MDPI

Article

OPTIMUS: a multidimensional global optimization package

Ioannis G. Tsoulos[1,*], Vasileios Charilogis[2], V.N. Stavrou[3], Alexandros Tzallas[4]

[1] Department of Informatics and Telecommunications, University of Ioannina, 45110 Ioannina, Greece;itsoulos@uoi.gr
[2] Department of Informatics and Telecommunications, University of Ioannina, 45110 Ioannina, Greece;v.charilog@uoi.gr
[3] Division of Physical Sciences, Hellenic Naval Academy, Military Institutions of University Education, 18539 Piraeus, Greece; vstavrou@hna.gr
[4] Department of Informatics and Telecommunications, University of Ioannina, 45110 Ioannina, Greece;tzallas@uoi.gr
* Correspondence: itsoulos@uoi.gr;
‡ These authors contributed equally to this work.

Abstract: A significant number of applications from many research areas can be considered global optimization problems, such as applications in the area of image processing, medical informatics, economic models, etc. This paper presents a programming tool written in ANSI C++, which researchers can use to formulate the problem to be solved and then make use of the local and global optimization methods provided by this tool to efficiently solve such problems. The main features of the suggested software are: a) Coding of the objective problem in a high level language such as ANSI C++ b) Incorporation of many global optimization techniques to tackle the objective problem c)Parameterization of global optimization methods using user-defined parameters.

Keywords: Global optimization; stochastic methods;

1. Introduction

The location of the global minimum for a continuous and differentiable function $f : S \to R, S \subset R^n$ is formulated as

$$x^* = \arg\min_{x \in S} f(x) \tag{1}$$

where the set S is defined as:

$$S = [a_1, b_1] \otimes [a_2, b_2] \otimes \ldots [a_n, b_n]$$

Methods that aim to locate the global minimum finds application in problems from the area of economics [1,2], problems that appear very often in the area of physics [3,4], chemistry [5,6], common problems from medicine [7,8], job scheduling problems [9,10], water resources planning [11,12], network security problems [13,14], robotics [15,16] etc. Also, global optimization methods were used on some symmetry problems [17–19] as well as on inverse problems [20–22]. In the relevant literature there are a number of global optimization techniques, such as Adaptive Random Search methods [23,24], Controlled Random Search methods [25,26], Simulated Annealing [27–29], Genetic algorithms [30,31], Ant Colony Optimization [32,33], Particle Swarm Optimization [34,35] etc.

Due to the high importance of the global optimization problem, a variety of hybrid optimization techniques have been proposed to handle the global optimization problem, such as methods that combine Particle Swarm Optimization and Genetic algorithms [36,37], combination of genetic algorithms and fuzzy logic classifier [38], incorporation of genetic algorithm and the K-Means algorithm [39], combination of Particle Swarm Optimization method with Ant Colony Optimization [40–42], methods that combine the Simplex method

Citation: Tsoulos, I.G.; Charilogis V.; Stavrou V.N.; Tzallas A. OPTIMUS: a multidimensional global optimization package. *Journal Not Specified* **2023**, *1*, 0. https://doi.org/

Received:
Revised:
Accepted:
Published:

and Inductive search [43] etc. Also, many hybrid techniques combining local and global optimization have been developed [44–46].

Just a few recent application examples include an adaptive genetic algorithm for crystal structure prediction [47], modeling of fusion plasma physics with genetic algorithms [48], usage of genetic algorithms for astroparticle physics studies [49], parameter extraction of solar cells using a Particle Swarm Optimization method [50], a new control approach of a a fleet of Unmanned Aerial Vehicles using the method of Particle Swarm Optimization[51] etc.

However, in most cases, global optimization methods require a lot of computing resources to implement both in memory and computing time. Because of the large demands that global optimization methods have on computing power, several techniques have been proposed, such as asynchronous methods [52–54], parallel approaches of the Multistart optimization method [55,56] and also some methods that take advantage of modern parallel GPU architectures [57–59].

In this paper, a new integrated computing environment for performing global optimization methods for multidimensional functions is presented and analyzed in detail. In this computing environment, the programmer can code the problem to be solved using a high-level programming language such as C++. In addition to the objective function, the programmer can also provide information that the objective problem should have at the start of the optimization process and, in addition, can formulate a series of actions that will take place after the optimization process is finished. Subsequently, the researcher can formulate a strategy to solve the problem. In this strategy, the researcher can choose from a series of sampling methods, choose a global minimization method established in the relevant literature and possibly some local minimization method to improve the produced result. Similar software environments can be found, such as the BARON software package [60] for non-convex optimization problems, the MERLIN optimization software [61] which is accompanied by the Merlin Control Language compiler to guide the optimization course, the DEoptim software [62] which is an R package implementing the differential evolution algorithm, the PDoublePop optimization software [63] that implements a parallel genetic algorithm for global optimization etc.

Also recently, some other optimization tools have appeared such as the Paradiseo [64] implemented in C++, which mainly includes evolutionary algorithms, the Pagmo software [65] where a wide range of evolution algorithms are incorporated to solve optimization problems, and finally another approach for evolutionary algorithms applied to optimization problems is the HeuristicLab freely available from https://dev.heuristiclab.com/trac.fcgi/, used mainly for online optimization. In the proposed software, the user can write the required objective function in simple C++ and then choose from a wide range of global optimization methods, the most suitable one for finding the global minimum. Furthermore, in the proposed software, the user can parameterize the local minimization method to be used as well as the termination method to be used for the successful termination of the technique. In addition, it is possible for the user to create his own global minimization method from scratch using the programming tools of the Optimus libraries.

The rest of this article is structured as follows: in section 2 the proposed software is outlined in detail, in section 3 some experiments are conducted to show the effectiveness of the proposed software and finally in section 4 some conclusions and guidelines for future work are presented.

2. Software

The suggested software is entirely coded in ANSI C++, using the freely available QT programming library, which can be downloaded from https://qt.io (accessed on 8 February 2023). The researcher should code the objective function and a number of other mandatory functions in the C++ programming language. Also, the researcher should provide the dimension of the objective function as well as the bound of the function (equation 1). Subsequently, the user can select a global optimization method to apply to the problem

from a wide range of available methods. Also, the user can extend the series of methods by adding any new method that follows the guidelines of the software. In the following subsections, the installation process of the suggested software will be analyzed and a complete example of running an objective problem will be given.

2.1. Installation

The software can be installed in almost any operating system running a C++ compiler and the freely available library of QT. The steps to install the software are similar to most operating systems and have as follows:

1. Download and install the QT programming library from https://qt.io.
2. Download and unzip the software from https://github.com/itsoulos/GlobalOptimus.
3. Issue the command: *cd GlobalOptimus-master*
4. Execute the command *qmake* (or *qmake-qt5* in some installations).
5. Execute the command *make*

The compilation will take some minutes and the final outcome of this compilation will be the executable *GlobalOptimus*.

2.2. Implemented global optimization methods

In the following, the global optimization methods present in the proposed software are presented. In most of them, a local optimization method is applied after their end in order to find the global minimum with greater reliability. In the proposed software, each implemented global optimization method has a set of parameters that can determine the global optimization path and the effectiveness of the method. For example, the genetic algorithm contains parameters such as the number of chromosomes or the maximum number of generations allowed. In addition, to make the optimization process easier, each method has been assigned a symbolic name, such as pso for particle swarm optimization. The implemented global optimization methods are:

1. Differential Evolution. The differential evolution method is included in the software as suggested by Storn[66] and denoted as **DifferentialEvolution**. This global optimization technique has been widely used in areas such as data mining applications [67,68], material design problems [69], feature selection [70], clustering methods [71] etc.
2. Parallel Differential Evolution. A parallel implementation of the Differential Evolution method as suggested in [72] is considered with the name **ParallelDe**. This parallel technique divides the total work into a number of available parallel computing units, and in each unit an independent Differential Evolution method is executed. The parallelization is done using the OpenMP programming library [73].
3. Double precision genetic algorithm. A modified genetic algorithm [74] is included in the software and it is denoted as **Genetic**. Genetic algorithms are typical representatives of evolutionary techniques with many applications such as scheduling problems [75], the vehicle routing problem [76], combinatorial optimization [77], architectural design etc [78].
4. Improved Particle Swarm Optimization. The improved Particle Swarm method as suggested by Charilogis and Tsoulos [82]. The particle swarm optimization method was applied successfully to a vast number of problems such as parameter extraction of solar cells [79], crystal structure prediction [80], molecular simulations [81] etc. The implemented method is denoted as **iPso**. The original Particle Swarm Optimization method is enhanced using a new inertia calculation mechanism as well as a novel termination method.
5. Multistart. A simple method that initiates local searches from different initial points is also implemented in the software. Despite its simplicity, the multistart method has been applied to many problems, such as the TSP problem [83], the vehicle routing problem [84], the facility location problem [85], the maximum clique problem [86], the maximum fire risk insured capital problem [87], aerodynamic problems [88] etc

6. NeuralMinimizer. A novel method that incorporates Radial Basis Functions (RBF)[89] to create an estimation of the objective function introduced in [90] is implemented and denoted by the name **NeuralMinimizer**.
7. Parallel Particle Swarm optimizer. A new method proposed in [91], that utilizes the OpenMP library to develop a parallel PSO variant. The method is denoted as **ParallelPso** in the Optimus package.
8. Simulated annealing optimizer. A Simulated Annealing optimizer as proposed by Corana et al [92] is included in the software under the name **Simman**.

2.3. Implemented local optimization methods

All global optimization methods can be enhanced by applying a local minimization method after they are terminated. The parameter used to determine the used local optimization procedure is the −−opt_localsearch parameter. The implemented local optimization methods are the following:

1. The **bfgs** method. The Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm was implemented using a variant of Powell [93].
2. The **lbfgs** method. The limited memory BFGS method [94] is implemented as an approximation of the BFGS method using a limited amount of computer memory. This local search procedure is ideal for objective functions of higher dimensions.
3. The Gradient descent method. This method is denoted as **gradient** in the software and implements the Gradient Descent local optimization procedure. This local search procedure is used in various problems such as neural network training [95], image registration [96] etc.
4. The Nelder Mead method. The Nelder - Mead simplex procedure for local optimization [97] is also included in the software and it is denoted as **nelderMead**.
5. The **adam** method. The adam local optimizer [98] is implemented also.

2.4. Implementing a user - defined optimization method

The software can be extended by implementing optimization techniques by the user himself. For this purpose there is the optimization method named UserMethod and the user can implement the provided functions according to the requirements of the method. The header file of this method is outlined in Algorithm 1. The functions in the class *UserMethod* have the following meaning:

1. UserMethod(). This is the constructor of the class. Critical parameters of the optimization method can also take place in it, as shown in the following code.

```
UserMethod::UserMethod()
{
        addParam(Parameter("userParam","1",
                "Example␣user␣parameter"));
}
```

   The method addParam() adds a new parameter to the command line program GlobalOptimus, that can be used with the notation −−userParam=value.
2. init(). This function is called every time the optimization method starts and will only be executed once before the optimization method steps. In it the user can initialize method parameters, create arrays or even generate samples from the objective function.
3. step(). This function implements the actual step of the optimization method.
4. terminated(). This function is used as the termination step of the optimization method. It returns true when the method should terminate and false otherwise.
5. done(). This function will be called when the optimization method terminates. At this point in the code, the *localSearch* function can be called in order to drive the

optimization method with greater certainty to some local minimum of the objective function.

6. ~UserMethod(). This is the destructor of the optimization method.

A flowchart of any used optimization method is outlined in Figure 1.

Algorithm 1 User defined optimization method (header file).

```
#ifndef USERMETHOD_H
#define USERMETHOD_H
# include <OPTIMUS/optimizer.h>

class UserMethod :public Optimizer
{
private:
    int paramValue;
public:
    UserMethod();
    virtual void init();
    virtual void step();
    virtual bool terminated();
    virtual void done();
    ~UserMethod();
};
#endif // USERMETHOD_H
```

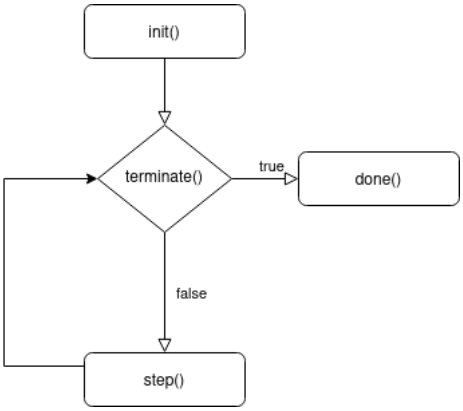

Figure 1. The flowchart of the execution steps of the optimization methods.

2.5. Objective problem deployment

The objective problem must be coded in the C++ programming language. The programmer must describe in detail the problem to be solved and must provide the software with detailed information about the dimension of the problem, the value limits of the variables of the problem, the objective function and also the derivative of the function. If the analytical derivative is not available or difficult to calculate, then the programmer can program it using finite differences or use some automatic differentiation software, such as the Adept software [99]. In the existing distribution for convenience, all objective problems are in the folder PROBLEMS.

2.5.1. Objective function coding

Figure 2 shows an example of objective function. The figure show also the required functions by the proposed software. This code is used for the minimization of the Rastrigin function defined as:

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

with $x \in [-1,1]^2$. In all methods the user defined type

```
typedef vector<double> Data;
```

Is used to define vectors of double precision numbers. The methods of the class RastriginProblem shown in the figure 2 have the following meaning:

1. The constructor method RastriginProblem, used to initialize the dimension of the problem and the corresponding bounds with the methods setLeftMargin() and setRightMargin().
2. **double** funmin(Data &x). This function returns the objective problem $f(x)$ for a given point x.
3. **Data** gradient(Data &x). This functions returns the gradient $\nabla f(x)$ for a given point x.

Figure 2. A typical representation of an objective problem, suitable for the OPTIMUS programming tool.

```
#include "rastriginproblem.h"
RastriginProblem::RastriginProblem(): Problem(2)
{
        Data l, r;
        l.resize(2);
        r.resize(2);
        for (int i = 0; i < 2; i++)
        {
                l[i] = -1.0;
                r[i] = 1.0;
        }
        setLeftMargin(l);
        setRightMargin(r);
}
double RastriginProblem::funmin(Data &x)
{
  return x[0] * x[0] + x[1] * x[1] - cos(18.0 * x[0]) - cos(18.0 * x[1])
}
Data RastriginProblem::gradient(Data &x)
{
  Data g;
  g.resize(2);
  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]);
  return g;
}
```

2.5.2. User defined problem

For convenience all objective problems have been stored in the PROBLEMS folder of the existing distribution, although the programmer can easily create his own objective function simply by overriding the class Problem. The user can also implement the methods of class UserProblem found in PROBLEMS subdirectory with contents shown in Figure 3. The class has two additional methods that may be used by the user:

1. void init(QJsonObject &params). The function init() is called before the objective function is executed and its purpose is to pass parameters from the execution environment to the objective function.
2. QJsonObject done(Data &x). This function is executed after the objective function optimization process is completed. The point x is the global minimum for the function $f(x)$.

Figure 3. The user defined problem UserProblem.

```
#include "userproblem.h"
# include <stdio.h>
UserProblem::UserProblem() :Problem(1)
{
}
double  UserProblem::funmin(Data &x)
{
        printf("This is a simple test function.\n");
        return 0.0;
}
Data UserProblem::gradient(Data &x)
{
        Data g;
        g.resize(x.size());
        return g;
}
void    UserProblem::init(QJsonObject &params)
{
}
QJsonObject UserProblem::done(Data &x)
{
}
UserProblem::~UserProblem() {
}
```

2.5.3. Objective function execution

A full working command for the Rastrigin problem using the utility program *GlobalOptimus* is shown below

```
./GlobalOptimus --opt_problem=rastrigin --opt_method=Genetic --
    opt_iters=1 --opt_localsearch=bfgs --gen_lrate=0.05
```

The parameters for the above command line are as follows:

1. The argument of the option −−opt_problem determines the objective problem. The objective problems are stored in the PROBLEMS subdirectory of the distribution.
2. The argument of the command line option −−opt_method sets the used global optimization procedure. For this case, the Genetic algorithm was used.
3. The argument of −−opt_iters determines the number of executions of the global optimization method.
4. The argument of −−gen_lrate determines the frequency of application of the local minimization method to the chromosomes of the genetic algorithm.
5. The argument −−opt_localsearch sets the used local optimization procedure.

The output of the previous command is shown in figure 4. As it is obvious, the global optimization method is quite close to the global minimum of the function, which is -2. However, with the help of the local optimization method applied after its end, this minimum is found with greater numerical accuracy. A number of shell scripts are also available in the existing distribution to simplify the task of running global optimization algorithms, such as the script *runfunmin.sh* for UNIX systems or the *runfunmin.bat* for Windows systems.

Figure 4. Output for the minimization of the Rastrigin function using the Genetic optimizer.

```
GENETIC. GENERATION=     1 BEST VALUE=          -1.845998656
GENETIC. GENERATION=     2 BEST VALUE=          -1.9458481
GENETIC. GENERATION=     3 BEST VALUE=          -1.9458481
GENETIC. GENERATION=     4 BEST VALUE=          -1.9458481
GENETIC. GENERATION=     5 BEST VALUE=          -1.9458481
GENETIC. GENERATION=     6 BEST VALUE=          -1.9458481
GENETIC. GENERATION=     7 BEST VALUE=          -1.9458481
GENETIC. terminate: -2.000000
Executions:      1 ==================
RUN:      1 BEST VALUE:                  -2
FUNCTION CALLS: 3459
Average Function calls:              3459.00
Minimum Function Value:                  -2
Percent Minimum Found:                100.00%
```

3. Experiments

To assess the ability of the software package to adapt to different problems, a series of experiments were performed under different conditions. In the first series of experiments, different global optimization techniques were applied to a series of objective functions that one can locate in the relevant literature. In the second series of experiments, the proposed software was applied to a difficult problem from the field of chemistry, that of finding the minimum potential energy of N interacting atoms of molecules. In the third set of experiments, the scaling of the required number of function calls was evaluated with a parallel technique applied to a difficult problem from the global optimization space, where the problem dimension was constantly increasing. In the fourth set of experiments, the genetic algorithm was utilized for the interface Fuchs-Kliewer polaritons. In the final set of experiments, different sampling techniques were incorporated for the used genetic algorithm.

3.1. Test functions

Some of the proposed methods are tested on a series of well - known test problems from the relevant literature. These problems are used by many researchers in the field. The description of the test functions has as follows:

- **Exponential** function, defined as:

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

  The values $n = 32, 64$ were used in the executed experiments.
- **Griewank2** function. This objective function is defined as:

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

- **Griewank10** function. The function is given by the equation

$$f(x) = \sum_{i=1}^{n}\frac{x_i^2}{4000} - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

  with $n = 10$.
- **Rastrigin** function. The function is provided by

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

- **Shekel 7** function.

$$f(x) = -\sum_{i=1}^{7} \frac{1}{(x-a_i)(x-a_i)^T + c_i}$$

with $x \in [0,10]^4$ and $a = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 3 & 5 & 3 \end{pmatrix}$, $c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \end{pmatrix}$

- **Shekel 5** function.

$$f(x) = -\sum_{i=1}^{5} \frac{1}{(x-a_i)(x-a_i)^T + c_i}$$

with $x \in [0,10]^4$ and $a = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \end{pmatrix}$, $c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \end{pmatrix}$.

- **Shekel 10** function.

$$f(x) = -\sum_{i=1}^{10} \frac{1}{(x-a_i)(x-a_i)^T + c_i}$$

with $x \in [0,10]^4$ and $a = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix}$, $c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.6 \end{pmatrix}$

- **Test2N** function. This function is given by the equation

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

This objective function has 2^n local minima in the specified range. During the conducted experiments the values $n = 4,5,6,7$ were used.

The experiments were performed using the above objective functions and ran 30 times using a different seed for the random number generator each time. During the execution of the experiments, the genetic algorithm (Genetic method) was used as a global optimizer in two versions: one without a local optimization method and one with periodic application of the bfgs method at a rate of 5% on the chromosomes in every generation. The execution parameters for the genetic algorithm are listed in Table 1. The experimental results for the two variants of the genetic algorithm are listed in Table 2. The numbers in cells denote average function calls for the 30 independent runs. The numbers in parentheses show the percentage of finding the global minimum in the 30 runs. If this number is absent, it means that the algorithm discovered the global minimum in all 30 executions. In this table, the line SUM represents the sum of the function calls. The experimental results indicate that the usage of a local search method in combination with the genetic algorithm

Table 1. Experimental settings

PARAMETER	VALUE
CHROMOSOMES	200
CROSSOVER RATE	90%
MUTATION RATE	5%
GENERATIONS	200
LOCAL SEARCH METHOD	bfgs

Table 2. Experimental results for some test functions using a series of global optimization methods.

FUNCTION	GENETIC	GENETIC WITH LOCAL
GRIEWANK2	3610	4575
GRIEWANK10	12604(0.07)	6542
EXP32	16743	3564
EXP64	11254	3883
RASTRIGIN	3334	4358
SHEKEl5	5791(0.60)	3343
SHEKEL7	5425(0.73)	3370
SHEKEL10	5533(0.73)	3496
TEST2N4	4953	3345
TEST2N5	6041	3805
TEST2N6	7042(0.90)	4299
TEST2N7	7895(0.90)	4969(0.97)
SUM	**90225(0.83)**	**49549(0.99)**

significantly reduces the required number of average function calls and also improves the reliability of the method in finding the global minimum. Of course, periodically applying a local minimization method to some of the chromosomes drastically increases the required execution time, but the large reduction in the total number of calls required is a big advantage of its application.

Also, a statistical comparion using boxplots for these results is shown in Figure 5.

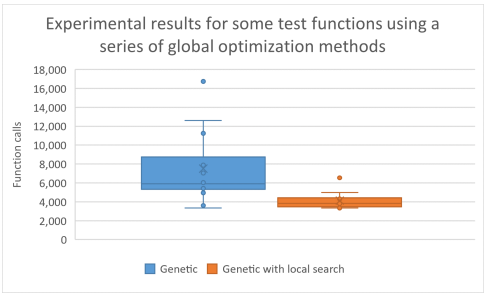

Figure 5. Comparison using boxplots for the results with the genetic algorithm method.

3.2. The Lennard Jones potential

The molecular conformation corresponding to the global minimum of the energy of N atoms interacting via the Lennard-Jones potential [107,109] is used as a test case here. The function to be minimized is given by:

$$V_{LJ}(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{2}$$

For testing purposes, the method iPSO of the package was applied to the above problem for a variety of number of atoms and the results are shown in Table 3. This method was experimentally compared with the genetic algorithm. In all cases, the number of chromosomes (or particles) was set to 200 and the maximum number of allowed iterations

was set to 200. As can be seen from the experimental results, the method iPSO requires a significantly reduced number of function calls compared to genetic algorithm, while its reliability in finding the global minimum for the potential remains high even when the number of atoms participating in the potential increases significantly.

Table 3. Optimizing the Potential problem for different numbers of atoms.

ATOMS	GENETIC(lrate=5%)	iPSO
3	3637	6814
4	5492	9484
5	6501	11542
6	16826(0.67)	12956(0.80)
7	8802	14584
8	9642	17259
9	20331	18052
10	41588	18384
11	34326(0.97)	18655
12	41278(0.97)	20258(0.97)
13	55367(0.90)	21185(0.97)
14	18314	23016
15	39088	24037
AVERAGE	**301192(0.96)**	**216626(0.98)**

The experimental results for the potential problem are also graphically demonstrated in Figure 6.

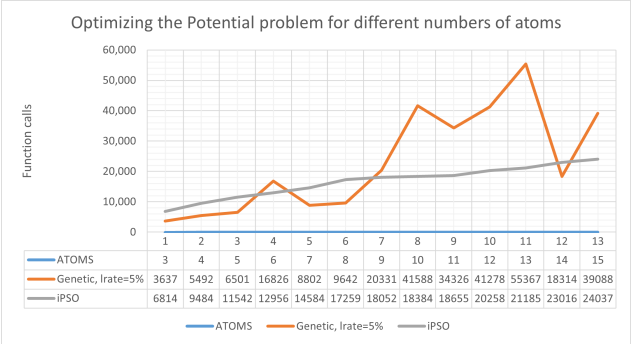

Figure 6. Graphical representation of the experimental results for the Potential problem.

3.3. Parallel optimization

The High Conditioned Elliptic function, defined as

$$f(x) = \sum_{i=1}^{n} \left(10^6\right)^{\frac{i-1}{n-1}} x_i^2$$

is used as a test case to measure the scalability of the parallel global optimization technique denoted as ParallelDe. This method was applied to the problem with dimension increasing from 2 to 20 and for a different number of processing threads. The experimental results are shown in diagram form in Figure 7. As one observes from the figure, the number of calls required to find the global minimum decreases as the total processing threads increase, although the problem becomes increasingly difficult with increasing dimension.

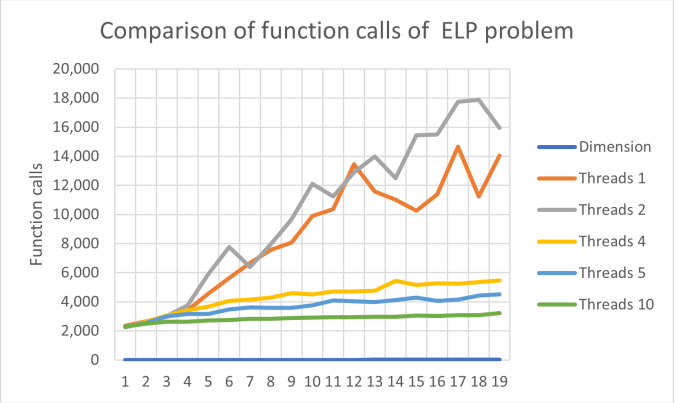

Figure 7. Scalabilty of the ParallelDe method.

3.4. The interface Fuchs-Kliewer polaritons

Let us consider a double heterostructure made with GaAs/AlAs. The dielectric functions to describe the interface Fuchs-Kliewer (FK) polaritons in the heterostructure are given by [108]

$$\epsilon_i = \epsilon_{\infty,i} \frac{\omega^2 - \omega_{L,i}^2}{\omega^2 - \omega_{T,i}^2} \tag{3}$$

where $\epsilon_{\infty,i}$ is the high-frequency dielectric constant, $\omega_{L,i}$ and $\omega_{L,i}$are the zone center LO and TO optical frequencies of the i-th material. The symmetric and the antisymmetric interface mode dispersion relations are respectively given by the following equations

$$\frac{\epsilon_2(\omega) q_1}{\epsilon_1(\omega) q_2} = -coth(\frac{q_2 d}{2}) \tag{4}$$

$$\frac{\epsilon_2(\omega) q_1}{\epsilon_1(\omega) q_2} = -tanh(\frac{q_2 d}{2}) \tag{5}$$

The wavevectors q_i and the in-plane wavevector $q_{||}$ are given by

$$q_i^2 = q_{||}^2 - \omega^2 \epsilon_i(\omega)/c^2 \tag{6}$$

In the figure 8 by using a genetic algorithm, we present the two lower interface polariton branches of the double heterostructure GaAs/AlAs by considering well width d=5 nm, $\hbar\,\omega_{L1}$=50.09 meV, $\hbar\,\omega_{T1}$=44.88 meV, $\hbar\,\omega_{L2}$=36.25 meV, $\hbar\,\omega_{T2}$=33.29 meV, $\epsilon_{\infty,1}$=8.16 and $\epsilon_{\infty,2}$=10.89. The results shown that the proposed algorithm is very competitive for the studied problem.

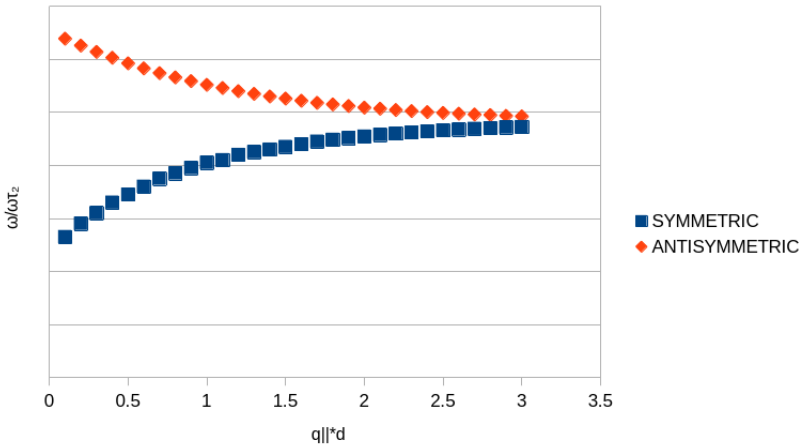

Figure 8. Using a genetic algorithm for the interface Fuchs-Kliewer polaritons

3.5. Experiments with the sampling method

One more parameter is available for the implemented global minimization methods of this software package with the name −−opt_sampler. With this parameter, the user can provide an alternative method used to draw samples from the objective problem. The default value is *uniform*, for the uniform distribution, although the user can use other distributions such as the triangular distribution [110] or use the K-Means [111] to draw samples. An experiment was performed using the Genetic optimizer and three sampling techniques: uniform, triangular and K-Means and the results are outlined in Table 4.

Table 4. Experiments with sampling methods using the Genetic optimizer.

PROBLEM	UNIFORM	TRIANGULAR	KMEANS
GRIEWANK2	4575	3960(0.97)	2273
GRIEWANK10	6542	6512	5250
EXP32	3564	3316	3471
EXP64	3883	3612	3789
RASTRIGIN	4358	3742	2474
SHEKEL5	3343	3066	2081
SHEKEL7	3370	3124	2084
SHEKEL10	3496	3175	2229
TEST2N4	3345	2968	2105
TEST2N5	3805	3597	2551
TEST2N6	4299	4036	2953
TEST2N7	4969(0.97)	4414(0.93)	3441(0.93)
SUM	**49549(0.99)**	**45522(0.99)**	**34701(0.99)**

As can be deduced from the results, the K-Means sampling improves the speed of the genetic algorithm by reducing the number of function calls required to obtain the global minimum of any given objective function in the experiment.

4. Conclusions

In this work, an environment for executing global optimization problems was presented. In this environment, the user can code the objective problem using some predefined functions and then has the possibility to choose one among several global optimization methods to solve the mentioned problem. In addition, it is given the possibility to choose to use some local optimization method to enhance the reliability of the produced results. This programming environment is freely available and easy to extend to accommodate more global optimization techniques. It is subject to continuous improvements and some of those planned for the near future are:

1. Use of modern parallel techniques to speed up the generated results and implementation of efficient termination techniques. In addition, new termination techniques specifically designed for parallel techniques should be devised and implemented.
2. Implementing a GUI interface to control the optimization process.
3. The ability to code the objective function in other programming languages such as Python, Ada, Fortran etc.
4. Creating a scripting language to efficiently guide the optimization of objective functions.

Author Contributions: I.G.T., V.C., A.T. and V.N.S. conceived the idea and methodology and supervised the technical part regarding the software. I.G.T. and V.C. conducted the experiments. A.T. performed the statistical analysis. V.N.S. and all other authors prepared the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Data Availability Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This research has been financed by the European Union : Next Generation EU through the Program Greece 2.0 National Recovery and Resilience Plan , under the call RESEARCH – CREATE – INNOVATE, project name “iCREW: Intelligent small craft simulator for advanced crew training using Virtual Reality techniques" (project code:TAEDK-06195)

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zwe-Lee Gaing, Particle swarm optimization to solving the economic dispatch considering the generator constraints, IEEE Transactions on **18** Power Systems, pp. 1187-1195, 2003.
2. C. D. Maranas, I. P. Androulakis, C. A. Floudas, A. J. Berger, J. M. Mulvey, Solving long-term financial planning problems via global optimization, Journal of Economic Dynamics and Control **21**, pp. 1405-1425, 1997.
3. Q. Duan, S. Sorooshian, V. Gupta, Effective and efficient global optimization for conceptual rainfall-runoff models, Water Resources Research **28**, pp. 1015-1031 , 1992.
4. P. Charbonneau, Genetic Algorithms in Astronomy and Astrophysics, Astrophysical Journal Supplement **101**, p. 309, 1995
5. A. Liwo, J. Lee, D.R. Ripoll, J. Pillardy, H. A. Scheraga, Protein structure prediction by global optimization of a potential energy function, Biophysics **96**, pp. 5482-5485, 1999.
6. P.M. Pardalos, D. Shalloway, G. Xue, Optimization methods for computing global minima of nonconvex potential energy functions, Journal of Global Optimization **4**, pp. 117-133, 1994.
7. Eva K. Lee, Large-Scale Optimization-Based Classification Models in Medicine and Biology, Annals of Biomedical Engineering **35**, pp 1095-1109, 2007.
8. Y. Cherruault, Global optimization in biology and medicine, Mathematical and Computer Modelling **20**, pp. 119-132, 1994.
9. Y. Gao, H. Rong, J.Z. Huang, Adaptive grid job scheduling with genetic algorithms, Future Generation Computer Systems **21**, pp. 151-161, 2005.
10. D.Y. Sha, H.H. Lin, A multi-objective PESO for job-shop scheduling problems, Expert Systems with Applications **37**, pp. 1065-1070, 2010.
11. X. Cai, D.C. McKinney, L.S. Lasdon, Solving nonlinear water management models using a combined genetic algorithm and linear programming approach, Advances in Water Resources **24**, pp. 667-676, 2001.
12. S.G. Gino Sophia, V. Ceronmani Sharmila, S. Suchitra et al, Water management using genetic algorithm-based machine learning, Soft Comput **24**, pp. 17153–17165, 2020.
13. Z. Bankovic, D. Stepanovic, S. Bojanic, O. Nieto - Taladriz, Improving network security using genetic algorithm approach, Computers & Electrical Engineering **33**, pp. 438-451, 2007.
14. S. Paul, I. Dutt, S.N. Choudhri, Design and implementation of network security using genetic algorithm. Int J Res Eng Technol **2**, pp. 172-177, 2013.
15. A. Tuncer, M. Yildrim, Dynamic path planning of mobile robots with improved genetic algorithm, Computers & Electrical Engineering **38**, pp. 1564-1572, 2012.
16. N. Kherici, Y.M. Ben Ali, Using PESO for a walk of a biped robot, Journal of Computational Science **5**, pp. 743-749, 2014.
17. B. Freisleben and P. Merz, A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems, In: Proceedings of IEEE International Conference on Evolutionary Computation, pp. 616-621, 1996.

18. R. Grbić, E.K. Nyarko and R. Scitovski, A modification of the DIRECT method for Lipschitz global optimization for a symmetric function, J Glob Optim **57**, pp. 1193–1212, 2013.
19. R. Scitovski, A new global optimization method for a symmetric Lipschitz continuous function and the application to searching for a globally optimal partition of a one-dimensional set, J Glob Optim **68**, pp. 713–727, 2017.
20. Barbara Kaltenbacher and William Rundell, The inverse problem of reconstructing reaction–diffusion systems, Invese Problems **36**, 2020.
21. N. Levashova, A. Gorbachev, R. Argun, D. Lukyanenko, The Problem of the Non-Uniqueness of the Solution to the Inverse Problem of Recovering the Symmetric States of a Bistable Medium with Data on the Position of an Autowave Front., Symmetry **13**, 2021.
22. Larisa Beilina, Michael V. Klibanov, A Globally Convergent Numerical Method for a Coefficient Inverse Problem, SIAM Journal on Scientific Computing **31**,pp. 478-509, 2008.
23. M. Brunato, R. Battiti, RASH: A Self-adaptive Random Search Method. In: Cotta, C., Sevaux, M., Sörensen, K. (eds) Adaptive and Multilevel Metaheuristics. Studies in Computational Intelligence, vol 136. Springer, Berlin, Heidelberg, 2008.
24. S. Andradóttir, A.A. Prudius, A.A., Adaptive random search for continuous simulation optimization. Naval Research Logistics **57**, pp. 583-604, 2010.
25. W.L. Price, Global optimization by controlled random search, J Optim Theory Appl **40**, pp. 333–348, 1983.
26. P. Kaelo, M.M. Ali, Some Variants of the Controlled Random Search Algorithm for Global Optimization. J Optim Theory Appl **130**, pp. 253–264 (2006).
27. S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science **220**, pp. 671-680, 1983.
28. K.M.El-Naggar, M.R. AlRashidi, M.F. AlHajri, A.K. Al-Othman, Simulated Annealing algorithm for photovoltaic parameters identification, Solar Energy **86**, pp. 266-274, 2012.
29. L.M. Rasdi Rere, M.I. Fanany, A.M. Arymurthy, Simulated Annealing Algorithm for Deep Learning, Procedia Computer Science **72**, pp. 137-144, 2015.
30. J. Mc Call, Genetic algorithms for modelling and optimisation, Journal of Computational and Applied Mathematics **184**, pp. 205-222, 2005.
31. C.K.H. Lee, A review of applications of genetic algorithms in operations management, Elsevier Engineering Applications of Artificial Intelligence **76**, pp. 1-12, 2018.
32. B. Chandra Mohan, R. Baskaran, A survey: Ant Colony Optimization based recent research and implementation on several engineering domain, Expert Systems with Applications **39**, pp. 4618-4627, 2012.
33. T. Liao, T. Stützle, M.A. Montes de Oca, M. Dorigo, A unified ant colony optimization algorithm for continuous optimization, European Journal of Operational Research **234**, pp. 597-609, 2014.
34. D. Wang, D. Tan, L. Liu, Particle swarm optimization algorithm: an overview. Soft Comput **22**, pp. 387–408, 2018.
35. N.K. Jain, U. Nangia, J. Jain, A Review of Particle Swarm Optimization. J. Inst. Eng. India Ser. B **99**, pp. 407–411, 2018.
36. D.H. Kim, A. Abraham, J.H. Cho, A hybrid genetic algorithm and bacterial foraging approach for global optimization, Information Sciences **177**, pp. 3918-3937, 2007.
37. Y.T. Kao, E. Zahara, A hybrid genetic algorithm and particle swarm optimization for multimodal functions, Applied Soft Computing **8**, pp. 849-857, 2008.
38. G.T. Reddy, M.P.K. Reddy, K. Lakshmanna et al, Hybrid genetic algorithm and a fuzzy logic classifier for heart disease diagnosis, Evol. Intel. **13**, pp. 185–196, 2020.
39. M.D. Anisur Rahman, M.D. Zahidul Islam, A hybrid clustering technique combining a novel genetic algorithm with K-Means, Knowledge-Based Systems **71**, pp. 345-365, 2014.
40. T. Niknam, An efficient hybrid evolutionary algorithm based on PESO and ACO for distribution feeder reconfiguration, European Transactions on Electrical Power **20**, pp. 575-590, 2010.
41. M.K. Patel, M.R. Kabat, C.R. Tripathy, A hybrid ACO/PESO based algorithm for QoS multicast routing problem, Ain Shams Engineering Journal **5**, pp. 113-120, 2014.
42. A.K. Dubey, A. Kumar, R. Agrawal, An efficient ACO-PSO-based framework for data classification and preprocessing in big data, Evol. Intel. **14**, pp. 909–922, 2021.
43. Offord C., Bajzer Ž. (2001) A Hybrid Global Optimization Algorithm Involving Simplex and Inductive Search. In: Alexandrov V.N., Dongarra J.J., Juliano B.A., Renner R.S., Tan C.J.K. (eds) Computational Science - ICCS 2001. ICCS 2001. Lecture Notes in Computer Science, vol 2074. Springer, Berlin, Heidelberg.
44. S. Li, M. Tan, I. W. Tsang, J. T. -Y. Kwok, A Hybrid PSO-BFGS Strategy for Global Optimization of Multimodal Functions, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **41**, pp. 1003-1014, 2011.
45. H. Badem, A. Basturk, A. Caliskan, M.E. Yuksel, A new hybrid optimization method combining artificial bee colony and limited-memory BFGS algorithms for efficient numerical optimization, Applied Soft Computing **70**, pp. 826-844, 2018.
46. A.A. Nagra, F. Han, Q.H. Ling, An improved hybrid self-inertia weight adaptive particle swarm optimization algorithm with local search, Engineering Optimization **51**, pp. 1115-1132, 2018.
47. S.Q. Wu, M. Ji, C.Z. Wang, M.C. Nguyen, X. Zhao, K. Umemoto, R. M. Wentzcovitch, K. M. Ho, An adaptive genetic algorithm for crystal structure prediction, Journal of Physics: Condensed Matter **26**, 035402, 2013.

48. M. Honda, Application of genetic algorithms to modelings of fusion plasma physics, Computer Physics Communications **231**, pp. 94-106, 2018.
49. X.L. Luo, J. Feng, H.H. Zhang, A genetic algorithm for astroparticle physics studies, Computer Physics Communications **250**, 106818, 2020.
50. M. Ye, X. Wang, Y. Xu, Parameter extraction of solar cells using particle swarm optimization, Journal of Applied Physics **105**, 094502. 2009.
51. A. Belkadi, L. Ciarletta, D. Theilliol, Particle swarm optimization method for the control of a fleet of Unmanned Aerial Vehicles, Journal of Physics: Conference Series, Volume 659, 12th European Workshop on Advanced Control and Diagnosis (ACD 2015) 19–20 November 2015, Pilsen, Czech Republic.
52. M. Depolli, R. Trobec, B. Filipič, Asynchronous Master-Slave Parallelization of Differential Evolution for Multi-Objective Optimization, Evolutionary Computation **21**, pp. 261-291, 2013.
53. A. P. Engelbrecht, Asynchronous particle swarm optimization with discrete crossover, In: 2014 IEEE Symposium on Swarm Intelligence, Orlando, FL, USA, 2014, pp. 1-8.
54. F. Bourennani, Cooperative asynchronous parallel particle swarm optimization for large dimensional problems, International Journal of Applied Metaheuristic Computing (IJAMC) **10.3**, pp. 19-38, 2019.
55. J. Larson and S.M. Wild, Asynchronously parallel optimization solver for finding multiple minima, Mathematical Programming Computation **10**, pp. 303-332, 2018.
56. H.P.J. Bolton, J.F. Schutte, A.A. Groenwold, Multiple Parallel Local Searches in Global Optimization. In: Dongarra J., Kacsuk P., Podhorszki N. (eds) Recent Advances in Parallel Virtual Machine and Message Passing Interface. EuroPVM/MPI 2000. Lecture Notes in Computer Science, vol 1908. Springer, Berlin, Heidelberg, 2000.
57. Y. Zhou and Y. Tan, GPU-based parallel particle swarm optimization, In: 2009 IEEE Congress on Evolutionary Computation, 2009, pp. 1493-1500.
58. L. Dawson and I. Stewart, Improving Ant Colony Optimization performance on the GPU using CUDA, In: 2013 IEEE Congress on Evolutionary Computation, 2013, pp. 1901-1908.
59. Barkalov, K., Gergel, V. Parallel global optimization on GPU. J Glob Optim **66**, pp. 3–20, 2016.
60. N.V. Sahinidis, BARON: A general purpose global optimization software package, J Glob Optim **8**, pp. 201–205, 1996.
61. D.G. Papageorgiou, I.N. Demetropoulos, I.E. Lagaris, Computer Physics Communications **159**, pp. 70-71, 2004.
62. K. Mullen, D. Ardia, D.L. Gil, D. Windover, J. Cline, DEoptim: An R Package for Global Optimization by Differential Evolution, Journal of Statistical Software **40**, pp. 1-26, 2011.
63. I.G. Tsoulos, A. Tzallas, D. Tsalikakis, PDoublePop: An implementation of parallel genetic algorithm for function optimization, Computer Physics Communications **209**, pp. 183-189, 2016.
64. J. Dreo, A. Liefooghe, S. Verel, M. Schoenauer, J.J. Merelo, A. Quemy, B. Bouvier, J. Gmys, Paradiseo: from a modular framework for evolutionary computation to the automated design of metaheuristics —22 years of Paradiseo—, GECCO'21: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 1522–1530, 2021.
65. F. Biscani, D. Izzo, A parallel global multiobjective framework for optimization: pagmo, Journal of Open Source Software **5**, 2338, 2020.
66. R. Storn, On the usage of differential evolution for function optimization, In: Proceedings of North American Fuzzy Information Processing, pp. 519-523, 1996.
67. I. Triguero, S. Garcia, F. Herrera, Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification, Pattern Recognition **44**, pp. 901-916, 2011.
68. Y.H. Li, J.Q. Wang, X.J. Wang, Y.L. Zhao, X.H. Lu, D.L. Liu, Community Detection Based on Differential Evolution Using Social Spider Optimization, Symmetry **9**, 2017.
69. W. Yang, E.M. Dilanga Siriwardane, R. Dong, Y. Li, J. Hu, Crystal structure prediction of materials with high symmetry using differential evolution, J. Phys.: Condens. Matter **33** 455902, 2021.
70. C.Y. Lee, C.H. Hung, Feature Ranking and Differential Evolution for Feature Selection in Brushless DC Motor Fault Diagnosis , Symmetry **13**, 2021.
71. S. Saha, R. Das, Exploring differential evolution and particle swarm optimization to develop some symmetry-based automatic clustering techniques: application to gene clustering, Neural Comput & Applic **30**, pp. 735–757, 2018.
72. V. Charilogis, I.G. Tsoulos, A Parallel Implementation of the Differential Evolution Method, Analytics **2**, pp. 17-30, 2023.
73. R. Chandra, L. Dagum, D. Kohr, D. Maydan,J. McDonald and R. Menon, Parallel Programming in OpenMP, Morgan Kaufmann Publishers Inc., 2001.
74. I.G. Tsoulos, Modifications of real code genetic algorithm for global optimization, Applied Mathematics and Computation **203**, pp. 598-607, 2008.
75. J.F.Gonçalves, J.J.M. Mendes, M.G.C. Resende, A genetic algorithm for the resource constrained multi-project scheduling problem, European Journal of Operational Research **189**, pp. 1171-1190, 2008.
76. W.Ho, G.T.S. Ho, P. Ji, H.C.W. Lau, A hybrid genetic algorithm for the multi-depot vehicle routing problem, Engineering Applications of Artificial Intelligence **21**, pp. 548-557, 2008.
77. J.F. Gonçalves, M.G.C. Resende, Biased random-key genetic algorithms for combinatorial optimization. J Heuristics **17**, pp. 487–525, 2011.

78. M. Turrin, P. Buelow, R. Stouffs, Design explorations of performance driven geometry in architectural design using parametric modeling and genetic algorithms, Advanced Engineering Informatics **25**, pp. 656-675, 2011.
79. M. Ye, X. Wang, Y. Xu, Parameter extraction of solar cells using particle swarm optimization, Journal of Applied Physics **105**, 094502, 2009.
80. Y. Wang, J. Lv, L. Zhu, Y. Ma, Crystal structure prediction via particle-swarm optimization, Phys. Rev. B **82**, 094116, 2010.
81. M. Weiel, M. Götz, A. Klein et al, Dynamic particle swarm optimization of biomolecular simulation parameters with flexible objective functions. Nat Mach Intell **3**, pp. 727–734, 2021.
82. V. Charilogis, I.G. Tsoulos, Toward an Ideal Particle Swarm Optimizer for Multidimensional Functions, Information **13**, 217, 2022.
83. Li W., A Parallel Multi-start Search Algorithm for Dynamic Traveling Salesman Problem. In: Pardalos P.M., Rebennack S. (eds) Experimental Algorithms. SEA 2011. Lecture Notes in Computer Science, vol 6630. Springer, Berlin, Heidelberg, 2011.
84. Olli Bräysy, Geir Hasle, Wout Dullaert, A multi-start local search algorithm for the vehicle routing problem with time windows, European Journal of Operational Research **159**, pp. 586-605, 2004.
85. Mauricio G.C. Resende, Renato F. Werneck,A hybrid multistart heuristic for the uncapacitated facility location problem, European Journal of Operational Research **174**, pp. 54-68, 2006.
86. E. Marchiori, Genetic, Iterated and Multistart Local Search for the Maximum Clique Problem. In: Cagnoni S., Gottlieb J., Hart E., Middendorf M., Raidl G.R. (eds) Applications of Evolutionary Computing. EvoWorkshops 2002. Lecture Notes in Computer Science, vol 2279. Springer, Berlin, Heidelberg.
87. Gomes M.I., Afonso L.B., Chibeles-Martins N., Fradinho J.M. (2018) Multi-start Local Search Procedure for the Maximum Fire Risk Insured Capital Problem. In: Lee J., Rinaldi G., Mahjoub A. (eds) Combinatorial Optimization. ISCO 2018. Lecture Notes in Computer Science, vol 10856. Springer, Cham. https://doi.org/10.1007/978-3-319-96151-4_19
88. Streuber, Gregg M. and Zingg, David. W., Evaluating the Risk of Local Optima in Aerodynamic Shape Optimization, AIAA Journal 59, pp. 75-87, 2012.
89. J. Park, I.W. Sandberg, Approximation and Radial-Basis-Function Networks, Neural Computation **5**, pp. 305-316, 1993.
90. I.G. Tsoulos, A. Tzallas, E. Karvounis, D. Tsalikakis, NeuralMinimizer, a novel method for global optimization that incorporates machine learning, Information **14**, 2, 2023.
91. V. Charilogis, I.G. Tsoulos, A. Tzallas, An Improved Parallel Particle Swarm Optimization, SN COMPUT. SCI. **4**, 766, 2023.
92. A. Corana, M. Marchesi, C. Martini, S. Ridella, Minimizing multimodal functions of continuous variables with the "Simulated Annealing" algo- rithm, ACM Trans. Math. Software **13**, pp. 262–280, 1987.
93. M.J.D Powell, A Tolerant Algorithm for Linearly Constrained Optimization Calculations, Mathematical Programming **45**, pp. 547-566, 1989.
94. D.C. Liu, J. Nocedal, On the Limited Memory Method for Large Scale Optimization, Mathematical Programming B **45**, pp. 503-528, 1989.
95. S.I. Amari, Backpropagation and stochastic gradient descent method, Neurocomputing **5**, pp. 185-196, 1993.
96. S. Klein, J.P.W. Pluim, M. Staring, Adaptive Stochastic Gradient Descent Optimisation for Image Registration, Int J Comput Vis **81**, pp. 227–239, 2009.
97. D.M. Olsson,L.S. Nelson, The Nelder-Mead Simplex Procedure for Function Minimization, Technometrics **17**, pp. 45-51, 1975.
98. D.P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization, ICLR (Poster), 2015.
99. R.J. Hogan, Fast reverse-mode automatic differentiation using expression templates in C++. ACM Trans. Math. Softw. **40**, pp. 1-26, 2014.
100. J.E. Lennard-Jones, On the Determination of Molecular Fields, Proc. R. Soc. Lond. A **106**, pp. 463–477, 1924.
101. C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.
102. G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control Signals and Systems **2**, pp. 303-314, 1989.
103. M.M. Ali and P. Kaelo, Improved particle swarm algorithms for global optimization, Applied Mathematics and Computation **196**, pp. 578-593, 2008.
104. H. Koyuncu, R. Ceylan, A PESO based approach: Scout particle swarm algorithm for continuous global optimization problems, Journal of Computational Design and Engineering **6**, pp. 129–142, 2019.
105. Patrick Siarry, Gérard Berthiau, François Durdin, Jacques Haussy, ACM Transactions on Mathematical Software **23**, pp 209–228, 1997.
106. I.G. Tsoulos, I.E. Lagaris, GenMin: An enhanced genetic algorithm for global optimization, Computer Physics Communications **178,** pp. 843-851, 2008.
107. J.A. Northby, Structure and binding of Lennard-Jones clusters: $13 \leq n \leq 147$, J. Chem. Phys. **87**, pp. 6166–6178, 1987.
108. B. K. Ridley, Electrons and Phonons in Semiconductor Multilayers, Cambridge University Press, 2nd edition (2014)
109. G.L. Xue, R.S. Maier, J.B. Rosen, Improvements on the Northby Algorithm for molecular conformation: Better solutions, J. Global. Optim. 4, pp. 425–440, 1994.
110. W.E. Stein, M.F. Keblis, A new method to simulate the triangular distribution, Mathematical and Computer Modelling Volume **49**, pp. 1143-1147, 2009.
111. M. Ahmed, R. Seraj, S.M.S. Islam, The k-means algorithm: A comprehensive survey and performance evaluation, Electronics **9**, 1295, 2020.