via a round-robin tactic among the workers. Finally the master accepts only the truly new minima rejecting the rest. The accepted minimizers are passed to all workers.

The implementation is based on MPI which is a widely accepted standard for parallel and distributed computing and is available for most computers and operating systems.

3. The `OPTIMA` interface

The Merlin/MCL-3.0 package, offers a powerful environment for optimization. Merlin expects suitable instructions from an available repertoire, to search for the minimum of a user-prepared objective function. This procedure has an interactive character, which is fine when the central issue is to obtain the optimum of a function. However when the minimization is only an intermediate task, required in the course of a calculation, a single call to a routine that returns a minimum point is more appropriate and far more convenient. To meet this need, we devised an interface, that permits one to call from within his own program the Merlin optimization environment and use it without having to instruct it interactively. One may wonder if such an interface is worthwhile and ask if it is any different from the so many library optimization routines that operate in a similar way. The answer to this will become apparent after the description is read, but let us say a few words in advance. Merlin [1] supports many optimization algorithms, not just one. Some of them use derivatives and are suitable for smooth functions, some others use only function values and are appropriate for noisy functions with, maybe, discontinuous derivatives etc. Merlin can also handle combinations of different algorithms, i.e. optimization strategies, which have proved to be important. On the other hand, library routines usually implement a single algorithm and hence each time a different requirement is to be met, one would have to change the call to another optimization routine, that implements the proper algorithm, and so on so forth. Tampering with the source code too often, is both unpleasant and error prone. By assigning the optimization task to Merlin, one has the immediate benefit that the customization (i.e. which method to use) is performed without changing the source code, but only an entry to an input text file. In addition, using MCL [2] directives, one can employ an intelligent strategy instead of a single method. It is understood that the objective function must be written in the form that is required by the Merlin package.

4. Documentation of the software

There are three Global Optimization modules and the `OPTIMA` Interface subroutine, as listed below.

- Program **PRICE**.
  The implementation of the Price algorithm for Global Optimization, as presented in Section 2.1.
- Program **TML**.
  The implementation of the HTMLSL, as presented in Section 2.2.
- Program **PTML**.
  The parallel implementation of HTMLSL, as presented in Section 2.3.
- Subroutine **OPTIMA**.
  The user-interface to the *Merlin* [1,2] optimization package.

Detailed instructions concerning installation and usage for the PRICE, TML and PTML modules, are included in the accompanying manual. The documentation of the `Optima` subroutine follows.

4.1. The OPTIMA interface

The interface is implemented as a `FORTRAN-77` subroutine. Its syntax and argument description follows.

```
 SUBROUTINE OPTIMA ( N, M, XP, XV, XLL,
                     XRL, IXAT, ICODE,
&                    FINP, FOUT, GRMS,
                     NF, NG, NH, NJ )
```

4.1.1. Argument description

- `N` is the space dimension (i.e. the number of parameters).
- `M` is the number of terms in a Sum of Squares objective function. For a general function `M` must be set to zero. Namely when `FUNCTION FUNMIN` is prepared then `M` is set to zero. Alternatively if `SUBROUTINE SUBSUM` is prepared for a sum of squares case, `M` is set to be the number of terms considered.
- `XP` is an array that on input it may contain a starting point, and upon return contains the minimizer.
- `XV` upon return contains the value of the minimum.
- `XLL, XRL, IXAT` are arrays that on input may contain the lower bounds, the upper bounds and the fix-status of the parameters.
- `ICODE` is an input integer array with four elements. Each element may take the values 0 or 1.
  1. If `ICODE(1) = 1` then the (input) contents of XP will be used to initialize the parameters.
  2. If `ICODE(2) = 1` then the contents of XLL will be used to set the lower bounds for the parameters.
  3. If `ICODE(3) = 1` then the contents of XRL will be used to set the upper bounds for the parameters.
  4. If `ICODE(4) = 1` then the contents of IXAT will be used to set the fix-status of the parameters.

  If any of the `ICODE` elements is zero, the corresponding action is not taken.
- `FINP` is an input character string, containing the name of a file that contains Merlin instructions or alternatively the object code of an MCL program.
- `FOUT` is an input character string, containing the name of a file where Merlin's output will be disposed. This file has no use (other than debugging) and in Unix systems can be set to equal `/dev/null` to suppress it. The setting `FOUT = ' '`, corresponds to the standard output device.
- `GRMS` upon return contains the value of the root mean square gradient, at the returned point `XP`.
- `NF` upon return contains the number of the performed function evaluations.
- `NG` upon return contains the number of the performed gradient evaluations, i.e. the calls to the (optionally) user-supplied, `SUBROUTINE GRANAL`.
- `NH` upon return contains the number of the performed calls to the (optionally) user-supplied `SUBROUTINE HANAL`.
- `NJ` upon return contains the number of the performed calls to the (optionally) user-supplied `SUBROUTINE JANAL`.

References

[1] D.G. Papageorgiou, I.N. Demetropoulos, I.E. Lagaris, Merlin-3.0. A multidimensional optimization environment, Comput. Phys. Commun. 109 (1998) 227–249.

[2] D.G. Papageorgiou, I.N. Demetropoulos, I.E. Lagaris, The Merlin Control Language for strategic optimization, Comput. Phys. Commun. 109 (1998) 250–275.

[3] W.L. Price, A controlled random search procedure for global optimization, in: L.C.W. Dixon, G.P. Szego (Eds.), Toward Global Optimization 2, North-Holland, Amsterdam.

[4] P. Brachetti, M. De Felice Ciccoli, G. Di Pillo, S. Lucidi, A new version of the Price's algorithm for global optimization, J. Global Optim. 10 (1997) 165–184.

[5] J.A. Nelder, R. Mead, A Simplex method for function minimization, Comput. J. 7 (1965) 308–313.

[6] A.H.G. Rinnooy Kan, G.T. Timmer, Stochastic global optimization methods; Part I: Clustering methods, Math. Programming 39 (1987) 27–56.

[7] A.H.G. Rinnooy Kan, G.T. Timmer, Stochastic global optimization methods; Part II: Multilevel methods, Math. Programming 39 (1987) 57–78.

[8] G.T. Timmer, Global optimization; A stochastic approach, PhD Thesis, Erasmus University, Rotterdam.

[9] M.M. Ali, C. Storey, Topographical multilevel single linkage, J. Global Optim. 5 (1994) 349–358.

[10] A. Törn, S. Viitanen, Topographical global optimization using presampled points, J. Global Optim. 5 (1994) 267–276.

[11] L. Özdamar, M. Demirhan, Experiments with new stochastic global optimization search techniques, Comput. Oper. Res. 27 (2000) 841–865.

[12] P.M. Pardalos, H.E. Romeijn, H. Tuy, Recent developments and trends in global optimization, J. Comput. Appl. Math. 124 (2000) 209–228.