Available online at www.sciencedirect.com
ScienceDirect
Computer Physics Communications ••• (••••) •••–•••
Computer Physics Communications
www.elsevier.com/locate/cpc

GenMin: An enhanced genetic algorithm for global optimization ☆

Ioannis G. Tsoulos *, I.E. Lagaris

Department of Computer Science, University of Ioannina, P.O. Box 1186, Ioannina 45110, Greece

Received 10 October 2007; received in revised form 2 January 2008; accepted 12 January 2008

Abstract

A new method that employs grammatical evolution and a stopping rule for finding the global minimum of a continuous multidimensional, multimodal function is considered. The genetic algorithm used is a hybrid genetic algorithm in conjunction with a local search procedure. We list results from numerical experiments with a series of test functions and we compare with other established global optimization methods. The accompanying software accepts objective functions coded either in Fortran 77 or in C++.

Program summary

Program title: GenMin
Catalogue identifier: AEAR_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEAR_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html
No. of lines in distributed program, including test data, etc.: 35 810
No. of bytes in distributed program, including test data, etc.: 436 613
Distribution format: tar.gz
Programming language: GNU-C++, GNU-C, GNU Fortran 77
Computer: The tool is designed to be portable in all systems running the GNU C++ compiler
Operating system: The tool is designed to be portable in all systems running the GNU C++ compiler
RAM: 200 KB
Word size: 32 bits
Classification: 4.9
Nature of 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 are frequently trapped in local minima. Global optimization is hence the appropriate tool. For example, solving a nonlinear system of equations via optimization, employing a least squares type of objective, one may encounter many local minima that do not correspond to solutions (i.e. they are far from zero).
Solution method: Grammatical evolution and a stopping rule.
Running time: Depending on the objective function. The test example given takes only a few seconds to run.

PACS: 02.60.-x; 02.60.Pn; 07.05.Kf; 02.70.Lq; 07.05.Mh

Keywords: Genetic algorithm; Genetic programming; Grammatical evolution; Global optimization; Stochastic methods; Stopping rule

☆ 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: itsoulos@cs.uoi.gr (I.G. Tsoulos).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2008.01.040

Please cite this article in press as: I.G. Tsoulos, I.E. Lagaris, GenMin: An enhanced genetic algorithm for global optimization, Computer Physics Communications (2008), doi:10.1016/j.cpc.2008.01.040

1. Introduction

The problem of locating the global minimum of a continuous and differentiable function f inside a bounded domain defined as $S \subset R^n$ is considered in this paper. The problem can be formulated as: Determine

$$x^* = \arg\min_{x \in S} f(x)$$

The nonempty set $S \subset R^n$ considered here, is a hyper box defined as:

$$S = [l_1, u_1] \otimes [l_2, u_2] \otimes \cdots [l_n, u_n].$$

The problem of locating the global minimum of a function finds many applications in a variety of scientific fields such as: physics, astronomy, chemistry, etc. Recently several methods have been proposed for the solution of the global optimization problem. These methods can be divided in two main categories, deterministic and stochastic. The methods which belong to the first category are more difficult to implement and they depend on a priori information about the objective function. Some examples of stochastic methods suggested for the global optimization problem are: Random Line Search, Adaptive Random Search [1], Competitive Evolution [2], Controlled Random Search [3], Simulated Annealing [4–6], Genetic Algorithms [7,8], Differential Evolution [9,10], methods based on Tabu Search [11], etc. In this article a genetic algorithm is introduced for the location of the global minimum. The method utilizes the Grammatical Evolution [12] procedure for the creation and evolution of the population and a new stopping rule for the genetic algorithm, that is based on asymptotic considerations. The reason behind the use of Grammatical Evolution in encoding real numbers is that the genetic operations in Grammatical Evolution are faster than of real-code genetic algorithm. The proposed method is useful for determining the most stable conformations of a molecule. We give examples of Lennard–Jones clusters associated with a molecule of 3 atoms (potential3) and a molecule consisting of 5 atoms (potential5). In these cases the potential energy of the molecule is minimized with respect to the atomic positions. The global minimum of the energy corresponds to the desired conformation.

The rest of this article is organized as follows: in Section 2 the Grammatical Evolution procedure is outlined and the proposed algorithm is explained in detail, in Section 3 the proposed method is applied on a series of test problems and a comparison is made against some well-known methods, in Section 4 the package which implements the proposed method is documented from the installation procedure to some illustrative examples and in Section 5 some conclusions are derived regarding the proposed method and the results from the test problems.

2. Method description

2.1. Grammatical evolution

Grammatical evolution is an evolutionary algorithm that can produce code in any programming language. The algorithm requires the grammar of the target language in BNF syntax and the proper fitness function. Chromosomes in grammatical evolution, in contrast to classical genetic programming [13], are not expressed as parse trees, but as vectors of integers. Each integer denotes a production rule from the BNF grammar. The algorithm starts from the start symbol of the grammar and gradually creates the program string, by replacing non terminal symbols with the right hand of the selected production rule. The selection is performed in two steps:

- Read an element from the chromosome (with value V).
- Select the rule according to the scheme

$$\text{RULE} = V \bmod \mathcal{R}, \tag{1}$$

where $\mathcal{R}$ is the number of rules for the specific non-terminal symbol. The process of replacing non terminal symbols with the right hand of production rules is continued until either a full program has been generated or the end of chromosome has been reached. In the latter case we can reject the entire chromosome or we can start over (wrapping event) from the first element of the chromosome. In our approach we allow at most two wrapping events to occur. If the limit of two wrapping events is reached the chromosome is rejected. The rejection of a chromosome means that a large fitness value is assigned to the chromosome and as a consequence it will not be used in the crossover procedure. The grammatical evolution procedure has been used with success in many fields such as symbolic regression [14], discovery of trigonometric identities [15], robot control [16], caching algorithms [17], financial prediction [18], etc.

2.2. Chromosome creation

The grammatical evolution is used to create trial solutions using the grammar in Fig. 1. The numbers in parentheses denote the sequence number of the corresponding production rule to be used in the mapping procedure of grammatical evolution (see Eq. (1)). Dempsey et al. have also described another approach for the creation of constants through Grammatical Evolution using meta-grammars in their work [19]. The symbol START in the grammar denotes the start symbol of the grammar. This grammar can create valid double precision numbers in the range [0, 1]. These are brought inside the required range. For example consider the chromosome $c = [7, 11, 26, 12]$. In Table 1 the mapping procedure from the chromosome c to a valid double precision number (0.12) is presented.

2.3. Genetic operations

The genetic algorithm utilizes the operations of crossover and mutation to create the evolving generations.

2.3.1. Crossover

The crossover procedure is performed on each generation to create new chromosomes from the old ones. These will replace the worst individuals in the population. For every couple

Please cite this article in press as: I.G. Tsoulos, I.E. Lagaris, GenMin: An enhanced genetic algorithm for global optimization, Computer Physics Communications (2008), doi:10.1016/j.cpc.2008.01.040

```
<START>::=0. <digitlist>
<digitlist>::=<digit>              (0)
             |<digit><digitlist>  (1)
<digit>::=0    (0)
         | 1   (1)
         | 2   (2)
         | 3   (3)
         | 4   (4)
         | 5   (5)
         | 6   (6)
         | 7   (7)
         | 8   (8)
         | 9   (9)
```

Fig. 1. The used grammar of the algorithm.

Table 1
An example of the mapping procedure

Chromosome	Action	Expression
7, 11, 26, 12	7 mod 2 = 1	0.<digit><digitlist>
11, 26, 12	11 mod 10 = 1	0.1<digitlist>
26, 12	26 mod 2 = 0	0.1<digit>
12	12 mod 10 = 2	0.12
		0.12

of new chromosomes two parents are selected, we cut these chromosomes at a randomly chosen point and we exchange the right hand side sub chromosomes, as shown in Fig. 2. The parents are selected through tournament selection: We first create a group of $K \geqslant 2$ randomly selected chromosomes from the current population and the individual with the best fitness in the group is selected as a parent. Note that there exist in the relevant literature various crossover schemes adopted in Grammatical Evolution using subtrees [20,21].

2.3.2. Mutation

For every element of each chromosome a random number in the range [0, 1] is chosen. If this number is less than or equal to the mutation rate the corresponding element is changed by selecting a random integer in the range [0, 255], otherwise it remains intact.

2.4. Algorithm description

The main steps of the algorithm are:

- **Initialization** step. Here each chromosome is initialized at random from a uniform distribution inside the feasible region using the procedure described in Section 2.2 along with the following parameters:
  - **Set** $k = 0$. A counter for the number of the generations.
  - **Denote** by $(x_{\text{best}}, y_{\text{best}})$ the final output of the algorithm (the discovered global minimum).
  - **Set** $y_{\text{min}} = \infty$. This is the best function value discovered by the genetic algorithm.
  - **Set** $v_1 = 0$, $v_2 = 0$. These are auxiliary variables.
- **Genetic operations** step. Here the main steps of the genetic algorithm are taken:
  - **Calculate** the fitness for every chromosome in the population. For example, consider the objective function given by the equation $f(x) = x_1^2 + x_2^2$. Suppose that we seek the global minimum of this function in the range $[-1, 1]^2$ and let the chromosome be $c = [7, 11, 26, 12, 3, 4, 28, 7]$. The chromosome is split into two parts (the dimension of the objective function). The first part is denoted by $c_1 = [7, 11, 26, 12]$ and the second by $c_2 = [3, 4, 28, 7]$. The mapping procedure of Section 2.2 produces the numbers (0.12, 0.47) and by rescaling them to the range $[-1, 1]^2$ we have the final result $x = (-0.76, -0.06)$. The value of the objective function at this point is given by $f(x) = 0.5812$ and that is the fitness of the chromosome c.
  - **Apply** the genetic operators of crossover and mutation.
  - **Set** $k = k + 1$.
  - If $k <$ MAXITERS terminate.
- **Local search** step. The chromosome with the lowest value in the current generation (x_0, y_0) is compared to the chromosome $(x_{\text{min}}, y_{\text{min}})$, which has the lowest value among all the previous generations. If $y_0 < y_{\text{min}}$, we update: $x_{\text{min}} = x_0$, $y_{\text{min}} = y_0$ and subsequently a local search is started from x_{min}, yielding a local minimum (x^*, y^*). If $y^* < y_{\text{best}}$ we update: $x_{\text{best}} = x^*$, $y_{\text{best}} = y^*$.

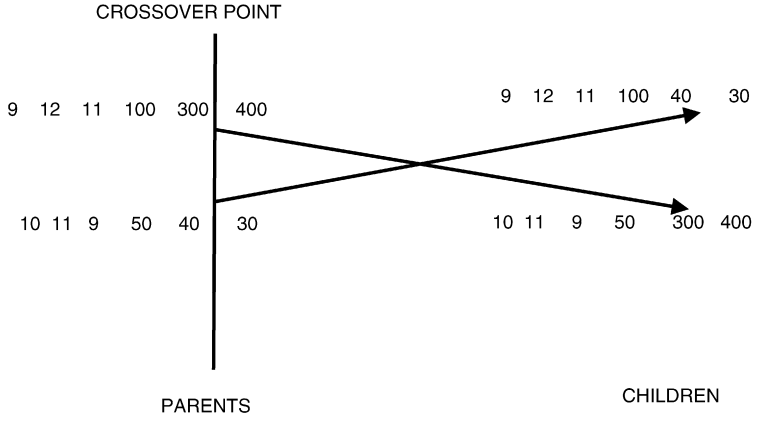

Fig. 2. One point crossover.

Please cite this article in press as: I.G. Tsoulos, I.E. Lagaris, GenMin: An enhanced genetic algorithm for global optimization, Computer Physics Communications (2008), doi:10.1016/j.cpc.2008.01.040

- **Termination** check step. At every iteration k, the best value $y_{\text{best}}^{(k)}$ is recorder. We calculate the variance $\sigma^{(k)}$ of this variable at every iteration k. When there is no progress for a number of iterations, we speculate that it is likely that $y_{\text{best}}^{(k)}$ is the global minimum and hence we would like to stop iterating. The stopping rule is then:

  stop if $\sigma^{(k)} < p\sigma^{(L)}$,

  where $p \in [0, 1]$ The parameter p controls the compromise between an exhaustive search ($p \to 0$) and a search optimized for speed ($p \to 1$). The suggested value is $p = 0.5$. The number L is the iteration where the current best point $y_{\text{best}}^{(k)}$ was first encountered.
- **Goto Genetic operations** step.

3. Experiments

3.1. Test problems

1. *Exponential function*.

$$f(x) = -\exp\left(-0.5\sum_{i=1}^{n} x_i^2\right), \quad -1 \leqslant x_i \leqslant 1$$

The global minimum is located at $x^* = (0, 0, \ldots, 0)$ and $f(x^*) = 0$. In our experiments we used this function with $n = 30$ and it is denoted by the label EXP.

2. *Zakharov function*.

$$f(x) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} \frac{i}{2} x_i\right)^2 + \left(\sum_{i=1}^{n} \frac{i}{2} x_i\right)^4,$$
$$-5.12 \leqslant x_i \leqslant 5.12.$$

The global minimum is located at $x^* = (0, 0, \ldots, 0)$ and $f(x^*) = 0$. In our experiments we used this function with $n = 10$.

3. *Rosenbrock function*.

$$f(x) = \sum_{i=1}^{n-1} \left(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right),$$
$$-30 \leqslant x_i \leqslant 30.$$

The global minimum is located at the $x^* = (0, 0, \ldots, 0)$ with $f(x^*) = 0$. In our experiments we used this function with $n = 50$.

4. *Ellipsoidal function*.

$$f(x) = \sum_{i=1}^{n} (x_i - i)^2, \quad -n \leqslant x_i \leqslant n.$$

The global minimum is located at the $x^* = (1, 2, \ldots, n)$ with $f(x^*) = 0$. In our experiments we used this function with $n = 10$. The function is denoted by the label ELP in the corresponding table.

5. *Sinusoidal function*.

$$f(x) = -\left(2.5\prod_{i=1}^{n} \sin(x_i - z) + \prod_{i=1}^{n} \sin\left(5(x_i - z)\right)\right),$$
$$0 \leqslant x_i \leqslant \pi.$$

The global minimum is located at $x^* = (2.09435, 2.09435, \ldots, 2.09435)$ with $f(x^*) = -3.5$. In our experiments we used $n = 10$ and $z = \frac{\pi}{6}$ and the function is denoted by the label SINU.

6. *Camel function*. The function is given by

$$f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4,$$
$$x \in [-5, 5]^2.$$

The global minimum has the value of $f(x^*) = -1.0316$.

7. *Rastrigin function*.

$$f(x) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2),$$
$$x \in [-1, 1]^2.$$

The global minimum is located at $x^* = (0, 0)$ with value -2.0.

8. *Griewank2 function*.

$$f(x) = 1 + \frac{1}{200}\sum_{i=1}^{2} x_i^2 - \prod_{i=1}^{2} \frac{\cos(x_i)}{\sqrt{(i)}},$$
$$x \in [-100, 100]^2.$$

The global minimum is located at the $x^* = (0, 0, \ldots, 0)$ with value 0.

9. *Gkls function*. $f(x) = \text{Gkls}(x, n, w)$, is a function with w local minima, described in [22], $x \in [-1, 1]^n$, $n \in [2, 100]$. In our experiments we use $n = 2, 3$ and $w = 50$.

10. *Goldstein & Price function*.

$$f(x) = \left[1 + (x_1 + x_2 + 1)^2 \times (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right] \times \left[30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right].$$

With $x \in [-2, 2]^2$. The global minimum is located at $x^* = (0, -1)$ with $f(x^*) = -3.0$.

11. *Test2N function*.

$$f(x) = \frac{1}{2}\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i, \quad x_i \in [-5, 5].$$

The function has 2^n local minima in the specified range and in our experiments we used $n = 4, 5, 6, 7$.

12. *Test30N function*.

$$f(x) = \frac{1}{10}\sin^2(3\pi x_1)\sum_{i=2}^{n-1}\left((x_i - 1)^2\left(1 + \sin^2(3\pi x_{i+1})\right)\right) + (x_n - 1)^2\left(1 + \sin^2(2\pi x_n)\right)$$

with $x \in [-10, 10]$. The function has 30^n local minima in the specified range and we used $n = 3, 4$ in our experiments.

Please cite this article in press as: I.G. Tsoulos, I.E. Lagaris, GenMin: An enhanced genetic algorithm for global optimization, Computer Physics Communications (2008), doi:10.1016/j.cpc.2008.01.040

Table 2
Experimental results

Function	SA	GSA	CRS	GCRS	GA	GENMIN
EXP(30)	72042	1380	89792	10019	13067	**573**
ELP(10)	24004	3849	29548	21361	49202	**436**
Zakharov(10)	24021	1960	27446	5737	34472	**483**
Rosenbrock(50)	120308	2058(0.73)	101590	92592	50788	**1464**
Sinu(10)	24042(0.93)	4459(0.93)	24855	12091	7184	**1212**
CAMEL	4820	1791	1852	1504	49639	**761**
RASTRIGIN	4843	488	1903	**428**	1640	750
GRIEWANK2	4832(0.27)	**580**	2105	977	4995(0.97)	764
GKLS(2, 50)	4820	1641	1627	1220	51013	**743**
GKLS(3, 50)	7228	2004	3349	2056	48972	**1941**
GOLDSTEIN	4842	1281	1923	961	20437	**748**
TEST2N4	9631	2923	6835(0.97)	4280(0.97)	2106	**1028**
TEST2N5	12034(0.87)	3456	25270(0.97)	7958	5711(0.93)	**1180**
TEST2N6	14438(0.66)	3633	32801(0.70)	9914	14109(0.73)	**1348**
TEST2N7	16840(0.37)	3840	38057(0.40)	9740	9639(0.87)	**1458**
TEST30N3	7930(0.23)	1425	3703	1519	1752	**508**
TEST30N4	9858(0.23)	1001	5135	1416	2758	**519**
POTENTIAL3	21404	3075	198046	9265	50915	**613**
POTENTIAL5	36212	2770	188646	9096	50662	**685**
NEURAL	76667(0.93)	**6241(0.93)**	122617	14559	52952(0.97)	9751

13. *Potential function*. The molecular conformation corresponding to the global minimum of the energy of N atoms interacting via the Lennard–Jones potential is determined for two cases: with $N = 3$ atoms and with $N = 5$ atoms. We refer to the first case as Potential(3) and to the second as Potential(5). The global minimum for the first is $f(x^*) = -3$ and $f(x^*) = -9.103852416$ for the second case.
14. *Neural network function*. A neural network (sigmoidal perceptron) with 10 hidden nodes (30 variables) was used for the least squares approximation of the function $g(t) = t\sin(t^2)$, $t \in [-2, 2]$. The error function is given by:

$$f(x,t) = \sum_{i=1}^{m} \big(N(x,t_i) - g(t_i)\big)^2,$$

where $N(x,t)$ is the output of the neural network with the weight vector denoted by x. This function is given by:

$$N(x,t) = \sum_{i=1}^{H} x_{3i-2}\sigma(x_{3i-1}t + x_{3i}),$$

where H is the number of hidden nodes (in our case $H = 10$). The constant m stands for the number of points in the training set (in our case $m = 100$). The global minimum of the error function is $f(x^*) = 0$.

3.2. Experimental setup

For the experiments we have used 100 chromosomes for the genetic population and each chromosome's length was set to $5n$, where n is the dimensionality of the objective function. The mutation rate was set to 5% and the selection rate to 95%. The maximum number of generations allowed (variable MAXITERS in the algorithm) was set to 500.

3.3. Results

In Table 2 we list the results from the application of Simulated Annealing (SA) as described in [5], Genetic Simulated Annealing (GSA) as described in [24], Controlled Random Search (CRS) of Price [26], Genetically Controlled Random Search (GCRS) as described in [23], a simple genetic algorithm (GA) as described in [27] (more specifically the algorithm named GA(c_r1, l)) with 100 chromosomes and maximum number of allowed generations set to 500 and the proposed method (GENMIN) on the test problems. The parameters for each of the algorithms are set according to the relevant literature. Each method was run 30 times for every problem using different random seeds. The local search method used in all methods was a BFGS variant due to Powell [25]. The numbers in the cells represent the average number of function evaluations spent. Not all of the runs discover the global minimum. In such cases we add in parentheses the fraction of successful runs.

3.4. Scalability of the method

In this subsection the method's scaling behavior is tested with respect to problem dimension using test functions suggested by Hansen et al. [28]. In Table 3 we present the results of our method for the functions EXP, ELP, ROSENBROCK and ZAKHAROV for $n = 2, 4, 8, 16, 32, 64, 100$. Similarly, in Table 4 the corresponding results for the standard Genetic Algorithm are presented. We counted the number of function evaluations versus the dimensionality of some appropriate test functions as proposed in [28]. In Figs. 3 and 4 we present graphically the scaling behavior of our method. We find that there is not a notorious dependence as in the case of Hansen et al. [28]. The only exception is a linear relation that seems to hold for the Sharp Ridge function.

Please cite this article in press as: I.G. Tsoulos, I.E. Lagaris, GenMin: An enhanced genetic algorithm for global optimization, Computer Physics Communications (2008), doi:10.1016/j.cpc.2008.01.040

Table 3
Scalability of the proposed method

	$n=2$	$n=4$	$n=8$	$n=16$	$n=32$	$n=64$	$n=100$
EXP	610	665	700	679	570	560	615
ELP	492	478	448	394	306	366	475
ROSENBROCK	581	688	787	982	1272	1551	1542
ZAKHAROV	534	539	502	469	441	531	564

Table 4
Scalability of the standard Genetic Algorithm

	$n=2$	$n=4$	$n=8$	$n=16$	$n=32$	$n=64$	$n=100$
EXP	1081	1309	1951	5324	11272	22839	33243
ELP	36199	46905	49310	48533	47126	45257	44153
ROSENBROCK	13908	30942	37408	48942	51239	51484	51771
ZAKHAROV	1282	3104	22376	50519	51013	51000	50281

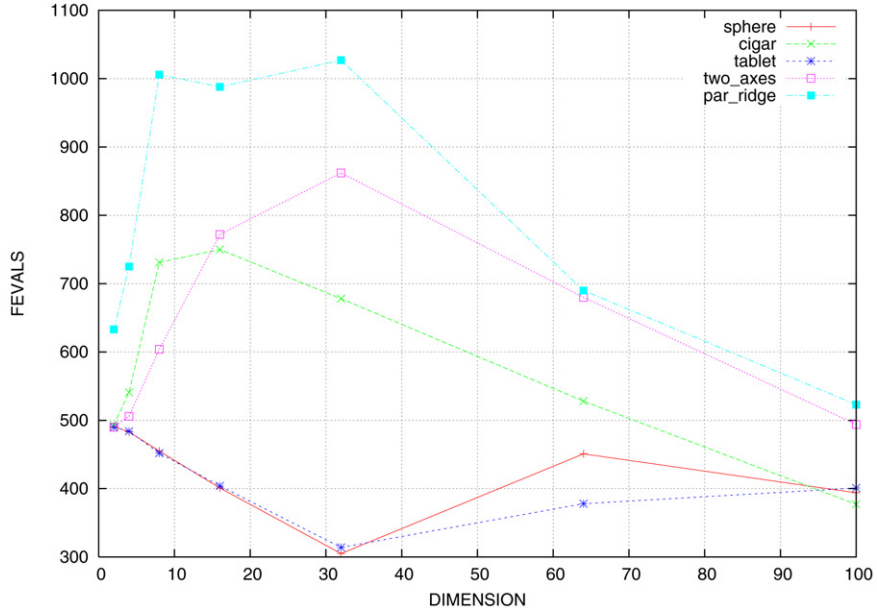

Fig. 3. Plot of average number of calls for functions: sphere, cigar, tablet, two_axes and par_ridge.

4. Software documentation

4.1. Distribution

The software package is distributed in a single tar.gz file named `GenMin.tar.gz` and it can be extracted under UNIX systems with the following commands:

1. gunzip `GenMin.tar.gz`
2. tar xfv `GenMin.tar`

The above step create the directory *GenMin* with the following contents:

1. **bin**: An empty directory that will contain the executable **make_genmin,** after the compilation of the package.
2. **doc**: A directory that contains the documentation of the package (this file) in different formats: A LyX file, a LaTeX file and a PostScript file.
3. **examples**: A directory with the test functions used in this article, coded in ANSI C++.
4. **include**: A directory which holds the header files needed for the compilation of the package.
5. **src**: This directory contains the source files for the compilation of the package.
6. **Makefile**: The main file used for the building of the tool. This file is used as input to the *make* utility, which is available in most UNIX systems. Usually, the user does not need to change this file.
7. **Makefile.inc**: This file contains some critical parameters for the compilation of the package, such as the name of the C++ compiler, the path of the tool, some linking options, etc. The user must edit and change this file before compilation, in order to meet his needs.

Please cite this article in press as: I.G. Tsoulos, I.E. Lagaris, GenMin: An enhanced genetic algorithm for global optimization, Computer Physics Communications (2008), doi:10.1016/j.cpc.2008.01.040

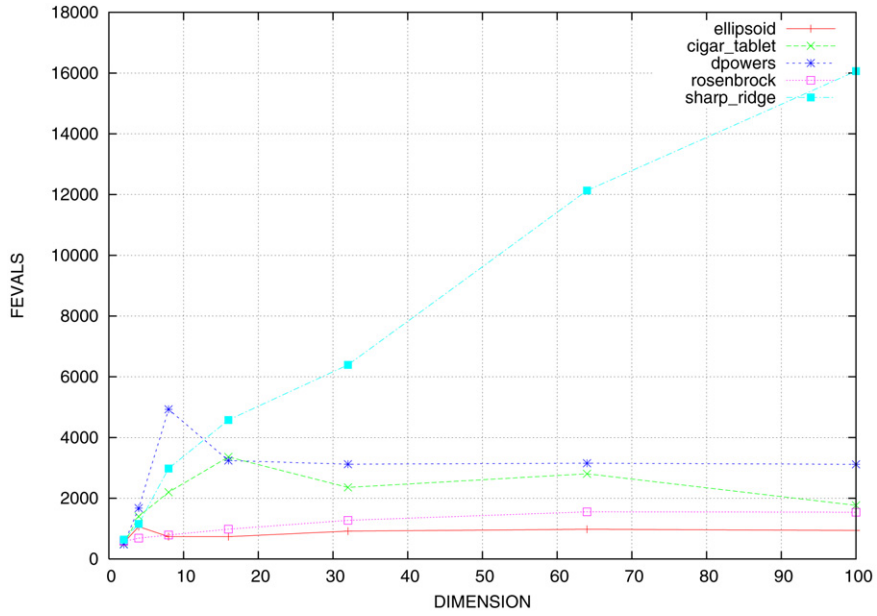

Fig. 4. Plot of average function evaluations for functions ellipsoid, cigar_tablet, dpowers, sharp_ridge and rosenbrock.

4.2. Installation

The following commands must be issued, in order to build the tool:

1. gunzip *GenMin.tar.gz*.
2. tar xfv *GenMin.tar*.
3. `cd GenMin`.
4. Change (if needed) the configuration parameters in the file `Makefile.inc`.
5. Type `make`.

The Makefile.inc contains the following compilation parameters:

1. **CXX**: This parameter specifies the name of the C++ compiler, that will be used for the compilation of the package. In most systems running the GNU C++ compiler this parameter must be set to g++.
2. **CC**: If the user written programs are in C, set this parameter to the name of the C compiler. Usually, for the GNU compiler suite, this parameter is set to gcc.
3. **F77**: If the user written programs are in Fortran 77, set this parameter to the name of the Fortran 77 compiler. For the GNU compiler suite a usual value for this parameter is g77.
4. **F77FLAGS**: The compiler GNU FORTRAN 77 (g77) appends an underscore to the name of all subroutines and functions after the compilation of a Fortran source file. In order to prevent this from happening we can pass some flags to the compiler. Normally, this parameter must be set to -fno-underscoring.
5. **ROOTDIR**: Is the location of the GenMin directory. It is critical for the system that this parameter is set correctly. In most systems, it is the only parameter which must be changed.
6. **LINK**: This parameter specifies additional linking options, such as external libraries etc. For example if we want to link the final executable against the library foo located in /home/user/libraries directory this parameter must be set to "-L/home/user/libraries -lfoo". The surrounding double quotes are necessary for the compilation. The default value for this parameter is set to "-lg2c" in order to enable FORTRAN Input Output commands in the user written subprograms.
7. **INCL**: This parameter specifies additional directories, where may be located needed header files. For example, the user can direct the program to search for header files under the subdirectory /home/user/include by setting this parameter to "-I/home/user/include".

4.3. User written subprograms

The user can code his objective function either in C, C++ or in Fortran77 in a single file. Each file has a series of functions in an arbitrary order. However, the C++ files must have the lines

```
extern "C" {
```

before the functions and the line

```
}
```

after them. The meaning of the functions are the following:

1. **getdimension**(): This integer function returns the dimension of the problem.
2. **getleftmargin**(left): It is a subroutine (or a void function in C) which fills the double precision array left with the lower bounds of the variables.
3. **getrightmargin**(right): Is a subroutine (or a void function in C) which fills the double precision array right with the upper bounds of the variables.

Please cite this article in press as: I.G. Tsoulos, I.E. Lagaris, GenMin: An enhanced genetic algorithm for global optimization, Computer Physics Communications (2008), doi:10.1016/j.cpc.2008.01.040

4. **funmin**(x): It is a double precision function which returns the value of the objective function evaluated at point x.
5. **granal**(x, g): It is a subroutine (or a void function in C) which returns in a double precision array g the gradient of the objective function at point x.

4.4. The utility make_genmin

The executable make_genmin will be placed after the compilation of the package under the subdirectory *bin*. This program is used to create the final executable from the objective function and it accepts the following command line parameters:

1. `-h`: Prints a help screen and terminates.
2. `-p` **filename**: The **filename** parameter specifies the name of the file containing the objective function. The utility checks the suffix of the file and it uses the appropriate compiler. If this suffix is .cc or .c++ or .CC or .cpp, then it invokes the C++ compiler. If the suffix is .f or .F or .for then it invokes the Fortran 77 compiler. Finally, if the suffix is .c it invokes the C compiler.
3. `-o` **filename**: The **filename** parameter specifies the name of the final executable. The default value for this parameter is GenMin.

4.5. The utility GenMin

The final executable GenMin has the following command line parameters:

1. `-h`: The program prints a help and it terminates.
2. `-c count`: The integer parameter `count` specifies the number of chromosomes used by the genetic algorithm. The default value for this parameter is 100.
3. `-s srate`: The double parameter `srate` specifies the selection rate used in genetic algorithm. The default value for this parameter is 0.10 (10%).
4. `-m mrate`: The double parameter `mrate` specifies the mutation rate used in the genetic algorithm. The default value for this parameter is 0.05 (5%).
5. `-r seed`: The integer parameter `seed` specifies the seed for the random number generator. The default value for this parameter is 1.
6. -g **generations**: The integer parameter **generation** specifies the maximum number of generations allowed. The default value for this parameter is 500.

4.6. A working example

Consider the Rastrigin function which is given by:

$$f(x_1, x_2) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2).$$

This function in the range $[-1, 1]^2$ has 49 local minima and the global minimum is located at (0, 0) with function value −2. The function is coded in C++ and is listed in Fig. 5 and resides in the *examples* subdirectory. In order to apply the proposed method to the Rastrigin function we issue the following commands

```
../bin/make\_genmin -p rastrigin.cc
./GenMin
```

```
# include <math.h>
extern "C"
{
  int getdimension()
  {
    return 2;
  }

  void getleftmargin(double *x)
  {
    x[0]=-1;
    x[1]=-1;
  }

  void getrightmargin(double *x)
  {
    x[0]=1;
    x[1]=1;
  }

  double funmin(double *x)
  {
    return x[0]*x[0]+x[1]*x[1]-cos(18.0*x[0])-cos(18.0*x[1]);
  }

  void granal(double *x,double *g)
  {
    g[0]=2.0*x[0]+18.0*sin(18.0*x[0]);
    g[1]=2.0*x[1]+18.0*sin(18.0*x[1]);
  }
}
```

Fig. 5. The Rastrigin function code.

```
ITER=   1       BEST VALUE=   -1.5156
VARIANCE=   0.57426     WILL STOP=   0.28713
ITER=   2       BEST VALUE=        -2
VARIANCE=   0.72574     WILL STOP=   0.36287
ITER=   3       BEST VALUE=        -2
VARIANCE=    0.6729     WILL STOP=   0.36287
ITER=   4       BEST VALUE=        -2
VARIANCE=   0.60004     WILL STOP=   0.36287
ITER=   5       BEST VALUE=        -2
VARIANCE=   0.53432     WILL STOP=   0.36287
ITER=   6       BEST VALUE=        -2
VARIANCE=   0.47898     WILL STOP=   0.36287
ITER=   7       BEST VALUE=        -2
VARIANCE=   0.43289     WILL STOP=   0.36287
ITER=   8       BEST VALUE=        -2
VARIANCE=   0.39432     WILL STOP=   0.36287
ITER=   9       BEST VALUE=        -2
VARIANCE=   0.36174     WILL STOP=   0.36287
X*=[ 0 0 ]  Y*=-2
FUNCTION CALLS =   920  GRADIENT CALLS=    43
```

Fig. 6. Output from the minimization of the Rastrigin function.

The output from the above command is listed in Fig. 6. Note that the minimization stops after 9 iterations and the program prints apart from the located minimum the function and gradient calls as well. In each iteration the program prints the value

of the variance of the best located value as well as the stopping value (variable s of the proposed algorithm).

5. Conclusions

A new method is proposed in this article that aims to locate global minimum of a continuous multidimensional, multimodal function. The user can code his problems either in C++ or in Fortran77 programming language. The method has been applied to a series of well-known optimization problems and it seems to be far more efficient compared to other established methods.

References

[1] H.A. Bremermann, Mathematical Biosciences 9 (1970) 1–15.
[2] R.A. Jarvis, IEEE Trans. on Syst., Man and Cybergenetics 75 (1975) 297–311.
[3] W.L. Price, Computer Journal 20 (1977) 367–370.
[4] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Science 220 (1983) 671–680.
[5] A. Corana, M. Marchesi, C. Martini, S. Ridella, ACM Transactions on Mathematical Software 13 (1987) 262–280.
[6] W.L. Goffe, G.D. Ferrier, J. Rogers, J. Econometrics 60 (1994) 65–100.
[7] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Company, Reading, MA, 1989.
[8] Z. Michaelewizc, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, 1996.
[9] R. Storn, K. Price, Journal of Global Optimization 11 (1997) 341–359.
[10] M.M. Ali, L.P. Fatti, Journal of Global Optimization 35 (2006) 551–572.
[11] D. Cvijoivic, J. Klinowski, Science 667 (1995) 664–666.
[12] M. O'Neill, C. Ryan, IEEE Trans. Evolutionary Computation 5 (2001) 349–358.
[13] J.R. Koza, Genetic Programming: On the programming of Computer by Means of Natural Selection, MIT Press, Cambridge, MA, 1992.
[14] M. O'Neill, C. Ryan, Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, Genetic Programming, vol. 4, Kluwer Academic Publishers, 2003.
[15] C. Ryan, M. O'Neill, J.J. Collins, in: Proceedings of Mendel 1998: 4th International Mendel Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets, Brno, Czech Republic, June 24–26, 1998, Technical University of Brno, Faculty of Mechanical Engineering, pp. 111–119.
[16] M. O'Neill, J.J. Collins, C. Ryan, in: Proceedings of AROB 2000, the Fifth International Symposium on Artificial Life and Robotics, pp. 351–354.
[17] M. O'Neill, C. Ryan, in: K. Miettinen, M.M. Mkel, P. Neittaanmki, J. Periaux (Eds.), Proceedings of Evolutionary Algorithms in Engineering and Computer Science, Jyvskyl, Finland, 1999, pp. 127–134.
[18] A. Brabazon, M. O'Neill, Biologically Inspired Algorithms for Financial Modelling, Springer, 2006.
[19] I. Dempsey, M. O'Neill, A. Brabazon, International Journal of Innovative Computing and Applications 1 (2007) 23–38.
[20] M. O'Neill, C. Ryan, M. Keijzer, M. Cattolico, Genetic Programming and Evolvable Machines 4 (2003) 67–93.
[21] R. Harper, A. Blair, in: D. Corne, Z. Michalewicz, M. Dorigo, G. Eiben, D. Fogel, C. Fonseca, G. Greenwood, T.K. Chen, G. Raidl, A. Zalzala, S. Lucas, B. Paechter, J. Willies, J.J. Merelo Guervos, E. Eberbach, B. McKay, A. Channon, A. Tiwari, L. Gwenn Volkert, D. Ashlock, M. Schoenauer (Eds.), Proceedings of the 2005 IEEE Congress on Evolutionary Computation, vol. 3, Edinburgh, UK, 2–5 September 2005, IEEE Press, pp. 2537–2544.
[22] M. Gaviano, D.E. Ksasov, D. Lera, Y.D. Sergeyev, ACM Trans. Math. Softw. 29 (2003) 469–480.
[23] I.G. Tsoulos, I.E. Lagaris, Comput. Phys. Comm. 174 (2006) 152–159.
[24] I.G. Tsoulos, I.E. Lagaris, Comput. Phys. Comm. 174 (2006) 846–851.
[25] M.J.D. Powell, Mathematical Programming 45 (1989) 547.
[26] W.L. Price, Computer Journal 20 (1977) 367–370.
[27] P. Kaelo, M.M. Ali, European Journal of Operationals Research 176 (2007) 60–76.
[28] N. Hansen, S.D. Müler, P. Koumoutsakos, Evolutionary Computation 11 (2003) 1–18.

Please cite this article in press as: I.G. Tsoulos, I.E. Lagaris, GenMin: An enhanced genetic algorithm for global optimization, Computer Physics Communications (2008), doi:10.1016/j.cpc.2008.01.040