RcppGO

Peter Kehler*

July 12, 2013

Contents

1 Introduction

Different kinds of optimization problems require a dedicated algorithm. "[...], optimization algorithms are guided by objective functions. A function is difficult from a mathematical perspective in this context if it is not continuous, not differentiable, or if it has multiple maxima and minima."[25, p. 56]

This is a natural field of application for heuristic optimization algorithms. Figure 1 shows a classification approach metaheuristics. Many algorithms are inspired by nature. So is the *Charged System Search* or *CSS* described in [8]. The *CSS* is inspired by the Newtonian laws of mechanics and the Coulomb law from electrostatics. The authors demonstrated its performance using standard benchmark problems as well as engineering design problems, showing that *CSS* outperforms other established evolutionary optimization algorithms. Since its first publication the algorithm has been enhanced and successfully applied to many problems. The literature considering the *CSS* algorithm can be devided into its application and further development.

In [7] the authors developed a discrete version of the *CSS* algorithm and a constrained optimization approach was added.Further enhancements were made in [9] by utilizing the *fields of forces* method resulting in an *enhanced CSS* outperforming the original algorithm. To increase the global search mobility of the *CSS* algorithm the authors introduce several chaos based methods in [13] and named the version *chaos-based CSS* or *CCSS*.

As engineers by profession, the authors apply the *CSS* variants to several engineering problems including the optimum design of geodesic domes taking their nonlinear respones account in [10] or the optimal design of sceletal structures in [11].

The purpose of the `RcppGO` package is to provide an algorithm to tackle difficult otimization problems as defined before. The algorithm utilizes Newton's laws of gravity and motion and is loosely based on

*Email: peter.kehler.jr@googlemail.com

[8]. In the interest of brevity the user guide provides only the steps to get startet with the package. A detailed article on the algorithm is in preparation.

The rest of the user guide is organized as follows. Section 2 describes how to install the `RcppGO` package. Section 3 presents two examples of application and goes into the differences for uni- and multidimensional optimization problems.

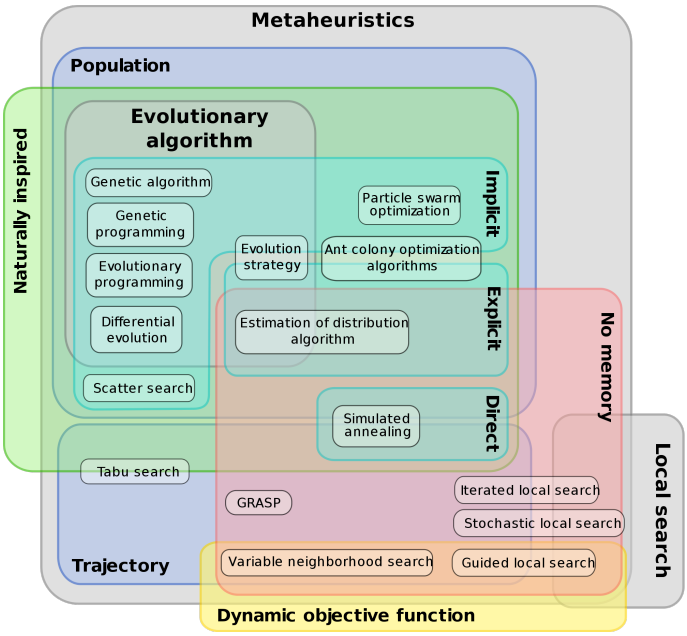

Figure 1: A classification of metaheuristic optimization algorithms [2].

2 Installation

The `RcppGO` package is available via the *Comprehensive R Archive Network* on CRAN. The developer version will be hosted on GitHub. In this section the installation of the `CRAN` and the `GitHub` versions are explained.

2.1 Requirements

`R` can be downloaded from the CRAN homepage. The installation of `R` is documented in [21]. `RcppGO` depends on the `R` packages `Rcpp` [5], `RcppArmadillo` [6, 22] and `lattice` [23].

Package	Version
`R`	≥ 2.15
`Rcpp`	$\geq 0.10.3$
`RcppArmadillo`	$\geq 0.3.900.0$
`lattice`	≥ 0.2

After installing `R` you can invoke the installation of the depending packages via the command:

```
1 install.packages(c("Rcpp", "RcppArmadillo", "lattice" ))
```

There are three other packages I suggest to install in order to compare the performance of `RcppGO` against other optimization algorithms. The suggestions include `DEoptim` [16], its `C++` implementation `RcppDE` [4] and `rbenchmark` [14]. Install these packages as shown before:

```
install.packages(c("DEoptim", "RcppDE", "rbenchmark"))
```

2.2 Installing the CRAN version

Installing RcppGO from CRAN is straight forward:

```
install.packages{"RcppGO"}
```

2.3 Installating the GitHub version

There are two options to install RcppGO on your maschine. You can either use the shell commands provided below or modify and use a prefabricated shell script. A prefrabricated self-executing[1] shell script can be found in the RcppGO package. The steps are:

(a) define Rcpp flags[2]

(b) export the defined Rcpp flags

(c) call the R command to install the RcppGO package

Execute the following commands in the shell. First open your shell and go to the folder where the RcppGO package is located, e.g. your download folder.

```
cd " ~/Downloads/"
```

The " $\sim$ " sign is an abbreviation for the home directory. Following the installation steps outlined above, enter the following commands in the shell window. The "#" sign indicates lines with comments.

```
# define flags
RCPP_CXXFLAGS=`Rscript -e 'Rcpp:::CxxFlags()'`
RCPP_LIBS=`Rscript -e 'Rcpp:::LdFlags()'`
RCPP_ARMADILLO=`Rscript -e 'RcppArmadillo:::CxxFlags()'`

# export flags
export PKG_CPPFLAGS="${RCPP_ARMADILLO} ${RCPP_CXXFLAGS}"
export PKG_LIBS="-larmadillo -llapack  ${RCPP_LIBS}"

## call the R install command
R CMD INSTALL RcppGO*
```

The procedure worked correct, if the output on your console looks like this:

[1] In order to execute the script the user may need to change the mode of the script. This can be done via a shell command provided in the script.

[2] See the [19, chapter 5] manual for further information.

```
** R
** preparing package for lazy loading
** help
*** installing help indices
** building package indices ...
** testing if installed package can be loaded
*** arch - i386
*** arch - x86_64

* DONE (RcppGO)
```

After installing the package it can be loaded into your R session.

```
library(RcppGO)
```

Loading the RcppGO package loads also the packages RcppGO dependens on. After loading the library into R the help page of the RcppGO package is available. Type the following command in R to open it:

```
help("RcppGO")
```

The help page gives a short description of the package, its functions and parameters. Use the demo command to get a simple introductory example.

```
demo("RcppGO")
```

3 Examples

The RcppGO package contains two main R functions. RcppGO() calls the optimization routine and the plot.RcppGO() visualizes results from two dimensional objective functions. In this section I give two introductory examples for the RcppGO() function and a short demo of the plot.RcppGO method. For convention the objective functions in the examples are minimized.

3.1 The RcppGO() function

There exist various standard mathematical benchmark functions. We will conduct the examples based on variants of the Aluffi-Pentiny function. For further standard benchmark functions see [27], [26] or [15]. Due to technical reasons RcppGO handles one- and multi-dimensional optimization in different notation. Here I want to outline an economic approach to solve optimization problems with RcppGO:

(a) Define the objective function,

(b) set the RcppGO() parameters,

(c) store the calculations in an R object,

(d) and process the results, e.g. print or plot them.

Before we proceed to the examples, let us have a brief overview of the RcppGO parameters. The RcppGO function is the core of the RcppGO package. She calls the optimization algorithm written in C++ and consists of the following arguments:

- `ObjectiveFunction`: The objective function to be optimized. Be aware of the matrix notation for multideimensional objective functions.
- `Args`: Defines the number of objective function arguments.
- `Lower, Upper`: Vectors specifying the lower and upper bounds of the optimization routine.
- `n`: The number of particles to be used in the optimization process. The default is 20.
- `g`: The number of solutions to be saved. `g` should be less or equal to `n`. The default value is 20.
- `Iterations`: Specifying the number of iterations after the algorithm terminates. Default is 200.
- `Scale`: A parameter for definig the attraction radius of the search particles. Default is 0.1.
- `User`: If the user wants to controll the `Scale` parameter manually, set `User=TRUE`. Default is `FALSE`.
- `Maximize`: Control parameter for maximizing (`Maximize=TRUE`) or minimizing (`Maximize=FALSE`) the objective function. Default is `FALSE`.

Some arguments have (arbitrary) default values which you can omit when you set up the optimization problem or change their values to your needs. The examples herafter will clarify the use of `RcppGO`.

3.1.1 A one dimensional optimization problem

The first demonstation represents a one dimensional Aluffi-Pentiny objective function, that is defined as:

$$f(x) = \frac{1}{4}x^4 + \frac{1}{10}x \tag{1}$$

The minimum is located at $x = \sqrt[3]{-\frac{1}{10}} \approx -0.46416$ with $f(x) \approx -0.03406$.

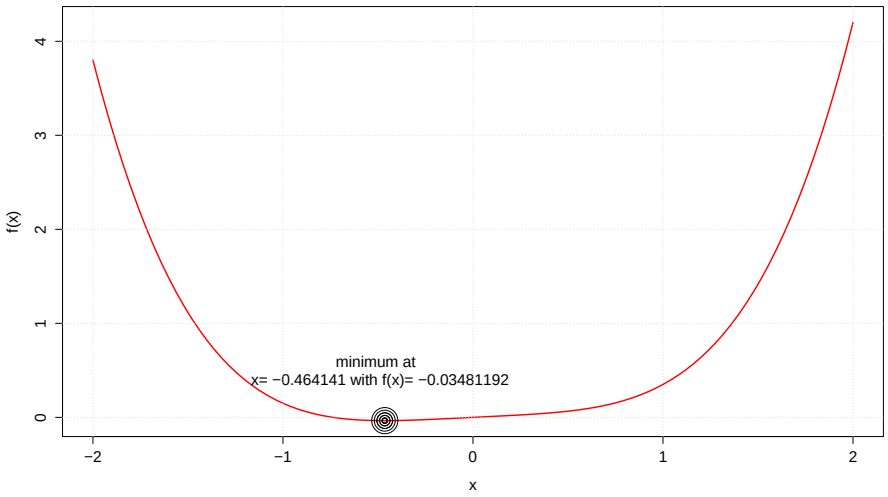

Figure 2: The Aluffi-Pentiny function

As outlined above we define the objective function in `R`:

```r
# Define f(x) = 1/4 x^4 + 1/10 x in R
AluffiPentiny01 <- function(X)
{
  1/4*X^4  + 1/10*X
}
```

We define the parameters, following the steps outlined above, omitting some parameters with default values:

```
1  Example01 <- RcppGO(                  # store calculations in an R object
2  ObjectiveFunction=AluffiPentiny01,    # passing the objective function
3  Args=1,                               # define the number of arguments
4  Lower = -10,                          # define lower limit of search space
5  Upper = 10                            # define upper limit of search space
6  )
```

Before we access the results, let us have a look at the structure of `Example01`. Typing `str(Example01)` into the `R` console reveals its composition:

```
1  >  str(Example01)                # structure of the results
2  List of 12
3   $ GravityParticles : num [1:20, 1:4, 1:200] -0.2349 -0.2246 0.0854 1.2793 -1.5062 ...
4    ..- attr(*, "dimnames")=List of 3
5    .. ..$ : chr [1:20] "1" "2" "3" "4" ...
6    .. ..$ : chr [1:4] "x1" "fn_x" "v_x1" "F_x1"
7    .. ..$ : NULL
8   $ ObjectiveFunction:function (X)
9    ..- attr(*, "srcref")=Class 'srcref'  atomic [1:8] 1 20 4 1 20 1 1 4
10   .. .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x20f7ae0>
11  $ GMemory          : num [1:20, 1:2] -0.464 -0.464 -0.464 -0.464 -0.464 ...
12   ..- attr(*, "dimnames")=List of 2
13   .. ..$ : chr [1:20] "1" "2" "3" "4" ...
14   .. ..$ : chr [1:2] "x1" "fn_x"
15  $ Iterations       : num 200
16  $ Args             : int 1
17  $ n                : int 20
18  $ g                : int 20
19  $ Lower            : num -10
20  $ Upper            : num 10
21  $ Scale            : num 0.1
22  $ User             : logi FALSE
23  $ Maximize         : logi FALSE
```

Line 2 of the above listing shows that `str(Example01)` is a `list` of 12 objects. The best found solutions of the optimization process are stored in `GMemory`. Typing `Example01` to the `R` console prints and expands all objects. Depending on your optimization problem the output might easily contain thousands of lines. To access any object of `Example01` you have to combine the listname followed by a $ sign and the object name, e.g.:

```
1  > Example01$GMemory            # print the results
2             x1        fn_x
3  1  -0.4641558 -0.03481192
4  2  -0.4641725 -0.03481192
5  3  -0.4641759 -0.03481192
6  4  -0.4641796 -0.03481192
7  5  -0.4641825 -0.03481192
8  6  -0.4641857 -0.03481192
9  7  -0.4641860 -0.03481192
10 8  -0.4641264 -0.03481192
11 9  -0.4641917 -0.03481192
12 10 -0.4641940 -0.03481192
```

The stochastic results match the theoretical results.

3.1.2 A multi-dimensional optimization problem

Let us now consider a two dimensional AluffiPentiny function.

$$f(x,y) = \frac{1}{4}x^4 - \frac{1}{2}x^2\frac{1}{10}x + \frac{1}{2}y^2 \tag{2}$$

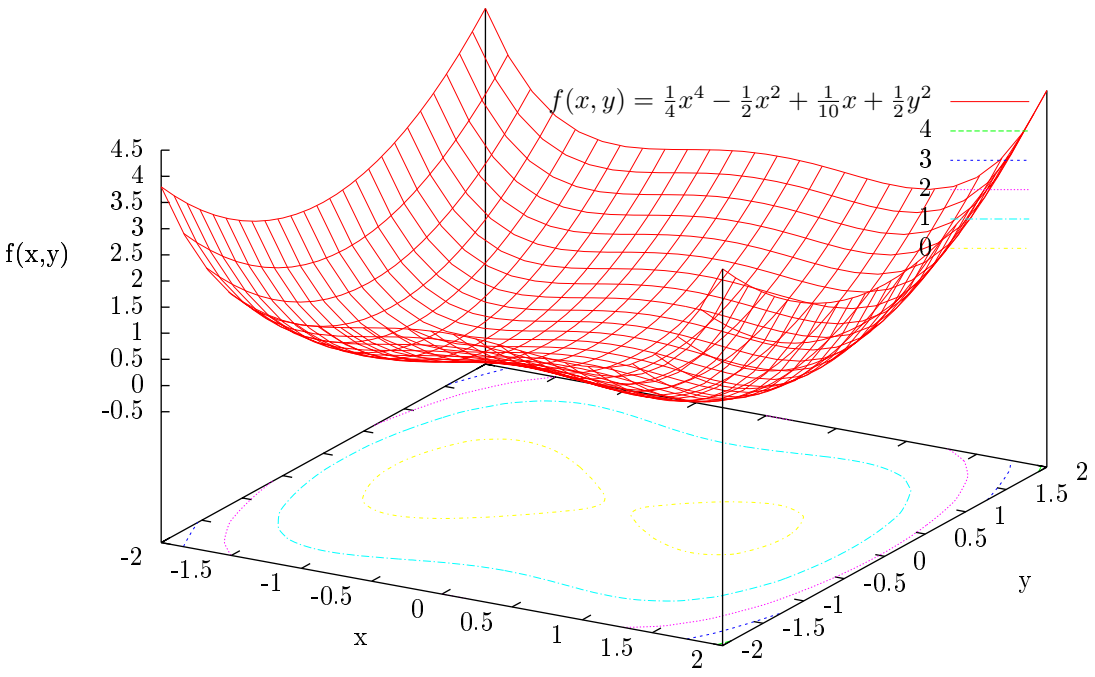

Figure 3: The two dimensional AluffiPentiny function.

We write the multidimensional objective functions in matrix notation. You would type equation (2) like this:

```
# min at -0.352386, X in [-10,10]^2
# The matrix notation X[1,2,] means: Access the the element in row 1 in coloumn 2.
# Therefore X[ ,2] means: Access all elements in coloumn 2.
AluffiPentiny02 <- function(X)
{
  1/4*X[,1]^4 - 1/2*X[,1]^2 + 1/10*X[,1] + 1/2*X[,2]^2
}
```

Whereby `X[ ,1]` replaced the x and `X[ ,2]` stands for the y. Let us suppose we want to compute an approximate solution within the lower bounds $\mathbf{x}_{lower} = \begin{pmatrix} -10 \\ -10 \end{pmatrix}$ and upper bounds $\mathbf{x}_{upper} = \begin{pmatrix} 10 \\ 10 \end{pmatrix}$. Naturally the bounds don't have to be symmetric. You initialize the optimization process with:

```
Example02 <- RcppGO(ObjectiveFunction=AluffiPentiny, Args=2, Lower = -10, Upper = 10)
```

The output of the optimization process is assigned to the variable `Example02`. Again typing `Example02` would result in a long chain of output. Accessing the best optained results is done by:

```
> Example02$GMemory
          x1             x2       fn_x
1  -1.045695 -0.0008089336 -0.3523846
2  -1.043545  0.0085946639 -0.3523379
3  -1.049650 -0.0112856804 -0.3523123
4  -1.036312 -0.0061996451 -0.3522451
5  -1.050161  0.0163825537 -0.3522380
6  -1.059847 -0.0025767241 -0.3521822
7  -1.033795  0.0109659926 -0.3521384
8  -1.031220 -0.0060378656 -0.3520984
9  -1.072199 -0.0038773126 -0.3516166
10 -1.062366 -0.0334229455 -0.3515422
```

The output is truncated and shows the top ten solutions generated during the optimization process. The first two coloumns represent the coordinates where the solutions were found and the third coloumn exhibits the corresponding function values.

3.2 The `plot()` function

The `plot.RcppGO` method provides two ways to visualize the optimization process of two dimensional optimization problems. Let us have a look at the parameters of the plot function, before proceeding to the examples.

- `x`: Requires the output from the `RcppGO` function.
- `...`: The `...` parameter is derived from the generic plot function. You can ignore it.
- `plot.type`: The plot type can be either 'static' or 'dynamic'.
- `delay`: The delay between the plot updates. Default is 0.3.
- `bestsolution`: Indicator of the overall best found solution. Default is `TRUE`.
- `nextposition`: The position of a particle in t+1. Default is `FALSE`.
- `velocity`: The velocity vector. Default is `FALSE`.
- `resForce`: The resultant force vector on a particle. Default is `FALSE`.
- `radius` The time dependent radius around a particle separating the acting forces. Default is `FALSE`.

The function is highly customizable with a pedagocial goal in mind. You can switch on and off various vector arrows referring to the particles position, velocity or force. Many parameter values, like the objective function or the upper and lower bounds, are taken into the plot method via `x`. The input for the `x` variable comes from the output of `RcppGO` function. Let's have a look at `Example02` from the previous section an see how we can visualize the before obtained results. We begin with a `static` plot:

```
plot(x=Example02, plot.type="static")
```

Figure 4 shows the result in `R`. The image consists of two panels. The left one exhibits a relief-like view of the objective function and the right one shows a contourplot. In both plots the function values are automatically coloured in topographical colours.

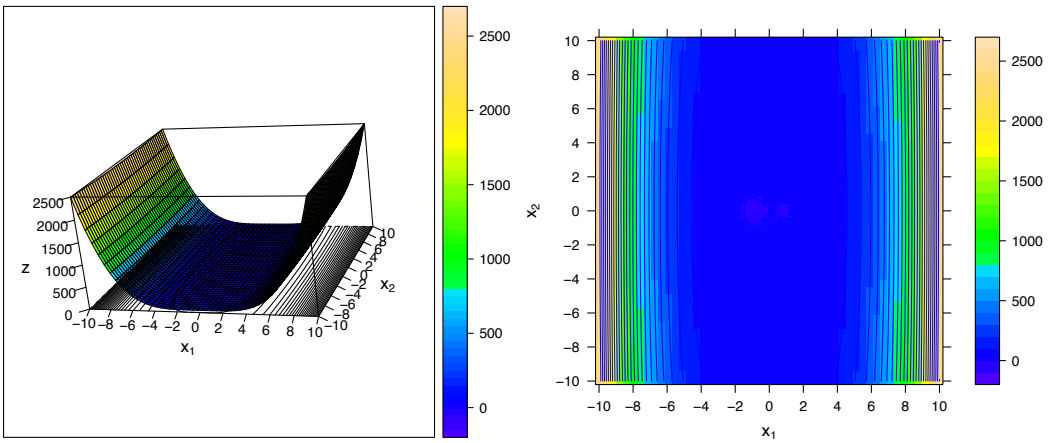

Figure 4: This figure shows the relief-like view of the search space (left) and a contourplot (right) when `plot.type=static`.

Let us now try the `dynamic` option.

```
plot(x=Example02, plot.type="dynamic")
```

You get an animated plot and thus can follow the particle movements (red dots) during the optimization process. The small box in the middle of figure 5 is the solution candidate with the best found fitness value. Feel free to play around with the parameter values of the plot function.

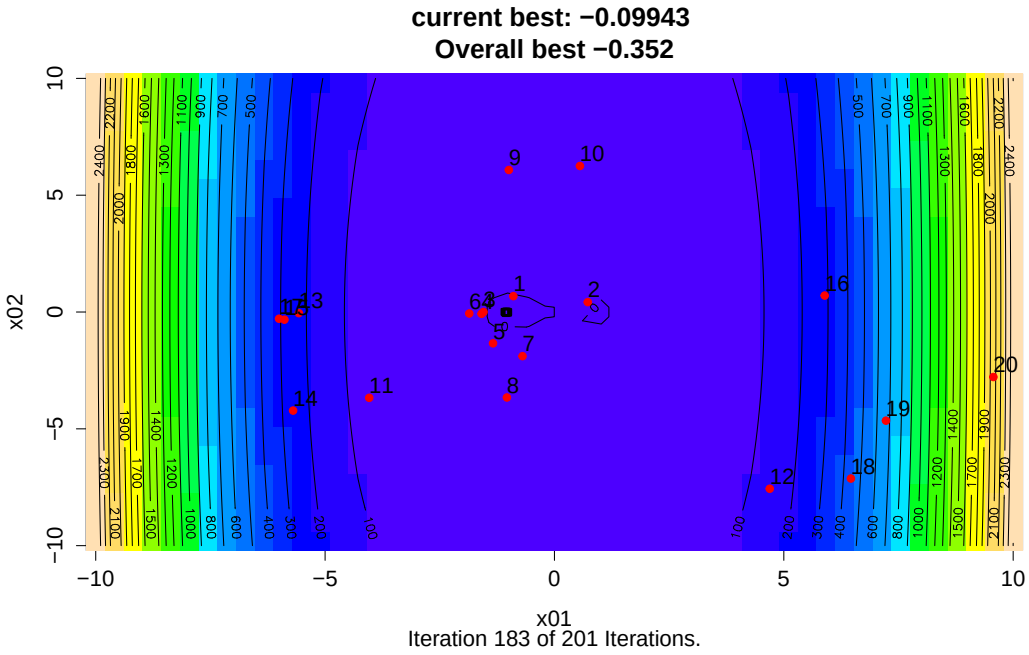

Figure 5: .

4 Performance

5 Discussion

6 Conclusion

References

[1] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8:239–287, 2009.

[2] Johann Dréo and Caner Candan. Different classifications of metaheuristics, 28 August 2011. URL `http://en.wikipedia.org/wiki/File:Metaheuristics_classification.svg`.

[3] Dirk Eddelbuetel and Romain Francois. Extending Rcpp. 2011.

[4] Dirk Eddelbuettel. From DEoptim to RcppDE: A case study in porting from C to C++ using Rcpp and RcppArmadillo. 2011.

[5] Dirk Eddelbuettel and Romain Francois. Writing a package that uses Rcpp. 2011.

[6] Dirk Eddelbuettel, Romain Francois, and Douglas Bates. RcppArmadillo: Rcpp integration for Armadillo templated linear algebra library. 2010. URL `http://cran.r-project.org//package=RcppArmadillo`.

[7] A. Kaveh and S. Talatahari. A Charged System Search With A Fly To Boundary Method For Discrete Optimum Design Of Truss Structures. *Asian Journal Of Civil Engineering (Building And Housing)*, 11(3):277–293, 2010.

[8] A Kaveh and S Talatahari. A Novel Heuristic Optimization Method: Charged System Search. *Acta Mechanica*, 213(3-4):267–289, 2010. ISSN 0001-5970. doi: 10.1007/s00707-009-0270-4.

[9] Ali Kaveh and Siamak Talatahari. An Enhanced Charged System Search For Configuration Optimization Using The Concept Of Fields Of Forces. *Structural and Multidisciplinary Optimization*, pages 1–13, 2010.

[10] Ali Kaveh and Siamak Talatahari. Geometry And Topology Optimization Of Geodesic Domes Using Charged System Search. *Structural and Multidisciplinary Optimization*, pages 1–15, 2010.

[11] Ali Kaveh and Siamak Talatahari. Optimal Design Of Skeletal Structures Via The Charged System Search Algorithm. *Structural and Multidisciplinary Optimization*, 41(6):893–911, 2010.

[12] Ali Kaveh and Siamak Talatahari. A general model for meta-heuristic algorithms using the concept of fields of forces. *Acta Mechanica*, 221:99–118, 2011.

[13] Ali Kaveh, Siamak Talatahari, and R. Sheikholeslami. An efficient charged system search using chaos for global optimization problems. *International Journal of Optimization in Civil Engineering*, 2:305–325, 2011.

[14] Wacek Kusnierczyk. *rbenchmark: Benchmarking routine for R*, 2012. URL `http://CRAN.R-project.org/package=rbenchmark`. R package version 1.0.0.

[15] Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. 2005.

[16] K.M. Mullen, D. Ardia, D.L. Gil, D. Windover, and J. Cline. DEoptim: An R Package for Global Optimization by Differential Evolution. *Journal of Statistical Software*, 40(6):1–26, 2011.

[17] Magnus Erik Hvass Pedersen. *Tuning & Simplifying Heuristical Optimization*. PhD thesis, University of Southampton Computational Engineering and Design Group School of Engineering Sciences, 2010.

[18] R Development Core Team. *R: A Language And Environment For Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL `http://www.r-project.org`.

[19] R Development Core Team. *Writing R Extensions*, 2012.

[20] R Development Core Team. *The R language definition*, 2012.

[21] R Development Core Team. *R Installation and Administration*, 2012.

[22] Conrad Sanderson. Armadillo: An open source C++ Algebra Library for Fast Prototyping and Computationally Intensive Experiments. Technical report, NICTA, 2010. URL `http://arma.sf.net`.

[23] Deepayan Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. URL `http://lmdvr.r-forge.r-project.org`. ISBN 978-0-387-75968-5.

[24] P. A. Tipler and G. Mosca. *Physics for scientists and engineers*, volume 1. Macmillan, 2008.

[25] Thomas Weise. *Global Optimization Algorithms - Theory and Application*. 2009.

[26] Xin-She Yang. *Engineering Optimization: An Introduction with Metaheuristic Applications*. John Wiley & Sons, 2010.

[27] Xin-She Yang. Test problems in optimization. *Department of Engineering, University of Cambridge, Cambridge CB2 1PZ, UK*, 2010.