Generalized Simulated Annealing for Global Optimization: The GenSA Package

An Application to Non-Convex Optimization in Finance and Physics

by Yang Xiang, Sylvain Gubian, Brian Suomela and Julia Hoeng

Abstract Many problems in statistics, finance, biology, pharmacology, physics, mathematics, economics, and chemistry involve determination of the global minimum of multidimensional functions. R packages for different stochastic methods such as genetic algorithms and differential evolution have been developed and successfully used in the R community. Based on Tsallis statistics, the R package GenSA was developed for generalized simulated annealing to process complicated non-linear objective functions with a large number of local minima. In this paper we provide a brief introduction to the R package and demonstrate its utility by solving a non-convex portfolio optimization problem in finance and the Thomson problem in physics. **GenSA** is useful and can serve as a complementary tool to, rather than a replacement for, other widely used R packages for optimization.

Introduction

Many problems in statistics, biology, physics, mathematics, economics, and chemistry involve the determination of the global minimum of multidimensional functions (Agostini et al., 2006; Serra et al., 1997; Guire et al., 2010; Xiang et al., 1997; Xiang and Gong, 2000a; Xiang et al., 2000). Methods of optimization can generally be classified as either deterministic or stochastic. For simple non-convex functions with only a few dimensions and well separated local minima, standard deterministic methods such as simplex optimization, steepest descent method, and the quasi-Newton method can provide reasonable results. While being fast, deterministic methods have the tendency to trap the system within a local minimum. By comparison, many stochastic methods are far less likely to get stuck in a local minimum, and may find a good approximation of the global minimum using a modest amount of computation. Many stochastic methods have been developed for the solution of global optimization problems such as genetic algorithms (Holland, 1975), evolution algorithms (Storn and Price, 1997), simulated annealing (Kirkpatrick et al., 1983), and taboo search (Glover et al., 1993; Cvijovic and Klinowski, 2002, 1995).

In metallurgy, annealing a molten metal causes it to reach its crystalline state which is the global minimum in terms of thermodynamic energy. The simulated annealing algorithm was developed to simulate the annealing process to find a global minimum of the objective function (Kirkpatrick et al., 1983). In the simulated annealing algorithm, the objective function is treated as the energy function of a molten metal and one or more artificial temperatures are introduced and gradually cooled, analagous to the annealing technique. This artificial temperature (or set of temperatures) acts as a source of stochasticity, which is convenient for the systems to eventually escape from local minima. Near the end of the annealing process, the system is hopefully inside the attractive basin of the global minimum (or in one of the global minima if more than one global minimum exists).

In contrast to the simulation of the annealing process of molten metal, genetic algorithms (Holland, 1975) were developed by mimicing the process of natural evolution. A population of strings which encode candidate solutions for an optimization problem evolve over many iterations toward better solutions. In general the solutions are represented by bitstrings, but other encodings such as floating-point numbers are also widely used. The evolution usually starts from a population of randomly generated individuals. In each generation, the fitness of each individual in the population is evaluated. New members of the population in the next generation are generated by cross-over, mutation, and selection (based on their fitness). Differential evolution belongs to a class of genetic algorithms.

The basic idea behind the taboo search method (Glover et al., 1993) is to forbid the search to return to points already visited in the (usually discrete) search space, at least for the upcoming few steps. Similar to simulated annealing, taboo search can temporarily accept new solutions which are worse than earlier solutions, in order to avoid paths already investigated. Taboo search has traditionally been applied to combinatorial optimization problems and it has been extended to be applicable to continuous global optimization problems by a discrete approximation (encoding) of the problem (Cvijovic and Klinowski, 2002, 1995).

For a review of stochastic optimization algorithms, please refer to Fouskakis and Draper (2002).

The R language and environment for statistical computing enables rapid prototyping of algorithms, access to a wide range of tools for statistical modeling, and the ability to easily generate customized plots of results. These advantages make use of R preferable in many situations over the use of other programming languages like Java, C++, Fortran, and Pascal (Mullen et al., 2011).

Theussl (2011) provided a comprehensive summary of the currently available R packages for solving optimization problems. No one method is the best for all the optimization problems. The best method for one specific problem depends on the problem itself. For example, simulated annealing could be chosen for optimization problems such as scheduling in the multiprocessor flowshop (Figielska, 2009), or pathway-based microarray analysis (Pavlidis et al., 2011), while genetic algorithms could be more appropriate for optimization problems such as design of an ATM network (Thompson and Bilbro, 2000). R users can choose the most appropriate method/package to handle different types of global optimization problems facilitated by the work of Mullen et al. (2011); Ardia et al. (2011); Mebane Jr and Sekhon (2011); Powell (2009) and other contributors listed in (Theussl, 2011). For example, the Nelder-Mead and BFGS quasi newton method in the function `optim` are appropriate to handle non-smooth and smooth functions with few local minima; packages for stochastic searches, such as **DEoptim** (Mullen et al., 2011; Ardia et al., 2011) and **rgenoud** (Mebane Jr and Sekhon, 2011), are a good choice for more complicated objective functions with many local minima. While the simulated annealing algorithm is also suitable to solve complicated objective functions with many local minima, the only package of simulated annealing serving as a general purpose continuous solver in R is `sann` in `optim` (Theussl, 2011). Several R packages, **ConsPlan** (VanDerWal and Januchowski, 2010), **likelihood** (Murphy, 2008), **dclone** (Sólymos, 2010), and **subselect** (Cerdeira et al., 2011), make use of the simulated annealing algorithm within their specific applications. As `sann` in `optim` cannot handle simple box constraints and it performs poorly compared to **DEoptim** for the Rastrigin function (Ardia et al., 2011), many R users would benefit from a new R package for simulated annealing which is suitable for box constraints and quickly solves global optimization problems.

Here we have developed an innovative R package **GenSA**, which is an implementation of modified generalized simulated annealing, which outperforms the classical simulated annealing algorithm (Xiang et al., 1997; Xiang and Gong, 2000a,b; Serra et al., 1997). **GenSA** can process complicated non-linear objective functions with a large number of local minima. The kernel function of **GenSA** is written in C++ to ensure the package runs as efficiently as possible.

In the remainder of this paper we will elaborate further on the background and improvements of GSA (in Section 1) and on the content of the package **GenSA** (in Section 2). The utility of this R package for financial and physical applications is demonstrated by solving a non-convex portfolio optimization problem and the famous Thomson problem in physics, respectively (in Section 3).

Generalized simulated annealing

Classical simulated annealing (CSA) was proposed by Kirkpatrick et al. (1983). Due to the inherent statistical nature of simulated annealing, in principle local minima can be hopped over more easily than for gradient methods. Using the simulated annealing technique, one or more artificial temperatures are introduced and gradually cooled to simulate thermal noise. A fast simulated annealing (FSA) method was proposed by Szu and Hartley (Szu and Hartley, 1987). CSA and FSA can be generalized according to Tsallis statistics with a unified picture called the generalized simulated annealing algorithm (Tsallis and Stariolo, 1996). GSA uses a distorted Cauchy-Lorentz visiting distribution, with its shape controlled by the parameter q_v

$$g_{q_v}(\Delta x(t)) \propto \frac{\left[T_{q_v}(t)\right]^{-\frac{D}{3-q_v}}}{\left[1+(q_v-1)\frac{(\Delta x(t))^2}{\left[T_{q_v}(t)\right]^{\frac{2}{3-q_v}}}\right]^{\frac{1}{q_v-1}+\frac{D-1}{2}}}. \tag{1}$$

Here t is the artificial time. This visiting distribution is used to generated a trial jump distance $\Delta x(t)$ of variable $x(t)$ under artificial temperature $T_{q_v}(t)$. The trial jump is accepted if it is downhill (in terms of the objective function). If the jump is uphill it might be accepted according to an acceptance probability. A generalized Metropolis algorithm is used for the acceptance probability:

$$p_{q_a} = \min\left\{1, \left[1-(1-q_a)\beta\Delta E\right]^{\frac{1}{1-q_a}}\right\}, \tag{2}$$

where q_a is a parameter. For $q_a < 1$, zero acceptance probability is assigned to the cases where

$$\left[1-(1-q_a)\beta\Delta E\right] < 0. \tag{3}$$

The artificial temperature $T_{q_v}(t)$ is decreased according to

$$T_{q_v}(t) = T_{q_v}(1)\frac{2^{q_v-1}-1}{(1+t)^{q_v-1}-1}. \quad (4)$$

When $q_v = 1$ and $q_a = 1$, GSA recovers CSA; when $q_v = 2$ and $q_a = 1$, GSA recovers FSA. When $q_v > 2$, the cooling is faster than that of CSA and FSA. When $T \to 0$, GSA behaves like the steepest descent algorithm. GSA not only converges faster than FSA and CSA, but also has the ability to escape from a local minimum more easily than FSA and CSA. Since GSA has a large probability to have a long jump, the probability of finding the global minimum with GSA is larger than that with FSA and CSA. Bigger q_v (< 3) makes the cooling faster and jumping further. Negative q_a with bigger absolute value makes GSA accept less hill climbing. Default values of q_v and q_a in our package **GenSA** are set to be 2.62 and -5 respectively according to our experience and our package works well with the default values in many applications. We also provide users the flexibility of changing the value of q_v and q_a easily in the *control* argument of **GenSA** for advanced usage of **GenSA**. For example, users can set any value between 2 and 3 for q_v and any value < 0 for q_a.

A simple technique to speed up the convergence is to set the acceptance temperature equal to the visiting temperature divided by the number of time steps, i.e. $T_{q_a} = \frac{T_{q_v}}{t}$. Our testing shows that this simple technique works well (Xiang et al., 2000), so this technique is used in **GenSA** to speed up the convergence.

For more details on GSA, we refer the readers to Tsallis and Stariolo (1996), Xiang and Gong (2000a) and Xiang et al. (1997).

The package GenSA

The core function of package **GenSA** is the function `GenSA`, whose arguments are:

- `par`: Numeric vector. Initial values for the parameters to be optimized over. Default is `NULL`, in which case initial values will be generated automatically.
- `lower`: Numeric vector with length of `par`. Lower bounds for the parameters to be optimized over.
- `upper`: Numeric vector with length of `par`. Upper bounds for the parameters to be optimized over.
- `fn`: A function to be minimized, with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.
- `...`: allows the user to pass additional arguments to the function `fn`.
- `control`: A list of control parameters, discussed below.

The `control` argument is a list that can be used to control the behavior of the algorithm. Some components of `control` are the following:

- `maxit`: Integer. Maximum number of iterations of the algorithm. Default value is 5000, which could be increased by the user for the optimization of a very complicated objective function.
- `threshold.stop`: Numeric. The program will stop when the objective function value is <= `threshold.stop`. Default value is NULL.
- `smooth`: Logical. TRUE when the objective function is smooth, or differentiable almost everywhere, and FALSE otherwise. Default value is TRUE.
- `max.call`: Integer. Maximum number of calls of the objective function. Default is 10000000.
- `max.time`: Numeric. Maximum running time in seconds.
- `temperature`: Numeric. Initial value for temperature.

The default values of the control components are set for a complicated optimization problem. For typical optimization problems with medium complexity, `GenSA` can find a reasonable solution quickly so the user is strongly recommended to let `GenSA` stop earlier by setting `threshold.stop` to the expected function value, or by setting `max.time` if the user just want to run `GenSA` for `max.time` seconds, or by setting `max.call` if the user just want to run `GenSA` within `max.call` function calls. Please refer to the examples below. For very complicated optimization problems, the user is recommended to increase `maxit` and `temperature`.

The returned value is a list with the following fields:

- par: The best set of parameters found.
- value: The value of fn corresponding to par.
- trace.mat: A matrix which contains the history of the algorithm. (By columns: Step number, temperature, current objective function value, current minimum objective function value).
- counts: Total number of calls of the objective function.

For more details about **GenSA**, the reader can refer to the manual (by typing ?GenSA).

Packages **DEoptim** and **rgenoud** have been widely used to successfully solve optimization problems arising in diverse fields (Ardia et al., 2011; Mullen et al., 2011; Mebane Jr and Sekhon, 2011). Since these two packages both use evolutionary strategies (Ardia et al., 2011), we have chosen one of them, **DEoptim**, as our gold standard for comparison purposes of various global optimization packages. Similar to Ardia et al. (2011), let us briefly illustrate the usage of package **GenSA** with the minimization of the Rastrigin function with 2 dimensions, which is a standard test function for global optimization:

```
> Rastrigin <- function(x) \{
+   fn.call <<- fn.call + 1
+   sum(x^2 - 10 * cos(2 * pi * x)) + 10 * length(x)
+ \}
```

The Rastrigin function with 2 dimensions has many local minima. The global minimum $f^{opt} = 0$ at point $x^{opt} = (0, 0)$. Please note that the dimension of x of the above Rastrigin function can be more than 2.

Since we chose **DEoptim** as our gold standard for global optimization packages, let's run **DEoptim** first:

```
> options(digits = 10)
> dimension <- 2
> lower <- rep(-5.12, dimension)
> upper <- rep(5.12, dimension)
> set.seed(1234)
> sink("tmp.txt")
> fn.call <<- 0
> library(DEoptim)
> out.DEoptim <- DEoptim(fn = Rastrigin, lower = lower, upper = upper,
+                        control = list(storepopfrom = 1))
> sink(NULL)
> out.DEoptim$optim[c(1, 2, 3)]
$bestmem
            par1             par2
-4.775211422e-07  6.390004611e-08

$bestval
[1] 4.60502747e-11

$nfeval
[1] 402

> cat("DEoptim call functions", fn.call, "times.{\textbackslash}n")
DEoptim call functions 4020 times.
```

The best value that **DEoptim** found is 4.60502747e-11, which is fairly close to the global minimum 0. However the latest version of **DEoptim**, 2.2.1, gave a wrong number of function calls nfeval, 402, which should be 4020 as shown by fn.call.

The simulated annealing solver sann in optim is applied to the Rastrigin function.

```
> set.seed(1234)
> x.ini <- lower + runif(length(lower)) * (upper - lower)
> fn.call <- 0
> out.sann <- optim(par = x.ini, fn = Rastrigin, method = "SANN")
> out.sann[c("value","par","counts")]
$value
[1] 3.980605068

$par
```

```
[1] -1.98836973902 -0.00123560529

$counts
function gradient
   10000       NA
```

sann does not find the global minimum although 10000 function calls (the default stopping criterion in sann) are performed.

GenSA also searches for a minimum of the Rastrigin function between the same lower and upper bounds. A call to GenSA can be made as follows:

```
> set.seed(1234)
> library(GenSA)
> expected.val <- 0
> absTol <- 1e-13
> fn.call <- 0
> out.GenSA <- GenSA(par = NULL, lower = lower, upper = upper, fn = Rastrigin,
+                    control = list(threshold.stop = expected.val + absTol))
> out.GenSA[c("value", "par", "counts")]
$value
[1] 0

$par
[1]  2.750668687e-12 -2.889218652e-12

$counts
[1] 196

> cat("GenSA call functions", fn.call, "times.{\textbackslash}n")
GenSA call functions 196 times.
```

The above call specifies a random vector as initial values for the parameters by setting par as NULL, the lower and upper bounds on the parameters, and, via the control argument, that we will stop searching after GenSA finds the expected global minimum $f^{opt} = 0$ within the absolute tolerance $\Delta = 1e{-}13$. GenSA actually found the global minimum $f^{opt} = 0$ after 196 function calls. GenSA and DEoptim both converge to the the global minimum but GenSA obtains a more accurate result with fewer function calls than DEoptim. sann performs poorly since it does not find the global minimum after a much larger number of function calls (10000) compared to DEoptim and GenSA.

The result of DEoptim can be further improved with a local search, large-scale bound-constrained BFGS (Byrd et al., 1995; Zhu et al., 1997), as below:

```
> out.DEoptim_BFGS <- optim(par = out.DEoptim$optim$bestmem, fn = Rastrigin,
+                           lower = lower, upper = upper, method = "L-BFGS-B")
> out.DEoptim_BFGS[c("value","par")]
$value
[1] 0

$par
            par1             par2
-9.362433236e-12  1.250185258e-12
```

So DEoptim + large-scale bound-constrained BFGS generated a comparable result to GenSA, however, with more function calls. L-BFGS-B and bobyqa are good local search methods for smooth functions while Nelder-Mead is recommended for local search of non-smooth functions although Nelder-Mead can manage both smooth and non-smooth functions (Zhu et al., 1997; Nash, 1990; Powell, 2009). It could be a good idea to run a local search for further refinement after running DEoptim. The user of GenSA does not need to run a local search for refinement after running GenSA since the local search refinement was performed within GenSA.

The Rastrigin function with 30 dimentions belongs to the most difficult class of problems for many optimization algorithms (including evolutionary programming) (Yao et al., 1999; Mebane Jr and Sekhon, 2011). However GenSA can find the global minimum in no more than 2 seconds (CPU: Xeon E5450), as shown below.

```
> set.seed(1234)
> dimension <- 30
> lower <- rep(-5.12, dimension)
```

```
> upper <- rep(5.12, dimension)
> fn.call <- 0
> out.GenSA <- GenSA(lower = lower, upper = upper, fn = Rastrigin,
+                    control = list(max.time=1.9, verbose=TRUE))
> out.GenSA[c("value")]
$value
[1] 0
```

Of course the time for solving the above problem using **GenSA** may vary depending on the computing power used. **DEoptim** and sann did not find the global minimum of Rastrigin function with 30 dimensions by using default settings.

To compare the performance of the above packages/methods more precisely, **GenSA**, **DEoptim**, sann, and **DEoptim**+ large-scale bound-constrained BFGS (denoted as **DEoptim_BFGS**) were run 100 times randomly. The testing function was Rastrigin with 2 dimensions. We noticed that the implementation in the package **DEoptim** is very flexible, but only default values of parameters were used in the comparison. Similarly only default values of parameters in **GenSA** were used in the comparison. Various absolute tolerances $\Delta \in \{1e{-}05,\ 1e{-}07,\ 1e{-}08,\ 1e{-}09\}$ were explored. The global minimum $f^{opt} = 0$ was considered to have been reached if the current function value f surpasses $f^{opt} + \Delta$. A run was successful if the global minimum was reached. Figures 1 and 2 show the percentage of the 100 runs that were successful, and the average number of function calls to reach the global minimum (for the successful runs), respectively. The error bar in Figure 2 represents the average number of function calls $\pm$ its standard error (SE).

None of the 100 runs for sann were successful, so the result of sann is not shown in Figure 1 and Figure 2.

In Figure 1, the percentage of successful runs for **DEoptim** was 88% for $\Delta = 1e{-}05$, but decreased to 22% for $\Delta = 1e{-}09$. The percentage of successful runs for **GenSA** and **DEoptim_BFGS** were 100% and 98% respectively for all absolute tolerances Δ. **GenSA**, **DEoptim** and **DEoptim_BFGS** had a similar percentage of successful runs when the absolute tolerance Δ was large, while **GenSA** and **DEoptim_BFGS** were much more likely to be successful than **DEoptim** as the absolute tolerance Δ was set to smaller values.

In Figure 2, the average number of function calls for **GenSA** was 479 for $\Delta = 1e{-}05$, and increased only slightly to 483 for $\Delta = 1e{-}09$. The average number of function calls for **DEoptim** was 2692 for $\Delta = 1e{-}05$, and increased significantly to 3344 for $\Delta = 1e{-}09$. The average number of function calls for **DEoptim_BFGS** was 2829 for $\Delta = 1e{-}05$, and increased to 3877 for $\Delta = 1e{-}09$. A student t-test was performed to determine if the average number of function calls of **GenSA** was significantly smaller than the average number of function calls of **DEoptim** and **DEoptim_BFGS**. The p-values shown above the error bars in Figure 2 indicate that the average number of function calls of **GenSA** is significantly smaller than the average number of function calls of both **DEoptim** and **DEoptim_BFGS**. The Wilcoxon rank-sum test gave a similar result.

GenSA, **DEoptim**, **DEoptim_BFGS** solve the Rastrigin problem more efficiently than sann. To show how the performance of the above R packages change with the dimensions of testing functions, we test them in more functions: generalized Rosenbrock function (ROS) (Mebane Jr and Sekhon, 2011) with 4 dimensions (dimension = 2, 10, 20, 30), Rastrigin function (RAS) (Mebane Jr and Sekhon, 2011) with 4 dimensions (dimension = 2, 10, 20, 30), Branin function (BRA) (Serra et al., 1997), Goldstein-Prices function (GP) (Serra et al., 1997). The range of coordindates of functions Rosenbrock, Rastrigin, and Goldstein-Prices, are $[-30,\ 30]$, $[-5.12,\ 5.12]$, and $[-2,\ 2]$, respectively. The range of the first and the second coordinate of function Branin are $[-5,\ 10]$ and $[0,\ 15]$ respectively. Four different tolerance levels, $1e{-}5,\ 1e{-}7,\ 1e{-}8,\ 1e{-}9$, were used. The results for tolerance level $1e{-}08$ are shown in Table 1. For Rosenbrock function with low dimension 2, **GenSA**, **DEoptim** and **DEoptim_BFGS** found the global minimum with 100% success. For Rosenbrock function with dimension $= 10,\ 20,\ 30$, **DEoptim** and **DEoptim_BFGS** failed to find the global minimum (tolerance level $1e{-}08$), while **GenSA** found the global minimum (tolerance level $1e{-}08$) with a 100% success rate. For Rosenbrock function with dimension $2,\ 10,\ 20,\ 30$, the average number of function calls of **GenSA** is significantly smaller than that of **DEoptim** and **DEoptim_BFGS**. For Rastrigin function with dimension ≤ 30, the success rate of **GenSA** is 100% regardless of the number of dimensions, but the success rate of **DEoptim** and **DEoptim_BFGS** decreased drastically with an increasing number of dimensions. **DEoptim** failed to find the global minimum (tolerance level $1e{-}08$) of the Rastrigin function for dimension ≥ 20. For Branin function and Goldstein-Prices function, although **GenSA**, **DEoptim** and **DEoptim_BFGS** found the global minimum with a 100% success rate, the average number of function calls of **GenSA** is significantly smaller than that of **DEoptim** and **DEoptim_BFGS**.

Please note that the performance testing was based on the usage of the R packages with only the default parameter settings of **GenSA**, **DEoptim**, and optim.

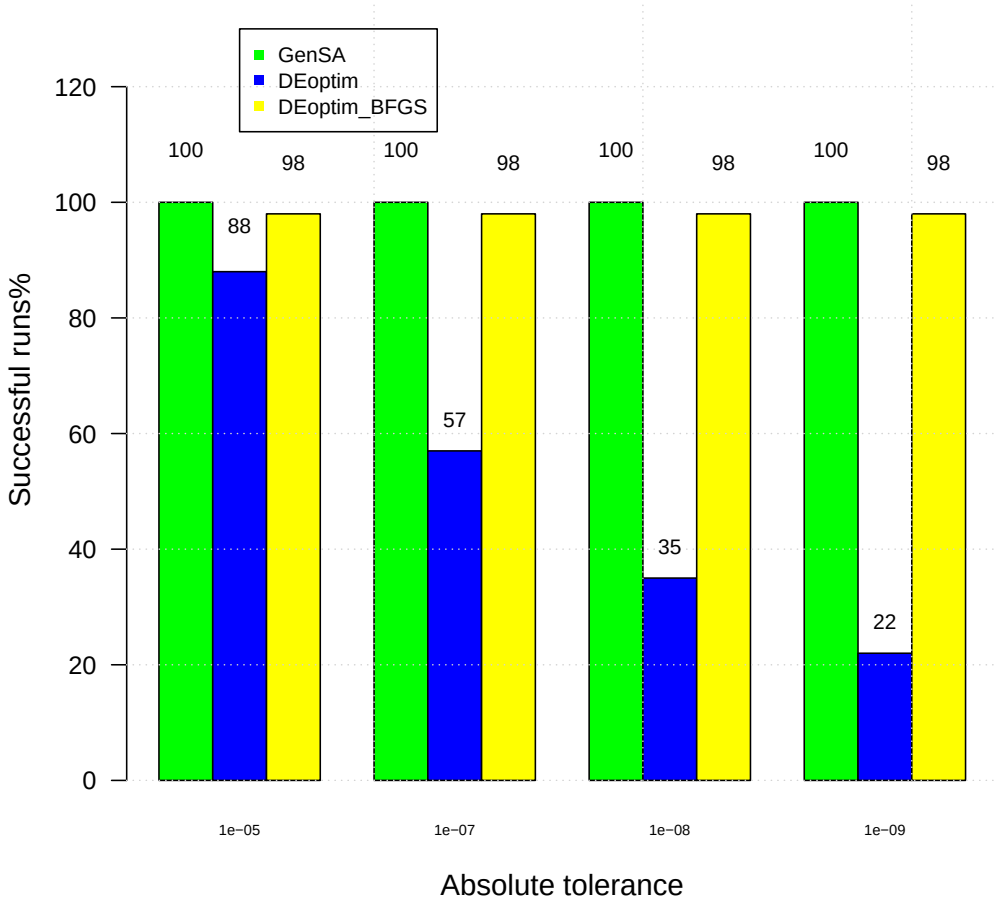

Figure 1: Percentage of successful runs vs. different absolute tolerances for **GenSA** (green), **DEoptim** (blue), and **DEoptim_BFGS** (yellow) in the test function Rastrigin. None of the runs for sann was successful, so the result of sann is not shown in this figure.

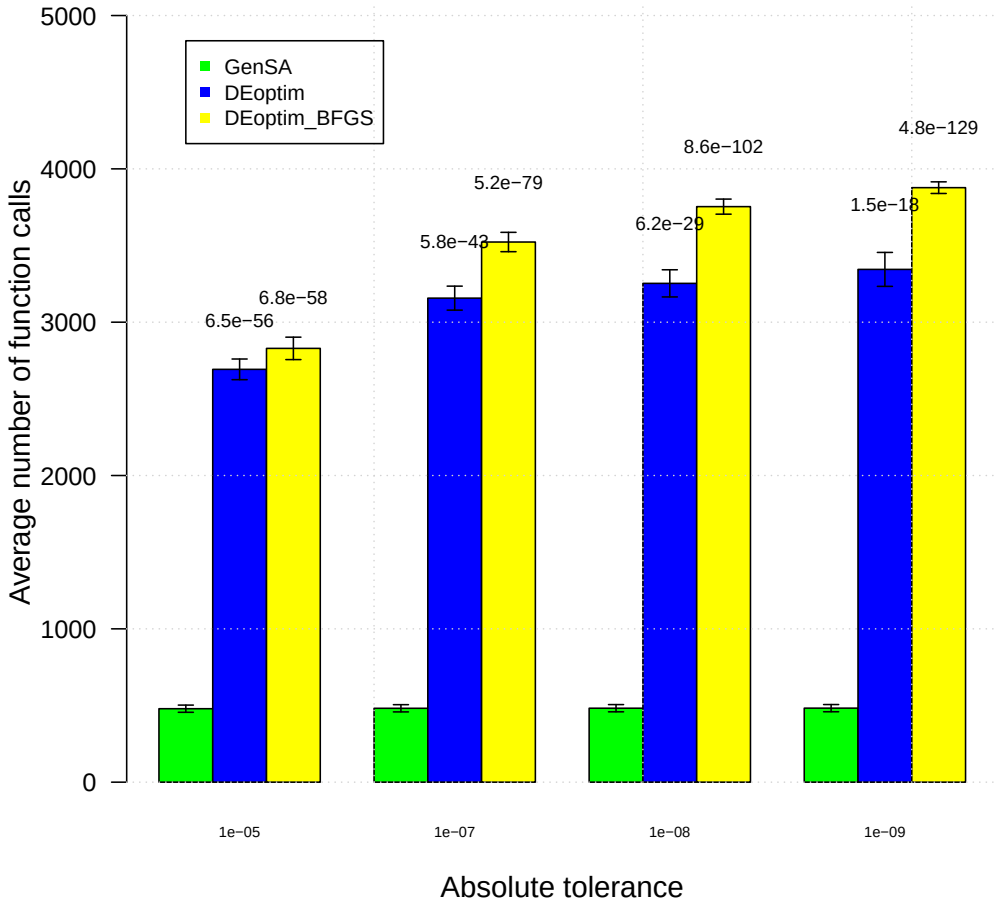

Figure 2: Average number of function calls to reach the global minimum in the successful runs for various absolute tolerances using **GenSA** (green), **DEoptim** (blue), and **DEoptim_BFGS** (yellow) in the test function Rastrigin. The error bar represents the average number of function calls $\pm$ its standard error. The p-values for comparison **DEoptim** and **DEoptim_BFGS** vs. **GenSA** are found above the error bar. None of the runs for sann was successful, so the result of sann is not shown in this figure.

func	GenSA	DEoptim	DEoptim-BFGS	func	GenSA	DEoptim	DEoptim-BFGS
ROS-2D	100.0% 106.0 (B) 1617.8 (A) ± 167.9 (S) 4689.0 (W) 1618.7 (T)	100.0% 1561.0 (B) 2483.1 (A) ± 50.5 (S) 3864.0 (W) 4020.0 (T)	100.0% 1561.0 (B) 2483.1 (A) ± 50.5 (S) 3864.0 (W) 4108.9 (T)	**RAS-10D**	100.0% 2919.0 (B) 5878.2 (A) ± 165.8 (S) 11820.0 (W) 5879.4 (T)	0.0% NA (B) NA (A) ± NA (S) NA (W) 20100.0 (T)	2.0% 20248.0 (B) 20258.5 (A) ± 10.5 (S) 20269.0 (W) 20290.3 (T)
ROS-10D	100.0% 5545.0 (B) 17562.3 (A) ± 696.3 (S) 31234.0 (W) 17564.1 (T)	0.0% NA (B) NA (A) ± NA (S) NA (W) 20100.0 (T)	0.0% NA (B) NA (A) ± NA (S) NA (W) 21657.8 (T)	**RAS-20D**	100.0% 6869.0 (B) 14682.6 (A) ± 318.2 (S) 21893.0 (W) 14683.6 (T)	0.0% NA (B) NA (A) ± NA (S) NA (W) 40200.0 (T)	0.0% NA (B) NA (A) ± NA (S) NA (W) 40585.8 (T)
ROS-20D	100.0% 11866.0 (B) 33547.9 (A) ± 1401.9 (S) 65427.0 (W) 33552.2 (T)	0.0% NA (B) NA (A) ± NA (S) NA (W) 40200.0 (T)	0.0% NA (B) NA (A) ± NA (S) NA (W) 45165.9 (T)	**RAS-30D**	100.0% 12713.0 (B) 27820.7 (A) ± 683.2 (S) 56432.0 (W) 27823.6 (T)	0.0% NA (B) NA (A) ± NA (S) NA (W) 60300.0 (T)	0.0% NA (B) NA (A) ± NA (S) NA (W) 60922.8 (T)
ROS-30D	100.0% 30192.0 (B) 52874.3 (A) ± 1403.8 (S) 117795.0 (W) 52878.8 (T)	0.0% NA (B) NA (A) ± NA (S) NA (W) 60300.0 (T)	0.0% NA (B) NA (A) ± NA (S) NA (W) 67613.9 (T)	**BRA**	100.0% 26.0 (B) 35.7 (A) ± 0.7 (S) 51.0 (W) 36.7 (T)	100.0% 287.0 (B) 788.9 (A) ± 27.4 (S) 1769.0 (W) 4020.0 (T)	100.0% 287.0 (B) 788.9 (A) ± 27.4 (S) 1769.0 (W) 4046.3 (T)
RAS-2D	100.0% 41.0 (B) 482.4 (A) ± 23.6 (S) 1242.0 (W) 483.5 (T)	35.0% 1975.0 (B) 3253.3 (A) ± 88.6 (S) 3991.0 (W) 4020.0 (T)	98.0% 1975.0 (B) 3753.6 (A) ± 49.2 (S) 4041.0 (W) 4115.5 (T)	**GP**	100.0% 41.0 (B) 158.7 (A) ± 11.5 (S) 585.0 (W) 159.7 (T)	100.0% 794.0 (B) 1023.0 (A) ± 9.4 (S) 1263.0 (W) 4020.0 (T)	100.0% 794.0 (B) 1023.0 (A) ± 9.4 (S) 1263.0 (W) 4125.0 (T)

Table 1: Comparison result for tolerance level 1e-08. ROS: Rosenbrock function; RAS: Rastrigin function ; BRA: Branin function; GP: Goldstein-Price function. For ROS and RAS, the number after the hyphen gives the dimension. E.g. ROS-30D is Rosenbrock with 30 dimensions. For each cell in this table, we show the percentage of successful runs %, minimum (B), average (A), standard error (S), and maximum (W), of number of function calls to reach the global minimum among all the successful runs, and average number of total function calls among the 100 runs (denoted by T) respectively.

Derivative-based deterministic methods, e.g. BFGS, are recommended for smooth functions with only a few local minima such as generalized Rosenbrock functions with few dimensions. Pure stochastic methods such as differential evolution and generalized simulated annealing are for complicated functions with multiple minima such as the Rastrigin function, since local minima can be hopped much more easily in stochastic methods than in deterministic methods, due to the inherent statistical nature of stochastic methods.

Application

Risk allocation portfolios

Mean-risk models were developed in the 1950s for portfolio selection problems. Value-at-Risk (VaR) and Conditional Value-at-Risk (CVaR) are the most popular measures of downside risk. Mullen et al. (2011) and Ardia et al. (2011) used **DEoptim** to find the portfolio weights for which the portfolio has the lowest CVaR and each investment can contribute at most 22.5% to total portfolio CVaR risk. For details, please refer to (Mullen et al., 2011; Ardia et al., 2011). The code for objective function in portfolio optimization are from Ardia et al. (2011).

```
> library("quantmod")
> tickers <- c("GE", "IBM", "JPM", "MSFT", "WMT")
> getSymbols(tickers, from = "2000-12-01", to = "2010-12-31")
[1] "GE"   "IBM"  "JPM"  "MSFT" "WMT"
> P <- NULL
> for(ticker in tickers) \{
+   tmp <- Cl(to.monthly(eval(parse(text = ticker))))
+   P <- cbind(P, tmp)
+ \}
> colnames(P) <- tickers
> R <- diff(log(P))
> R <- R[-1,]
```

```
> mu <- colMeans(R)
> sigma <- cov(R)
> library("PerformanceAnalytics")
> pContribCVaR <- ES(weights = rep(0.2, 5),
+                    method = "gaussian", portfolio_method = "component",
+                    mu = mu, sigma = sigma)$pct_contrib_ES
> obj <- function(w) \{
+   fn.call <<- fn.call + 1
+   if (sum(w) == 0) { w <- w + 1e-2 }
+   w <- w / sum(w)
+   CVaR <- ES(weights = w,
+              method = "gaussian", portfolio_method = "component",
+              mu = mu, sigma = sigma)
+   tmp1 <- CVaR$ES
+   tmp2 <- max(CVaR$pct_contrib_ES - 0.225, 0)
+   out <- tmp1 + 1e3 * tmp2
+   return(out)
+ \}
```

We first run **DEoptim** with the same setting as in (Ardia et al., 2011).

```
> set.seed(1234)
> fn.call <- 0
> sink("tmp.txt")
> out.DEoptim <- DEoptim(fn = obj, lower = rep(0, 5), upper = rep(1, 5))
> sink(NULL)
> fn.call.DEoptim <- fn.call
> out.DEoptim$optim$bestval
[1] 0.1142884416
> out.DEoptim$optim$nfeval
[1] 402
> cat("DEoptim call functions", fn.call.DEoptim, "times.{\textbackslash}n")
DEoptim call functions 10050 times.
```

The minimum found by **DEoptim** was 0.1142884416 after 10050 function calls. The latest version of **DEoptim**, 2.2.1, gave a wrong number of function calls `nfeval` again. Nelder-Mead method can be used for further refinement since this objective function is non-smooth.

```
> out.DEoptim.fur <- optim(par = out.DEoptim$optim$bestmem, fn = obj, method = "Nelder-Mead")
> out.DEoptim.fur$value
[1] 0.1141564043
```

Note that the best value found by **DEoptim** was further improved by the use of a local search method. We ran **GenSA** as in the example below:

```
> set.seed(1234)
> fn.call <<- 0
> out.GenSA <- GenSA(fn = obj, lower = rep(0, 5), upper = rep(1, 5),
+                    control = list(smooth = FALSE, max.call = 3000))
> fn.call.GenSA <- fn.call
> out.GenSA$value
[1] 0.1141484884
> out.GenSA$counts
[1] 3000
> cat("GenSA call functions", fn.call.GenSA, "times.{\textbackslash}n")
GenSA call functions 3000 times.
> wstar.GenSA <- out.GenSA$par
> wstar.GenSA <- wstar.GenSA / sum(wstar.GenSA)
> rbind(tickers, round(100 * wstar.GenSA, 2))
        [,1]    [,2]    [,3]   [,4]    [,5]
tickers "GE"    "IBM"   "JPM"  "MSFT"  "WMT"
        "18.92" "21.23" "8.33" "15.92" "35.6"
> 100 * (sum(wstar.GenSA * mu) - mean(mu))
[1] 0.03790568876
```

GenSA found a minimum 0.1141484884 within 3000 function calls. Though both **GenSA** and **DEoptim** find good approximations to the global minimum of the portfolio objective function, the result of **GenSA** is closer to the global minimum with fewer function calls.

Thomson problem in physics

The objective of the Thomson problem is to find the global energy minimum of N equal point charges on a sphere (Thomson, 1904). The Thomson model has been widely studied in physics (Erber and Hockney, 1991; Altschuler et al., 1994; Morris et al., 1996; Xiang et al., 1997; Xiang and Gong, 2000a; Levin and Arenzon, 2003; Wales and Ulker, 2006; Altschuler and Pérez-Garrido, 2006; Ji-Sheng, 2007; Wales et al., 2009). In the Thomson model, the energy of N point charges constrained on a sphere is

$$E = \frac{1}{2} \sum_{j \neq i} \frac{1}{\left| \vec{r}_i - \vec{r}_j \right|}. \tag{5}$$

For the Thomson problem, the number of metastable structures grows exponentially with N (Erber and Hockney, 1991). Moreover, the region containing the global minimum is often small compared with those of other minima (Morris et al., 1996). This adds to the difficulty of the Thomson problem and in part explains why it has served as a benchmark for global optimization algorithms in a number of previous studies (Erber and Hockney, 1991; Xiang et al., 1997; Xiang and Gong, 2000a; Altschuler and Pérez-Garrido, 2006). The Thomson problem has been tackled by several methods such as steepest descent (Erber and Hockney, 1991), constrained global optimization (CGO) (Altschuler et al., 1994), generalized simulated annealing algorithm (Xiang et al., 1997; Xiang and Gong, 2000a), genetic algorithms (Morris et al., 1996), and variants of Monte Carlo (Wales and Ulker, 2006; Wales et al., 2009). Typically, deterministic methods such as steepest descent with multiple starts can provide a good solution when there are fewer point charges (Erber and Hockney, 1991). When N is large, there are an enormous number of local minima since it increases exponentially with N (Erber and Hockney, 1991). For large N, the stochastic methods such as generalized simulated annealing, genetic algorithms, and variants of Monte Carlo, can give reasonable results (Xiang et al., 1997; Xiang and Gong, 2000a; Morris et al., 1996; Wales and Ulker, 2006; Wales et al., 2009).

We can define an R function for the Thomson problem.

```
> Thomson.fn <- function(x) \{
+   fn.call <<- fn.call + 1
+   x <- matrix(x, ncol = 2)
+   y <- t(apply(x, 1, function(z)\{
+    c(sin(z[1]) * cos(z[2]),
+      sin(z[1]) * sin(z[2]), cos(z[1]))\}))
+   n <- nrow(x)
+   tmp <- matrix(NA, nrow = n, ncol = n)
+   index <- cbind(as.vector(row(tmp)), as.vector(col(tmp)))
+   index <- index[index[, 1] < index[, 2], , drop=F]
+   rdist <- apply(index, 1, function(z) \{
+     tmp <- 1/sqrt(sum((y[z[1], ] - y[z[2], ])^2))
+   \})
+   res <- sum(rdist)
+   return(res)
+ \}
```

In this example we chose a small number of point charges (12), since our purpose is only to show how to use **GenSA**. The global energy minimum of 12 equal point charges on the unit sphere is 49.16525306 with the corresponding configuration of an icosahedron (Erber and Hockney, 1991; Morris et al., 1996; Wales and Ulker, 2006).

We applied **DEoptim** with default settings to the Thomson problem.

```
> n.particles <- 12
> lower.T <- rep(0, 2 * n.particles)
> upper.T <- c(rep(pi, n.particles), rep(2 * pi, n.particles))
>
> options(digits = 10)
> set.seed(1234)
> sink("tmp.txt")
> fn.call <<- 0
> out.DEoptim <- DEoptim(fn = Thomson.fn, lower = lower.T, upper = upper.T)
> sink(NULL)
> fn.call.DEoptim <- fn.call
> out.DEoptim$optim[c(2, 3)]
$bestval
```

```
[1] 49.59590424

$nfeval
[1] 402

> cat("DEoptim call functions", fn.call.DEoptim, "times.{\textbackslash}n")
DEoptim call functions 48240 times.
```

DEoptim used 48240 function calls to reach the function value 49.59590424, which is still far from the global minimum 49.16525306. Then we used a local search method to further refine the best value given by DEoptim.

```
> out.DEoptim_BFGS <- optim(par = out.DEoptim$optim$bestmem, fn = Thomson.fn,
+                           lower = lower.T, upper = upper.T, method = "L-BFGS-B")
> out.DEoptim_BFGS[c("value")]
$value
[1] 49.16525309
```

L-BFGS-B refined the function value to 49.16525309, which is close to the global minimum 49.16525306.

We applied GenSA to the Thomson problem with the same number of function calls as that of DEoptim:

```
> set.seed(1234)
> fn.call <<- 0
> out.GenSA <- GenSA(par = NULL, lower = lower.T, upper = upper.T,
+                    fn = Thomson.fn, control = list(max.call = fn.call.DEoptim))
> out.GenSA[c("value", "counts")]
$value
[1] 49.16525306

$counts
[1] 48240

> cat("GenSA call functions", fn.call, "times.{\textbackslash}n")
GenSA call functions 48240 times.
```

We plotted the cumulative minimum of function value `Fmin` vs. number of function calls (`#fn.call`) for GenSA (green) and DEoptim (blue) in the Thomson problem in Figure 3. GenSA reached the global minimum (red asterisk) after 2791 function calls while DEoptim does not reach the global minimum after a much larger number of function calls (48240). Following the instruction for termination criteria in setion "The package GenSA", solving the Thomson problem with just 12 point charges is easy and we can stop GenSA much earlier by setting a smaller `max.call`.

For the Thomson problem, both DEoptim and GenSA converge to the global minimum. GenSA obtained more accurate results with fewer function calls than DEoptim.

GenSA relies on repeated evaluation of the objective function, so users who are interested in making GenSA run as fast as possible should ensure that evaluation of the objective function is also as efficient as possible.

Summary

Many R packages for solving optimization problems such as DEoptim and rgenoud have been developed and successfully used in the R community. Based on Tsallis statistics, an R package GenSA was developed for generalized simulated annealing. We propose GenSA be added to the tool kit for solving optimization problems in R. The package GenSA has proven to be a useful tool for solving global optimization problems. The applicability of GenSA was demonstrated with a standard test function, the Rastrigin function, a simple example of a stylized non-convex portfolio risk contribution allocation, and the Thomson problem in physics. As no single method is the best for all the optimization problems, GenSA can serve as a complementary tool to, rather than a replacement for, other widely used R packages for optimization. We hope that the availability of GenSA will allow users to have more choices when they process difficult objective functions. The results are based on GenSA version 1.1.3, DEoptim version 2.2.1, and R 2.15.2.

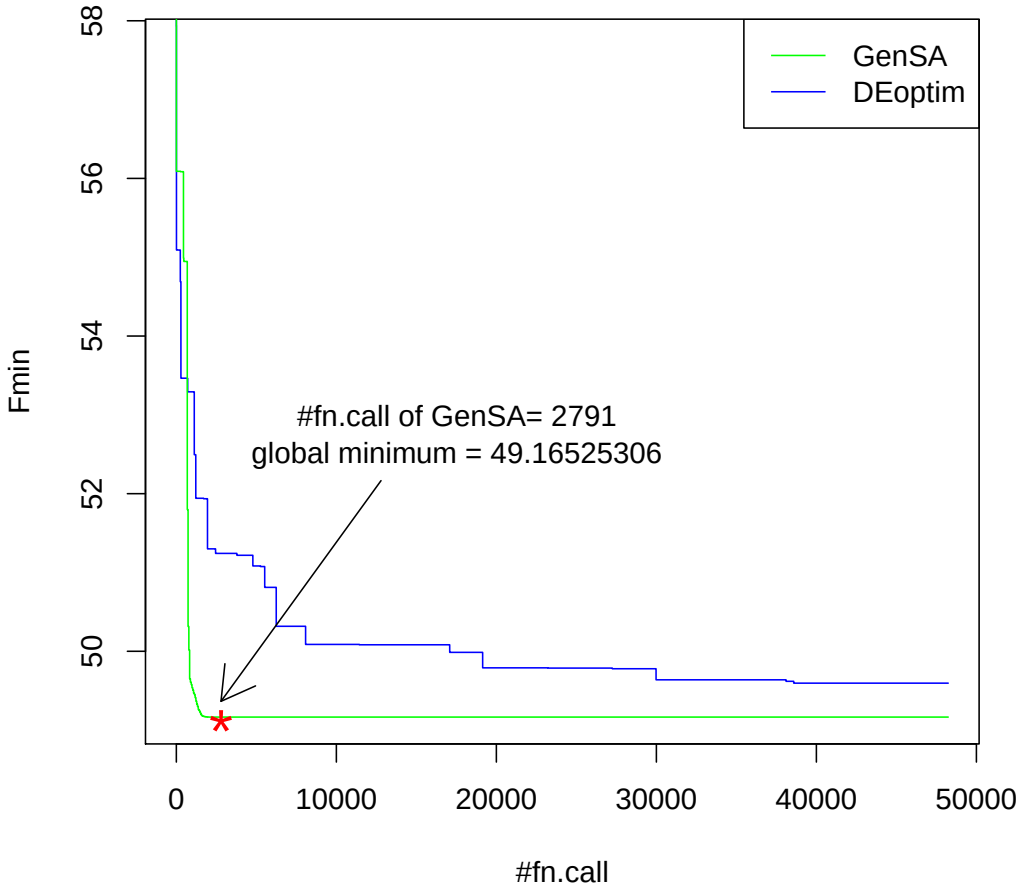

Figure 3: Cumulative minimum of function value `Fmin` vs. number of function calls (`#fn.call`) for **GenSA** (green) and **DEoptim** (blue) in the Thomson problem (number of point charges is 12). **GenSA** reaches the global minimum (red asterisk) after 2791 function calls while **DEoptim** does not reach the global minimum after a much larger number of function calls (48240).

Acknowledgements

This work would not be possible without the R open source software project. We would like to thank our colleague Florian Martin for inspirational discussions. We acknowledge our colleague Lynda Conroy-Schneider for editing our manuscript. We would also like to thank the reviewers for giving such constructive comments which substantially improved the quality of the manuscript.

Bibliography

F. P. Agostini, D. D. O. Soares-Pinto, M. A. Moret, C. Osthoff, and P. G. Pascutti. Generalized simulated annealing applied to protein folding studies. *Journal of Computational Chemistry*, 27:1142–1155, 2006. [p13]

E. Altschuler and A. Pérez-Garrido. Defect-free global minima in thomson's problem of charges on a sphere. *Physical Review E*, 73(3):036108, 2006. [p23]

E. L. Altschuler, T. J. Williams, E. R. Ratner, F. Dowla, and F. Wooten. Method of constrained global optimization. *Phys. Rev. Lett.*, 72:2671–2674, Apr 1994. doi: 10.1103/PhysRevLett.72.2671. URL http://link.aps.org/doi/10.1103/PhysRevLett.72.2671. [p23]

D. Ardia, K. Boudt, P. Carl, K. M. Mullen, and B. G. Peterson. Differential evolution with DEoptim. *The R Journal*, 3, 2011. [p14, 16, 21, 22]

R. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995. [p17]

J. O. Cerdeira, P. D. Silva, J. Cadima, and M. Minhoto. *subselect: Selecting variable subsets*, 2011. URL http://CRAN.R-project.org/package=subselect. R package version 0.11-1. [p14]

D. Cvijovic and J. Klinowski. Taboo search: An approach to the multiple minima problem. *Science*, 267:664–666, 1995. [p13]

D. Cvijovic and J. Klinowski. Taboo search: An approach to the multiple-minima problem for continuous functions. In R. H. Pardalos Pm, editor, *Handbook of Global Optimization*, volume 2, pages 387–406. Kluwer Academic Publisher, 2002. [p13]

T. Erber and G. M. Hockney. Equilibrium configurations of n equal charges on a sphere. *Journal of Physics A: Mathematical and General*, 24:L1369, 1991. [p23]

E. Figielska. A genetic algorithm and a simulated annealing algorithm combined with column generation technique for solving the problem of scheduling in the hybrid flowshop with additional resources. *Computers & Industrial Engineering*, 56(1):142–151, 2009. [p14]

D. Fouskakis and D. Draper. Stochastic optimization: a review. *International Statistical Review*, 70(3): 315–349, 2002. [p13]

F. Glover, E. Taillard, and E. Taillard. A user's guide to tabu search. *Annals of operations research*, 41(1): 1–28, 1993. [p13]

V. D. Guire, M. Caron, N. Scott, C. Menard, M. F. Gaumont-Leclerc, P. Chartrand, F. Major, and G. Ferbeyre. Designing small multiple-target artificial rnas. *Nucleic Acids Research*, 38(13):e140, 2010. [p13]

J. H. Holland. Adaptation in natural and artificial systems. *The University of Michigan Press, Ann Arbor*, 1975. [p13]

C. Ji-Sheng. Ground state energy of unitary fermion gas with the thomson problem approach. *Chinese Physics Letters*, 24:1825, 2007. [p23]

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220: 671–680, 1983. [p13, 14]

Y. Levin and J. Arenzon. Why charges go to the surface: A generalized thomson problem. *EPL (Europhysics Letters)*, 63:415, 2003. [p23]

W. Mebane Jr and J. Sekhon. Genetic optimization using derivatives: the rgenoud package for r. *Journal of Statistical Software*, 42(11):1–26, 2011. [p14, 16, 17, 18]

J. R. Morris, D. M. Deaven, and K. M. Ho. Genetic-algorithm energy minimization for point charges on a sphere. *Physical Review B*, 53:1740–1743, 1996. [p23]

K. Mullen, D. Ardia, D. 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. URL http://www.jstatsoft.org/v40/i06/. [p14, 16, 21]

L. Murphy. *likelihood: Methods for maximum likelihood estimation*, 2008. R package version 1.4. [p14]

J. C. Nash. *Compact numerical methods for computers: linear algebra and function minimisation*. Taylor & Francis, 1990. ISBN 978-0852743195. [p17]

S. Pavlidis, A. Payne, and S. Swift. Multi-membership gene regulation in pathway based microarray analysis. *Algorithms for Molecular Biology*, 6(1):22, 2011. ISSN 1748-7188. doi: 10.1186/1748-7188-6-22. URL http://www.almob.org/content/6/1/22. [p14]

M. J. D. Powell. The bobyqa algorithm for bound constrained optimization without derivatives. *Report No. DAMTP, 2009/NA06, Centre for Mathematical Sciences, University of Cambridge*, 2009. [p14, 17]

P. Serra, A. F. Stanton, and S. Kaisb. Comparison study of pivot methods for global optimization. *Journal of Chemical Physics*, 106:7170–7177, 1997. [p13, 14, 18]

P. Sólymos. dclone: Data cloning in R. *The R Journal*, 2(2):29–37, 2010. URL http://journal.r-project.org/. [p14]

R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997. [p13]

H. Szu and R. Hartley. Fast simulated annealing. *Physics Letters A*, 122:157–162, 1987. [p14]

S. Theussl. *CRAN Task View: Optimization and Mathematical Programming*, 04 2011. URL http://cran.r-project.org/web/views/Optimization.html. [p14]

D. Thompson and G. Bilbro. Comparison of a genetic algorithm with a simulated annealing algorithm for the design of an atm network. *Communications Letters, IEEE*, 4(8):267–269, 2000. [p14]

J. J. Thomson. On the structure of the atom: an investigation of the stability and periods of oscillation of a number of corpuscles arranged at equal intervals around the circumference of a circle; with application of the results to the theory of atomic structure. *Philosophical Magazine*, 7:237–265, 1904. [p23]

C. Tsallis and D. A. Stariolo. Generalized simulated annealing. *Physica A*, 233:395–406, 1996. [p14, 15]

J. VanDerWal and S. Januchowski. *ConsPlan: Conservation Planning Tools*, 2010. R package version 0.1. [p14]

D. Wales and S. Ulker. Structure and dynamics of spherical crystals characterized for the thomson problem. *Physical Review B*, 74(21):212101, 2006. [p23]

D. Wales, H. McKay, and E. Altschuler. Defect motifs for spherical topologies. *Physical Review B*, 79 (22):224115, 2009. [p23]

Y. Xiang and X. G. Gong. Efficiency of generalized simulated annealing. *Physical Review E*, 62:4473–4476, 2000a. [p13, 14, 15, 23]

Y. Xiang and X. G. Gong. Generalized simulated annealing method and its application. *Progress In Physics*, 20:319–334, 2000b. [p14]

Y. Xiang, D. Y. Sun, W. Fan, and X. G. Gong. Generalized simulated annealing algorithm and its application to the thomson model. *Physics Letters A*, 233:216–220, 1997. [p13, 14, 15, 23]

Y. Xiang, D. Y. Sun, and X. G. Gong. Generalized simulated annealing studies on structures and properties of ni_n (n=2-55) clusters. *Journal of Physical Chemistry A*, 104:2746–2751, 2000. [p13, 15]

X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, 1999. [p17]

C. Zhu, R. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997. [p17]

Yang Xiang
Philip Morris International R&D
Philip Morris Products S.A.
Quai Jeanrenaud 5
2000 Neuchatel
Switzerland
Yang.Xiang@pmi.com

Sylvain Gubian
Philip Morris International R&D
Philip Morris Products S.A.
Quai Jeanrenaud 5
2000 Neuchatel
Switzerland
Sylvain.Gubian@pmi.com

Brian Suomela
Philip Morris International R&D
Philip Morris Products S.A.
Quai Jeanrenaud 5
2000 Neuchatel
Switzerland
Brian.Suomela@pmi.com

Julia Hoeng
Philip Morris International R&D
Philip Morris Products S.A.
Quai Jeanrenaud 5
2000 Neuchatel
Switzerland
Julia.Hoeng@pmi.com