GenAnneal: Genetically modified Simulated Annealing

Ioannis G. Tsoulos[1], Isaac E. Lagaris[2*]

[1]Department of Computer Science, University of Ioannina, P.O. Box 1186, Ioannina 45110 - GREECE
[2]Physics Department, University of South Africa (UNISA), Pretoria RSA.

Abstract

A modification of the standard Simulated Annealing (SA) algorithm is presented for finding the global minimum of a continuous multidimensional, multimodal function. We report results of computational experiments with a set of test functions and we compare to methods of similar structure. The accompanying software accepts objective functions coded both in Fortran 77 and C++.

PACS: 02.60.-x ; 02.60.Pn ; 07.05.Kf; 02.70.Lq; 07.05.Mh
Keywords: Global optimization, genetic programming, grammatical evolution, simulated annealing.

PROGRAM SUMMARY

Title of program: GenAnneal

Catalogue identifier:

Program available 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: The tool is designed to be portable in all systems running the GNU C++ compiler.

Installation: University of Ioannina, Greece on Linux based machines.

Programming language used: GNU-C++, GNU-C, GNU Fortran - 77.

*Permanent address: Department of Computer Science, University of Ioannina, P.O. Box 1186, Ioannina 45110 - GREECE. E-mail: lagaris@cs.uoi.gr

Memory required to execute with typical data: 200KB.

No. of bits in a word: 32

No. of processors used: 1

Has the code been vectorised or parallelized?: No.

No. of bytes in distributed program,including test data etc.: 100 Kbytes.

Distribution format: gzipped tar file.

Keywords: Global optimization, stochastic methods, genetic programming, grammatical evolution.

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 are frequently trapped in local minima. Global optimization is hence the appropriate tool. For example, solving a non - linear 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).

Typical running time: Depending on the objective function.

Method of solution: We modified the process of step selection that the traditional Simulated Annealing employs and instead we used a global technique based on grammatical evolution.

LONG WRITE UP

1 Introduction

In this article we deal with a modification of the Simulated Annealing (SA) algorithm [1, 2, 3, 4] that results in a performance enhancement. We first give a brief description of the standard SA method for locating a global minimum of a continuous, differentiable function f with many variables inside a bounded domain $S \subset R^n$. SA is based on an analogy with the physical phenomenon of annealing. In a liquid the molecules move almost freely when the temperature is high. If the temperature is lowered gradually and slowly enough, then the system crystallizes into a state of minimum energy. If the temperature is lowered quickly , (this is called quenching) the system settles into a state of higher energy. A typical description of SA algorithm is as follows:

1. Initially **Set** $k = 0, \;\; T_0 > 0$. Sample $x_0 \in S$.

2. **Sample** a point $y \in S$.

3. **Set** $x_{k+1} = x_k$.

4. **Reset** $x_{k+1} = y$, with probability $\min\left\{1, \exp\left(-\frac{f(y)-f(x_k)}{T_k}\right)\right\}$

5. **Set** $k = k + 1$

6. **Lower** the temperature T_k using an annealing schedule.

7. **Goto** step 2.

When the temperature T_k is high, the method performs a global exploration; however, when the temperature T_k tends to 0, the behavior of the algorithm closely resembles that of a local search. The new method proposes a modification in step 2 and samples a new point utilizing the GRS technique [5]. The GRS method produces steps by employing genetic programming and more specifically grammatical evolution [6, 7, 8], and it is briefly reviewed in subsection 2.1.

The rest of this article is organized as follows: in section 2 the proposed algorithm is presented in detail. In section 3 some experimental results from the application of the proposed method are listed and a comparison is made against traditional simulated annealing methods and in section 4 the installation and the execution procedures of the GenAnneal are presented.

2 Description of the algorithm

2.1 The GRS procedure

The main steps of the GRS procedure are the following:

INPUT Data:

- A point $x = (x_1, x_2, \ldots, x_n)$, $x \in S \subset R^n$.
- ϵ, a small positive number. Typical values for this parameter are 10^{-4} to 10^{-5}.
- **Set** k as the maximum number of allowed generations.
- **Set** selection rate and mutation rate.

INITIALIZATION step:

- The initialization of each element of the genetic population is performed by selecting a random integer in the range [0,255].

LOOP Step:

- **For** $i = 1, ..., k$ **Do**

- **Set** $x_{\text{old}} = x$.
- **Create** a new generation of chromosomes in the population with the use of the genetic operations (crossover, mutation, reproduction).
- **Set** y=f(x).
- **For** every chromosome **Do**
  - **Split** the chromosome uniformly into n parts $(p_i,\ i = 1, \ldots, n)$, one for each dimension. On every piece p_i the grammatical evolution transformation is applied to generate a univariate function f_i.
  - **Denote** by d the vector $(d_1 = f_1(x_1), d_2 = f_2(x_2), ..., d_n = f_n(x_n))$.
  - **Set** $x_+ = x + d$.
  - **If** $x_+ \notin S$ or $f(x_+) > y$ **then**
    - **Set** $x_- = x - d$.
    - **If** $x_- \notin S$ or $f(x_-) > y$, **then**

      Set the fitness value to a very large number.
    - **Else**

      Set the fitness value to $f(x_-)$.
    - **Endif**
  - **Else**
    - **Set** the fitness value to $f(x_+)$.
  - **Endif**
- **Endfor**
- **Set** $x = x + d_{\text{best}}$, where d_{best} the movement that corresponds to the chromosome with the best fitness value.
- **If** $|x - x_{\text{old}}| \leq \epsilon$, terminate and **return** x as the located minimizer.

- **Endfor**
- **Return** x as the located minimizer.

2.2 The proposed algorithm

The steps of the proposed algorithm are presented below:

Initialization_Step

- **Set** values for the parameters TLAST, T and ϵ. The parameters TLAST and ϵ control the termination of the algorithm. The algorithm terminates when either the best value was not changed in the last TLAST iterations, or the temperature T has dropped below ϵ.
- **Set** the value of the temperature reduction factor $a \in (0, 1)$.

- **Sample** uniformly a point $x \in S$.
- **Set** $f_c = f(x)$, $x_{\text{best}} = x$, $f_{\text{best}} = f_c$.

Main_step

- **Start** the GRS procedure from point x and denote by $\tilde{x}$ the resulting point.
- **If** $f(\tilde{x}) < f_c$ **then**
  - **Set** $x = \tilde{x}$, $f_c = f(\tilde{x})$
  - **If** $f_c < f_{\text{best}}$ **then**
    - **Set** $x_{\text{best}} = x$, $f_{\text{best}} = f_c$
  - **EndIf**
- **Else**
  - **If** $e^{\frac{f(\tilde{x})-f_c}{T}} > \xi$, where ξ a random number in the range (0,1) **then**
    - **Set** $x = \tilde{x}$**,** $f_c = f(\tilde{x})$
  - **EndIf**
- **EndIf**

Update_Step

- **Update** the temperature T according to the scheme $T = aT$.
- **If** there was no improvement during the last TLAST iterations or **if** the temperature T has dropped below ϵ, **go to Main_Step else goto Optimization_Step**.

Optimization_Step:

- Start a local search at x and let the resulting point be defined by x^*.
- **Return** x^*.

3 Experimental results

The suggested approach was compared against two publicly available variants of Simulated Annealing: The method of Corana [3] (labeled SA in Table 1) and the Adaptive Simulated Annealing of Ingber [4] (labeled ASA in Table 1) . The comparison is performed on a suite of test problems listed in Appendix A. Each method was run 50 times on every problem using different random seeds each time. We have measured the percentage of times the global minimum was found and the number of function evaluations it required. The parameters for the proposed method are listed in table 2. Both for SA and ASA we used the default parameters as suggested in the provided software packages available from the sites http://www.netlib.org and http://www.ingber.com respectively. The trial steps produced by the grammatical evolution were evaluated using the FunctionParser programming library [9]. The local search procedure used in all methods was a BFGS implementation due to Powell [10]. In table 1 we list the results for SA, ASA and the proposed method (GSA). Numbers in the cells represent the average number of function evaluations required by each method. The figures in parentheses denote the fraction of runs during which the global minimum was discovered. Absence of this number denotes that the global minimum was recovered in every single run. From the reported results one can easily deduce the superiority of the new method as far as the above benchmarks are concerned. Hence, we recommend its use for solving difficult practical problems.

Table 1: Average number of function evaluations for SA, ASA and GSA.

FUNCTION	SA	ASA	GSA
CAMEL	4820	3125	1791
RASTRIGIN	4843	3534	488
GRIEWANK2	4832(0.27)	3271(0.43)	580
GKLS(2,50)	4820	3354	1641
GKLS(3,50)	7228	5269(0.17)	2004
GOLDSTEIN	4842	3385(0.93)	1281
TEST2N(4)	9631	6460	2923
TEST2N(5)	12034(0.87)	10763	3456
TEST2N(6)	14438(0.66)	18466	3633
TEST2N(7)	16840(0.37)	29972	3840
TEST30N(3)	7930(0.23)	3220	1425
TEST30N(4)	9858(0.23)	6002	1001
POTENTIAL(3)	21404	50202	3075
POTENTIAL(5)	36212	80527	2770
NEURAL	76667(0.93)	67167	6241(0.93)

4 Software documentation

4.1 Distribution

The package is distributed in a tar.gz file named `GenAnneal.tar.gz` and under UNIX systems the user must issue the following commands to extract the associated files:

1. gunzip `GenAnneal.tar.gz`

Table 2: GenAnneal parameters

PARAMETER	VALUE
SELECTION RATE	90%
MUTATION RATE	5%
CHROMOSOME COUNT	100
CHROMOSOME LENGTH	$10 \times$ problem dimensionality
ϵ	10^{-5}
TLAST	4

2. tar xfv `GenAnneal.tar`

These steps create a directory named `GenAnneal` with the following contents:

1. **bin**: A directory which is initially empty. After compilation of the package, it will contain the executable **make_genanneal**
2. **examples**: A directory that contains the test functions used in this article, written in ANSI C++.
3. **include**: A directory which contains the header files for all the classes of the package.
4. **src**: A directory containing the source files of the package.
5. **Makefile**: The input file to the `make` utility in order to build the tool. Usually, the user does not need to change this file.
6. **Makefile.inc**: The file that contains some configuration parameters, such as the name of the C++ compiler etc. The user must edit and change this file before installation.

4.2 Installation

The following steps are required in order to build the tool:

1. Uncompress the tool as described in the previous section.
2. `cd GenAnneal.`
3. Edit the file `Makefile.inc` and change (if needed) the five configuration parameters.
4. Type `make`.

The five parameters in `Makefile.inc` are the following:

1. **CXX**: It is the most important parameter. It specifies the name of the C++ compiler. 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 GenAnneal 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.

4.3 User written subprograms

The user can write 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**(): It is an integer function which returns the dimension of the objective function.
2. **getleftmargin**(left): It is a subroutine (or a void function in C) which fills the double precision array left with the left margins of the objective function.
3. **getrightmargin**(right): Is is a subrourine (or a void function in C) which fills the double precision array right with the right margins of the objective function.
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_genanneal

After the compilation of the package, the executable `make_genanneal` will be placed in the subdirectory `bin` in the distribution directory. This program creates the final executable and it takes 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 GenAnneal..

4.5 The utility GenAnneal

The final executable `GenAnneal` 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 for the Genetic Random Search procedure. The default value for this parameter is 20.
3. `-s srate`: The double parameter `srate` specifies the selection rate used in the Genetic Random Search procedure. 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 Random Search procedure. 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. It can assume any integer value.
6. `-o filename`: The parameter `filename` specifies the file where the output from the GenAnneal will be placed. The default value for this parameter is the standard output.

4.6 A working example

Consider the Rastrigin function given by

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

with 49 local minima. The implementation of this function in C++ and in Fortran77 is shown in examples 1 and 2. Let the file with the C++ code be named `rastrigin.cc` and that with the Fortran code `rastrigin.f`. Let these files be located in the `examples` subdirectory. Change to the `examples` subdirectory and create the GenAnneal executable with the `make_genanneal` command:

```
../bin/make_genanneal -p rastrigin.cc
```

or for the Fortran 77 version

```
../bin/make_genanneal -p  rastrigin.f
```

The `make_genanneal` responds:

```
RUN ./GenAnneal IN ORDER TO RUN THE PROBLEM
```

Run GenAnneal by issuing the command:

```
./GenAnneal  -c 100 -r 1
```

The resulting output appears as:

```
FUNCTION EVALUATIONS = 468
GRADIENT EVALUATIONS = 1
MINIMUM =  0.000000  0.000000  -2.000000
```

A. Test Functions

The functions used for testing accompanied with the corresponding parameter range and the global minimum are listed below. Further information may be found in Floudas et al [11] as well as in the URL:

`http://www.imm.dtu.dk/~km/GlobOpt/testex/testproblems.html.`

Camel

$f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$, $x \in [-5,5]^2$ with 6 local minima and global minimum $f^* = -1.031628453$.

Rastrigin

$f(x) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2)$, $x \in [-1,1]^2$ with 49 local minima and global minimum $f^* = -2.0$.

Griewank2

$f(x) = 1 + \frac{1}{200}\sum_{i=1}^{2} x_i^2 - \prod_{i=1}^{2} \frac{\cos(x_i)}{\sqrt{(i)}}$, $x \in [-100, 100]^2$ with 529 loca minima and global minimum $f^* = 0.0$

Gkls

$f(x) = \text{Gkls}(x, n, w)$, is a function with w local minima, described in [12], $x \in [-1, 1]^n$, $n \in [2, 100]$. In our experiments we use $n = 2, 3$ and $w = 50$.

GoldStein & Price

$$\begin{aligned} f(x) &= [1 + (x_1 + x_2 + 1)^2 \\ &\quad (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times \\ &\quad [30 + (2x_1 - 3x_2)^2 \\ &\quad (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \end{aligned}$$

The function has 4 local minima in the range $[-2, 2]^2$ and global minimum $f^* = 3.0$.

Test2N

$$f(x) = \frac{1}{2}\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i$$

with $x \in [-5, 5]^n$. The function has 2^n local minima in the specified range. In our experiments we used the cases of $n = 4, 5, 6, 7$.

Test30N

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

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

Potential

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)** (a problem with 9 variables) and to the second as **Potential(5)** (a problem with 15 variables). The global minimum for the first cases is $f^* = 3$ and $f^* = -9.103852416$

Neural

A neural network (sigmoidal perceptron) with 10 hidden nodes (30 variables) was used for the approximation of the function $g(x) = x \sin\left(x^2\right),\ x \in [-2, 2]$. The global minimum of the training error is $f^* = 0.0$

B. Examples

Example 1: Implementation of Rastrigin function in C++.

```
extern "C"{
 int getdimension()
 {
   return 2;
 }

 void getleftmargin(double *left)
 {
   left[0]=-1.0;
   left[1]=-1.0;
 }

 void getrightmargin(double *right)
 {
   right[0]=1.0;
   right[1]=1.0;
 }

 double funmin(double *x)
 {
   double x1=x[0],x2=x[1];
   return x1*x1+x2*x2-cos(18.0*x1)-cos(18.0*x2);
 }

 void granal(double *x,double *g)
 {
   double x1=x[0],x2=x[1];
   g[0]=2.0*x1+18.0*sin(18.0*x1);
   g[1]=2.0*x2+18.0*sin(18.0*x2);
 }
}
```

Example 2: Implementation of Rastrigin function in Fortran 77.

```
integer function getdimension()
getdimension = 2
end

subroutine getleftmargin(left)
double precision left(2)
left(1)=-1.0
left(2)=-1.0
end

subroutine getrightmargin(right)
double precision right(2)
right(1)= 1.0
right(2)= 1.0
end

double precision function funmin(x)
double precision x(2)
double precision x1,x2
x1=x(1)
x2=x(2)
funmin=x1**2+x2**2-cos(18.0*x1)-cos(18.0*x2)
end

subroutine granal(x,g)
double precision x(2)
double precision g(2)
double precision x1,x2
x1=x(1)
x2=x(2)
g(1)=2.0*x1+18.0*sin(18.0*x1)
g(2)=2.0*x2+18.0*sin(18.0*x2)
end
```

References

[1] Kirkpatrick S., Gelatt C. D. and Vecchi M. P., Optimization by simulated annealing, Science **220**, 671-680 (4), 1983.

[2] P. J. M. van Laarhoven and E. H. L. Aarts, "Simulated Annealing: Theory and Applications", 1987, D. Riedel, Boston.

[3] Corana A., Marchesi M., Martini C. and Ridella S., Minimizing Multimodal Functions of Continuous Variables with the "Simulated Annealing" Algorithm, ACM Transactions on Mathematical Software, Vol. **13**, pp. 262-280, 1987.

[4] Ingber L., Simulated Annealing: Practice versus theory, J. Math. Comput. Modelling **18**, pp. 29-57,1983.

[5] Tsoulos I.G. and Lagaris I.E., Genetically Controlled Random Search: A global optimization method for continuous multidimensional functions, to be appear on Computer Physics Communications.

[6] M. O'Neill, Automatic Programming in an Arbitrary Language: Evolving programs with grammatical evolution, PhD Thesis, University of Limerick, Ireland, August 2001.

[7] M. O'Neill and C. Ryan, Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, volume 4 of Genetic programming. Kluwer Academic Publishers, 2003.

[8] M. O'Neill and C. Ryan, "Grammatical Evolution," IEEE Trans. Evolutionary Computation, Vol. 5, pp. 349-358, 2001.

[9] Nieminen J. and Yliluoma J., "Function Parser for C++, v2.7", available from http://www.students.tut.fi/˜warp/FunctionParser/.

[10] Powell M. J. D., "A Tolerant Algorithm for Linearly Constrained Optimization Calculations", Mathematical Programming 45 (1989), 547.

[11] Floudas C. A. and Pardalos P.M., Adjiman C., Esposoto W., Gümüs Z., Harding S., Klepeis J., Meyer C, Schweiger C., Handbook of Test Problems in Local and Global Optimization, Kluwer Academic Publishers, Dordrecht, 1999.

[12] Gaviano M., Ksasov D. E., Lera D. and Sergeyev, Y. D. Software for generation of classes of test functions with known local and global minima for global optimization, ACM Trans. Math. Softw. **29**, pp. 469-480, 2003.