Journal of Statistical Software
December 2016, Volume 75, Issue 4. doi: 10.18637/jss.v075.i04

Memetic Algorithms with Local Search Chains in R: The Rmalschains Package

Christoph Bergmeir
Monash University

Daniel Molina
University of Cádiz

José M. Benítez
University of Granada

Abstract

Global optimization is an important field of research both in mathematics and computer sciences. It has applications in nearly all fields of modern science and engineering. Memetic algorithms are powerful problem solvers in the domain of continuous optimization, as they offer a trade-off between exploration of the search space using an evolutionary algorithm scheme, and focused exploitation of promising regions with a local search algorithm. In particular, we describe the memetic algorithms with local search chains (MA-LS-Chains) paradigm, and the R package **Rmalschains**, which implements them. MA-LS-Chains has proven to be effective compared to other algorithms, especially in high-dimensional problem solving. In an experimental study, we demonstrate the advantages of using **Rmalschains** for high-dimension optimization problems in comparison to other optimization methods already available in R.

Keywords: continuous optimization, memetic algorithms, MA-LS-Chains, R, **Rmalschains**.

1. Introduction

Global optimization, i.e., finding the inputs to a function that yield minimal/maximal output, is an important mathematical problem with applications in nearly all fields of modern Science and Engineering. Nowadays, as the functions to optimize are often complex and high-dimensional, mathematical analysis may be difficult, costly, or even impossible. In contrast, computational power is steadily growing, and optimization has evolved to an important line of research in Computer Sciences. Here, meta-heuristics are developed for general optimization to produce approximations to an exact solution with sufficiently good quality in a reasonable amount of time and with reasonable computational effort (Michalewicz and Fogel 2004). The algorithms treat the target function in a black-box manner, i.e., no preconditions or further information is required, such as continuity, differentiability, or derivatives of the function.

One such meta-heuristic, which led to a vast amount of successful algorithms and implementations in the past years, is the evolutionary algorithm (EA) framework (Bäck, Fogel, and Michalewicz 1997). Here, a population of possible solutions is evolved by altering (mutation) the solutions, and by letting them interact with each other (crossover). The candidate solutions are evaluated for their fitness, and newly created solutions replace the solutions with worst fitness, in order to converge around the region with best fitness. Note that in the context of EAs, the term solution is often used in the meaning of a candidate solution, i.e., a parameter configuration to be evaluated (Bäck *et al.* 1997). A solution is not necessarily a good or an optimal solution.

Using a population allows EAs to perform good exploration of the search space, but sometimes they are not capable of exploiting a promising region to reach the local optimum of that region. So, the solutions they obtain are sometimes not accurate.

Local search (LS) methods, on the other hand, can improve a solution very quickly, but they are not able to explore a complex search domain, as they find only local optima, i.e., solutions which are optimal in a certain region of the search space. Only for convex problems, LS methods are fully suitable, as in this case a local optimum is also a global optimum (see, e.g., Mullen 2014).

Memetic algorithms (MA; Moscato 1999; Krasnogor and Smith 2005) are a hybridization between EA and LS methods, with the objective to take advantage of both the exploration power of EAs and the exploitative power of the LS, therewith improving the overall results (Goldberg and Voessner 1999). MAs that integrate an LS method within the EA iteration can potentially obtain better results than applying a final LS method after the EA run, because the improvements obtained by the LS can guide better the search to best solutions, allowing that better solutions could be selected by the EA. As not all solutions are equally good, MAs can obtain best results if they apply the LS method with a higher *intensity*, i.e., using more fitness function evaluations (more iterations), to the most promising solutions, which are the ones with best fitness.

In this paper, we present the package **Rmalschains** (Bergmeir, Molina, and Benítez 2016) for R (R Core Team 2016) that implements various variants of the memetic algorithm with local search chains paradigm (MA-LS-Chains; Molina, Lozano, García-Martínez, and Herrera 2010). MA-LS-Chains is an MA whose main feature lies in its ability to apply the LS various times on the same solution, using every time a fixed amount of iterations/function evaluations. The final state of the LS parameters after each LS application becomes the initial point of a subsequent LS application over the same solution, creating an *LS chain*. This way, with varying length of the chain, i.e., varying amounts that the LS is called on an individual, MA-LS-Chains adapts the intensity of the LS to a solution in function of its quality.

The MA-LS-Chains algorithm family has proven to be very effective in continuous optimization problems in the past. MA-SW-Chains, which employs the Solis-Wets algorithm (SW) for LS, was the competition winner of CEC'2010 for high-dimensional optimization (Tang, Li, and Suganthan 2010). MA-CMA-Chains, which employs the covariance matrix adaptation evolution strategy (CMA-ES) as LS (see Section 2), performed very well in the BBOB'2009 competition (Hansen, Auger, Ros, Finck, and Pošík 2010), and also on the data of the CEC'2005 competition (Deb and Suganthan 2005; García, Molina, Lozano, and Herrera 2009); though it did not take part in the official competition, it was evaluated using the same conditions in Molina *et al.* (2010). In this package, both LS methods are available and the user can choose which one will be used (the default LS method is CMA-ES).

There is already a host of choices for continuous optimization methods that are readily available in R; Section 4 gives an overview. However, an important result in research on optimization is the existence of several "No Free Lunch" theorems, which "mean that if an algorithm does particularly well on average for one class of problems then it must do worse on average over the remaining problems" (Wolpert and Macready 1997). So, a method which takes into account problem specific knowledge has the potential to perform better than general methods. And though most optimization algorithms do not take into account problem-specific knowledge explicitly, they are usually implicitly better/worse suited for certain types of problems. Taking this into account together with the good performance of MA-LS-Chains, especially for high-dimensional problems, we find it justified to present another package for optimization to the R community. The **Rmalschains** package also performed well in a recent study by Burns (2012), and in Section 5, we perform a comparison of our package to other methods, with a focus on high-dimensional problems.

The algorithm is implemented in C++, and encapsulated in a library called **librealea** (Molina 2012), so that it can also be used outside of R. **Rmalschains** uses **Rcpp** (Eddelbuettel and François 2011) to make the functionality of **librealea** accessible from within R. The package **Rmalschains** is available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=Rmalschains`, and has a dedicated website at `http://sci2s.ugr.es/dicits/software/Rmalschains`. Also, the interested reader can find further information on the state of the art of EAs on the thematic web site of our research group on "Evolutionary Algorithms and other Metaheuristics for Continuous Optimization Problems" (`http://sci2s.ugr.es/EAMHCO/`).

The remainder of this paper is structured as follows. Section 2 presents the theory of the MA-LS-Chains algorithm, and Section 3 shows a brief example of the usage of the package. Section 4 gives an overview on the other packages available in R for continuous optimization, and Section 5 shows experiments comparing the methods already present in R with **Rmalschains**. Section 6 concludes the paper.

2. The theory of the algorithm

In the following, we describe briefly the general scheme of the MA-LS-Chains algorithm and its main components, i.e., the EA and the LS methods employed. For more details, the reader may refer to Molina *et al.* (2010).

2.1. General scheme

The algorithm was designed with the idea that the LS should be applied with higher intensity on the most promising regions. As promising regions, we consider the areas/regions where solutions with good fitness are located.

MA-LS-Chains is a steady-state MA that is combined with different methods for the LS. It uses a steady-state genetic algorithm (SSGA) as EA (Whitley 1989; Smith 2002). Different from a generational algorithm, where the genetic operators are applied to large parts of the population simultaneously, in a steady-state EA only single individuals are used at a time to generate offspring, which replaces other single individuals of the population.

Algorithm 1 Pseudocode of MA-LS-Chains.

```
1:  Generate the initial population
2:  while not termination-condition do
3:     Perform the SSGA with $n_{frec}$ evaluations.
4:     Build the set $S_{LS}$ of individuals which can be refined by LS.
5:     Pick the best individual $c_{LS}$ in $S_{LS}$.
6:     if $c_{LS}$ belongs to an existing LS chain then
7:        Initialize the LS operator with the LS state stored with $c_{LS}$.
8:     else
9:        Initialize the LS operator with the default LS parameters.
10:    end if
11:    Apply the LS algorithm to $c_{LS}$ with $I_{str}$, giving $c^r_{LS}$.
12:    Replace $c_{LS}$ by $c^r_{LS}$.
13:    Store the final LS state with $c^r_{LS}$.
14: end while
```

MA-LS-Chains allows for improving the same solution several times, thus creating an *LS chain*. Also, it uses a mechanism to store the final state of the LS parameters along with the solution, after each LS application. In this way, the final state of an LS application on a solution can be used for the initialization of a subsequent LS application on the same solution, *continuing* the LS.

The general algorithm is shown in Algorithm 1. After generating the initial population, in a loop the following is executed: The SSGA is run with a certain amount of evaluations n_{frec}. Then, the set S_{LS} is built with the individuals of the population that have never been improved by the LS, or that have been improved by the LS but with an improvement (in fitness) superior to $\delta^{\min}_{LS}$, where $\delta^{\min}_{LS}$ is a parameter of the algorithm (by default $\delta^{\min}_{LS} = 10^{-8}$). If $|S_{LS}| \neq 0$, the LS is applied with an intensity of I_{str} to the best individual in S_{LS}. If S_{LS} is empty, the whole population is reinitialized except for the best individual which is maintained in the population.

With this mechanism, if the SSGA obtains a new best solution, it should be improved by the LS in the following application of the LS method.

2.2. The evolutionary algorithm

In MA-LS-Chains, the SSGA applied is specifically designed to promote high population diversity levels by means of the combination of the $BLX - \alpha$ crossover operator (Eshelman and Schaffer 1993) with a high value for its associated parameter (we use a default of $\alpha = 0.5$) and the negative assortative mating (NAM) strategy (Fernandes and Rosa 2001). Diversity is favored as well by means of the BGA mutation operator. The replacement strategy used is replacement worst (RW) (Goldberg and Deb 1991). The combination NAM-RW produces a high selective pressure.

Crossover. The $BLX - \alpha$ operator (Eshelman and Schaffer 1993) performs crossover in the following way. Let $a, b \in \mathbb{R}$ be the respective numbers at the ith position of two individuals. Without loss of generality, we assume $a < b$. Using the distance $d = b - a$, the outcome z of the crossover operation is a random number chosen uniformly from the interval $[a - d \cdot \alpha,\ b + d \cdot \alpha]$. It can be shown (Nomura and Shimohara 2001) that values of $\alpha > \frac{\sqrt{3}-1}{2} \approx 0.366$ yield a spread of the individuals in the distribution, whereas smaller values of α lead to a concentration of the individuals.

Negative assortative mating. Assortative mating means that the individuals which are crossed are not chosen fully at random, but depending on their similarity. According to whether crossing of similar or dissimilar individuals is favored, the strategy is called positive or negative assortative mating. We use a mating strategy proposed by Fernandes and Rosa (2001), which favors diversity in the population. The algorithm chooses 4 individuals, and computes the similarity (in form of the Euclidean distance) between the first one and all others. Then, the first individual and the individual with maximal distance from it are chosen for mating.

Mutation: The *BGA* operator. This is the operator of the breeder genetic algorithm (BGA; Mühlenbein and Schlierkamp-Voosen 1993). Its main purpose is to assure diversity in the population. Let $c \in [a,\ b]$ be the value at the ith position of the individual subject to mutation, with $a, b \in \mathbb{R}$ being the corresponding upper and lower domain bounds. Let r be the mutation range, normally defined as $0.1 \cdot (b - a)$. Then, a new value c' for the ith position of the chromosome, lying in the interval $[c - r,\ c + r]$, is computed in the following way:

$$c' = c \pm r \cdot \sum_{k=0}^{15} \alpha_k 2^{-k},$$

where addition or subtraction are chosen with a probability of 0.5, and the α_k are chosen as either zero or one, with a probability for one of $\mathsf{P}(\alpha_k = 1) = \frac{1}{16}$. So, the probability of generating a c' in the neighborhood of c is very high.

The replacement worst strategy. This is a standard replacement strategy, where the worst individuals are replaced by better ones. It generates high selective pressure, so that in combination with the negative assortative mating, many different solutions are generated throughout the search, but only the best ones are kept in the population.

2.3. The local search method

Within the MA-LS-Chains paradigm, different methods for the LS can be used, depending on the application. Usually, the CMA-ES strategy works best. But as the CMA-ES algorithm does not scale well with the amount of parameters, for high-dimensional problems other LS strategies, such as the Solis-Wets or the Subgrouping Solis-Wets solver are to be preferred (Molina, Lozano, Sánchez, and Herrera 2011).

CMA-ES. The CMA-ES algorithm (Hansen, Müller, and Koumoutsakos 2003) can be considered the state of the art in continuous optimization. Thanks to the adaptability of its parameters, its convergence is very fast and obtains very good results. Implementations are available in R in packages **cmaes** (Trautmann, Mersmann, and Arnu 2011), **adagio** (Borchers

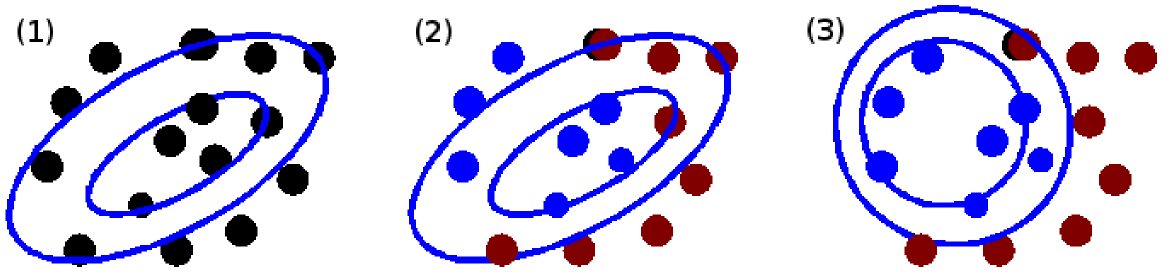


Figure 1: Example of convergence of the CMA-ES algorithm. (1) Solutions are generated from the distribution function. (2) The solutions are evaluated and a subset of best solutions is created (blue dots). (3) The distribution function is adapted accordingly.

2016), and **parma** (Ghalanos and Pfaff 2016). **Rmalschains** includes a C implementation from the original author's web page (`https://www.lri.fr/~hansen/`). CMA-ES is an algorithm that uses a distribution function (adapted Gaussian curve) to obtain new solutions, and adapt the distribution around the best created solutions. The global scheme can be observed in Figure 1.

Its only parameters are the initial average of the distribution $\vec{m}$ and the initial σ. MA-CMA-Chains sets the individual to optimize c_{LS} as $\vec{m}$, and as the initial σ value the half of the distance of c_{LS} to its nearest neighbor in the EA's population.

Solis-Wets algorithm. The algorithm presented by Solis and Wets (1981) is a randomized hill climber with adaptive step size. Starting from the current position c_{LS} in the search space, two candidate solutions are generated in the following way. Using a multivariate normal distribution that has the same dimension as c_{LS} and a standard deviation of ρ, a sample is drawn and used as distance d to compute the candidate solutions $c_{LS} + d$ and $c_{LS} - d$. If the better one of the candidate solutions is better than c_{LS}, c_{LS} is updated to this new solution and a success is recorded. If c_{LS} is better than both of the candidate solutions, c_{LS} is not updated and a failure is recorded. After several successes/failures in a row, ρ is increased/decreased. Furthermore, there is a bias term added, to put the search momentum to regions that are promising. This term is continuously updated using its previous value and d. For details, see Molina *et al.* (2010).

Though this algorithm is rather unsophisticated, it usually yields good results, is fast and easy to compute, scalable, and does not need derivatives to be computed. This makes it suitable to be used in the MA-LS-Chains framework.

Subgrouping Solis-Wets. In this adaptation to high-dimensional data of the Solis-Wets algorithm (Molina *et al.* 2011), a subgroup of the overall amount of parameters is chosen randomly, and then optimized for a certain amount of evaluations (defined by the parameter `maxEvalSubset`). Then, a new subset is chosen. In the current implementation, the subsets contain 20% of the overall amount of variables.

Nelder-Mead downhill simplex. This method, presented by Nelder and Mead (1965) (see also Nelder and Singer 2009, or, e.g., Press, Teukolsky, Vetterling, and Flannery 2007), is a popular standard algorithm for optimization without using derivatives. In R, it is the

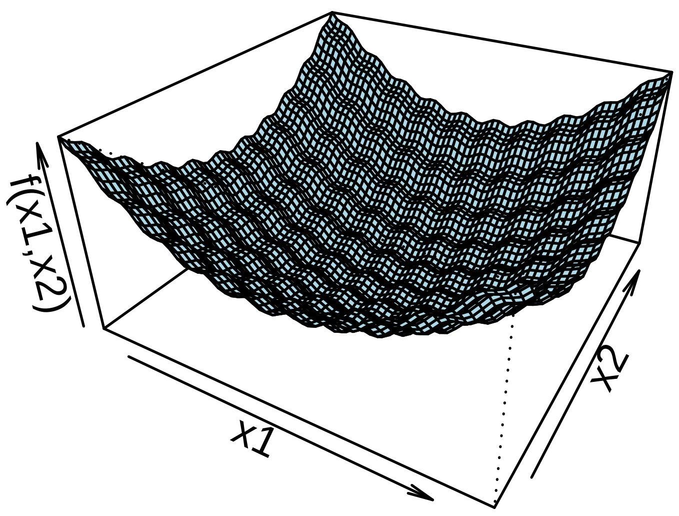


Figure 2: The 2-dimensional Rastrigin function, in the $[-5.12, 5.12]$ interval of the two input parameters.

standard method of the `optim` function (Venables and Ripley 2002). Also, it is implemented in the packages **neldermead** (Bihorel and Baudin 2015), **dfoptim** (Varadhan and Borchers 2016), **gsl** (Hankin 2006), **adagio** (Borchers 2016), and **nloptr** (Ypma, Borchers, and Eddelbuettel 2014). In this algorithm, a simplex is initialized in an n-dimensional parameter space using $n+1$ candidate solutions. Then, the simplex is evolved according to the fitness of its vertices. In MA-LS-Chains, the simplex is initialized with c_{LS} and n further candidate solutions which are generated as $c_{LS} + \lambda_i e_i$, $i = 1, \ldots, n$, with the e_i being unit vectors and λ_i a scaling factor for the respective parameter dimension, which is usually set to one. Then, within the chaining mechanism, the simplex is stored and reloaded with each individual.

3. A simple example

We use minimization of the n-dimensional Rastrigin function as an example, which is a common benchmark in global optimization (Mühlenbein, Schomisch, and Born 1991). The function is defined as follows:

$$f(\vec{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) \right).$$

It has a global minimum at $f(\vec{0}) = 0$. The cosine causes a lot of local optima, so that this function is considered a difficult optimization problem. In R, it can be implemented as follows:

```
R> rastrigin <- function(x) 10 * length(x) + sum(x^2 - 10 * cos(2 * pi * x))
```

Figure 2 shows the 2-dimensional case for the input parameters in the $[-5.12, 5.12]$-interval. The `malschains` function can then be used to minimize the function (for maximization, the objective function would have to be inverted). For the 30-dimensional Rastrigin function, the optimization is performed, e.g., with the following command:

```
R> res <- malschains(rastrigin, lower = rep(-5.12, 30),
+    upper = rep(5.12, 30), maxEvals = 200000, verbosity = 0,
+    control = malschains.control(popsize = 50, istep = 300, ls = "cmaes",
+    optimum = 0))
```

Here, the first parameter is the objective function, the parameters `lower` and `upper` define the lower and upper bounds of the search space, and also define the amount of parameters that the optimizer assumes the objective function to have. The parameter `maxEvals` defines the maximal number of function evaluations that are to be used. The parameter `verbosity` controls the verbosity level of the output, where 0 indicates no output, 1 indicates a summary, and values of 2 and above currently indicate full output. Finally, the `control` parameter can be used to define parameters of the optimization method itself. In the example, we use the parameter `popsize` to set the population size of the EA, the parameter `istep` to set the amount of function evaluations within each run of the LS, and the parameter `ls`, to set the type of LS to use.

Furthermore, the parameter `optimum` gives the global optimum of the function that the algorithm will try to achieve.

The solution is a list (an object of class '`malschains`') containing the best individual `sol`, and its fitness. Furthermore, it contains some information about the convergence process, such as the total amount of evaluations spent for the EA and the LS, respectively, the ratio of the spent evaluations (also called effort), the ratio of total improvement of the fitness, the percentage of times that application of the EA/LS yielded improvement, and some timing results in milliseconds:

```
R> res

NumTotalEvalEA: 36000
NumTotalEvalLS: 35112
RatioEffort EA/LS: [51/49]
RatioImprovement EA/LS: [31/69]
PercentageNumImprovement[EA]: 28%
PercentageNumImprovement[LS]: 90%
Time[EA]: 230.88
Time[LS]: 1143.27
Time[MA]: 1375.44
RatioTime[EA/MA]: 16.79
RatioTime[LS/MA]: 83.12
Fitness:
[1] 9.218184e-09
Solution:
 [1]  1.201915e-07  3.748141e-07 -3.306534e-07  1.122996e-06  2.451434e-06
 [6]  5.482329e-07 -2.016876e-06  2.047226e-06 -2.086381e-06 -1.026613e-06
[11]  1.670471e-06  6.020225e-07 -1.228457e-06  3.205899e-07  1.165649e-07
[16] -6.457484e-07  8.226205e-07 -5.470387e-07 -2.166945e-06 -1.685302e-06
[21]  1.264090e-06 -1.525650e-06  1.747982e-06 -2.998119e-07  9.149707e-07
[26] -3.178070e-07 -1.226612e-06 -2.350812e-07  9.139988e-07  9.228283e-07
```

It can be seen that the solution is near to the global optimum of zero both for all values of the solution and the fitness.

4. Other packages in R for continuous optimization

Throughout the last years, a rich variety of packages in R for optimization has been developed. A constantly updated overview is provided at the "CRAN Task View: Optimization and Mathematical Programming" (Theussl and Borchers 2016). In the following, we present methods from the section "General Purpose Continuous Solvers" of that task view, in which our package is also present. Some packages are omitted in the following overview because they are not applicable in our context or because they are very similar to other packages. Some of the available non-population-based methods are:

- The `optim` function (Venables and Ripley 2002) from the **stats** package is the standard optimization function in R. It implements a number of LS methods, like the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method, a box-constrained version of it, called L-BFGS-B, the Nelder-Mead method, a conjugate-gradient algorithm, and a simulated annealing method. The methods, though still pretty popular, cannot be seen as state of the art by today's standards, and attempts exist to replace these methods with newer ones (Nash 2014). Also, the first three of the methods are LS methods suitable for convex optimization only, and as one of the reviewers brought to our attention, the simulated annealing implementation has obvious shortcomings, and it also was not competitive in a recent comparison (Mullen 2014).

- The package **dfoptim** (Varadhan and Borchers 2016) implements derivative-free optimization algorithms. Concretely, the Nelder-Mead and the Hooke-Jeeves algorithms are implemented. The Nelder-Mead algorithm can be used within our package as the LS.

- The package **adagio** (Borchers 2016) implements several derivative-free global optimization algorithms, such as a scalable Nelder-Mead algorithm, CMA-ES, and Wolfe Line Search.

- **Rsolnp** (Ghalanos and Theussl 2016) and **alabama** (Varadhan 2015) have algorithms for objective functions with constraints. In particular, they use nonlinear augmented Lagrange multiplier method solvers, based on sequential quadratic programming (SQP), for optimization.

- The package **GenSA** (Xiang, Gubian, Suomela, and Hoeng 2013) implements generalized simulated annealing (Tsallis and Stariolo 1996).

- The packages **cmaes** (Trautmann *et al.* 2011) and **parma** (Ghalanos and Pfaff 2016) implement the CMA-ES algorithm, which we presented in Section 2.3 and use as an LS algorithm. **cmaes** implements a basic version of CMA-ES (lacking sophisticated convergence detection), while **parma**, oriented to optimization in financial portfolios, implements the most advanced version of CMA-ES (a direct translation of version 3.60 of the original author's code, Hansen 2012, implemented in MATLAB, The MathWorks Inc. 2014).

The available population-based methods include:

- The packages **DEoptim**, and **RcppDE** (Mullen, Ardia, Gil, Windover, and Cline 2011; Ardia, Boudt, Carl, Mullen, and Peterson 2011) implement differential evolution (DE), an EA that uses difference vectors of the individuals for crossover and mutation (Price, Storn, and Lampinen 2005). **RcppDE** is a reimplementation of **DEoptim** in C++ and yields the same results in terms of accuracy.
- The package **rgenoud** (Mebane Jr. and Sekhon 2011) implements a genetic algorithm that is also able to use an LS algorithm for improvement of single individuals. So, it can be considered a memetic algorithm. The LS employed is the BFGS algorithm. It is applied after each iteration to the individual with the best fitness or used as a genetic operator.
- The packages **PSO** (Bendtsen 2012) and **NMOF** (Schumann 2016) both implement particle swarm optimization (PSO). **PSO** implements the Standard PSO 2007 (SPSO-07, `http://www.particleswarm.info/Programs.html`).
- The package **nloptr** (Ypma *et al.* 2014) is an R interface to the popular library for optimization **NLopt** (Johnson 2012). **NLopt** implements a host of optimization methods, most of them also available by other R packages. We focus on the controlled random search (CRS) algorithm (and in particular, the CRS2 variant) with the local mutation modification. This algorithm starts with a random population of points, and randomly evolves these points by heuristic rules.

5. Experimental study: Comparison with other algorithms

In this section, we compare **Rmalschains** with the packages discussed in Section 4. The comparison is performed using a benchmark which contains 19 scalable objective functions with different characteristics. Our analysis considers accuracy and execution time of the methods with a fixed amount of function evaluations. Accuracy measures directly the quality of the solution, and may be considered the primary criterion to assess performance of a method. But execution time may be critical as well, in the sense that application of many methods is not feasible if problem dimension and complexity grow. Experiments are performed in different use cases, with medium-, and high-dimensional data, to analyze the behavior of the methods in detail, especially regarding the high-dimensional use case. Results are presented as boxplots, tabulars, and diagrams, showing accuracy, execution time, and ranking of average accuracy.

5.1. Test suite and experimental conditions

We use the well-known benchmark of Lozano, Molina, and Herrera (2011), which is especially good to test the scalability of the algorithms. This test set is composed of 19 scalable function optimization problems (also see `http://sci2s.ugr.es/eamhco/testfunctions-SOCO.pdf`):

- 6 Functions: F_1–F_6 of the CEC'2008 test suite. A detailed description may be found in Tang (2008).

Function	Name	Range	Global minimum
F_1	Shifted Sphere Function	$[-100, 100]^D$	-450
F_2	Shifted Schwefel's Problem 2.21	$[-100, 100]^D$	-450
F_3	Shifted Rosenbrock's Function	$[-100, 100]^D$	390
F_4	Shifted Rastrigin's Function	$[-5, 5]^D$	-330
F_5	Shifted Griewank's Function	$[-600, 600]^D$	-180
F_6	Shifted Ackley's Function	$[-32, 32]^D$	-140
F_7	Shifted Schwefel's Problem 2.22	$[-10, 10]^D$	0
F_8	Shifted Schwefel's Problem 1.2	$[-65.536, 65.536]^D$	0
F_9	Shifted Extended f10	$[-100, 100]^D$	0
F_{10}	Shifted Bohachevsky	$[-15, 15]^D$	0
F_{11}	Shifted Schaffer	$[-100, 100]^D$	0

Table 1: Range and global minima of the benchmark functions F_1–F_{11}.

Function	Unimodal/ multimodal	Shifted	Separable	Easily optimized dimension by dimension
F_1	U	Y	Y	Y
F_2	U	Y	N	N
F_3	M	Y	N	Y
F_4	M	Y	Y	Y
F_5	M	Y	N	N
F_6	M	Y	Y	Y
F_7	U	Y	Y	Y
F_8	U	Y	N	N
F_9	U	Y	N	Y
F_{10}	U	Y	N	N
F_{11}	U	Y	N	Y

Table 2: Characteristics of the benchmark functions F_1–F_{11}.

- 5 Shifted Functions: Schwefel's Problem 2.22 (F_7), Schwefel's Problem 1.2 (F_8), Extended f10 (F_9), Bohachevsky (F_{10}), and Schaffer (F_{11}).
- 8 Hybrid Composition Functions (F_{12}–F_{19}): These functions are built by combining two functions belonging to the set of functions F_1–F_{11}.

All the functions are minimization problems. Table 1 shows range and global minima of F_1–F_{11}, and Table 2 shows some further characteristics. F_{12}–F_{19} all have a global minimum at 0, and are non-separable.

In the comparison, we follow the guideline proposed by the authors of the benchmark. We apply the test suite for dimensions 2, 10, 30, 50, 100, 200, 500, and 1000. We consider the cases with dimensions 2, 10, 30, and 50 as low-dimensional problems, the cases with dimensions 100 and 200 as medium-dimensional problems, and the cases with dimensions 500 and 1000 as high-dimensional problems. Each algorithm is run 25 times for each test function, and the average error, with respect to the known global optimum, is obtained. Each run stops when a maximum of calls to the fitness function is reached, named *MaxFES*, depending on the dimension D in the following way: $MaxFES = 5000 \cdot D$.

To use *MaxFES* instead of a maximum iteration number allows us to make a fair comparison between optimization methods that have a very different structure. Unfortunately, for several of the considered packages (**rgenoud**, **cmaes**, **DEoptim**, **RcppDE**), only the maximum of iterations can be defined, but not a *MaxFES*. In these cases, we count the number of fitness evaluations, and we return the best solution after the first *MaxFES* evaluations. Then, we use the `callCC` function from the **base** package to stop the algorithm.

The experiments are performed on a Sun Grid Engine (SGE) cluster. Parallelization of the experimental executions is performed in the way that the different algorithms are run sequentially on the benchmark functions, and execution of different algorithms is parallelized. The nodes of the cluster have each an Intel Core i7 CPU with a frequency of 2.80 GHz, and 24 GB of RAM. We establish the following limits: One R session (one algorithm executed with a particular dimension on all benchmarks 25 times) is limited to 6 GB of RAM, and a maximal global execution time of 10 days. This is approximately 30 minutes for every execution of one algorithm, and seems reasonable taking into account that computation of the fitness within the benchmark functions is considerably less expensive than in usual real-world problems.

5.2. Parameters and used methods

The aim of this work is to make a comparison between different packages on CRAN that can be used for optimization. In order to simulate the most general use of the packages, we compare the results using the packages in the most simple and straightforward way, which is with their default parameters.

The only parameter that we define for the benchmark is the population size, as usually this parameter should be adapted in dependence of the dimensionality, and maintaining constant this parameter for all dimensions could result in degeneration of the methods' errors. According to the authors of the test suite (Lozano *et al.* 2011), this parameter is set to $\min(10 \cdot D, 100)$ for algorithms involving a population like DE and PSO.

From package **adagio**, we use the Nelder-Mead algorithm, denoted as `adagio_NM`, and from package **dfoptim** we use the box-constrained Hooke-Jeeves algorithm, which will be denoted as `dfoptim_HJKB`. Furthermore, we use the CMA-ES implementation available in package **parma**, denoted as `parma_CMAES`, and the CRS2 method available from package **nloptr**, denoted as `nloptr_CRS2`. The PSO algorithm available from package **NMOF** will be denoted as `NMOF_PSO` in the following.

Our function `malschains` is used with the local search methods CMA-ES (the default method), and SW (the Solis-Wets solver), because it is a more adequate method for higher dimensional problems, due to its lower computation time. These methods will be called in the following `malschains-CMA`, and `malschains-SW`.

All other applied methods have the same name as their respective packages/functions, namely **DEoptim**, **RcppDE**, and **PSO**.

From the packages in Section 4, the following are not finally used in the comparisons:

- As the methods in `optim` are either LS methods or are not competitive, we follow the suggestions of one of the reviewers and do not consider these methods in our study.

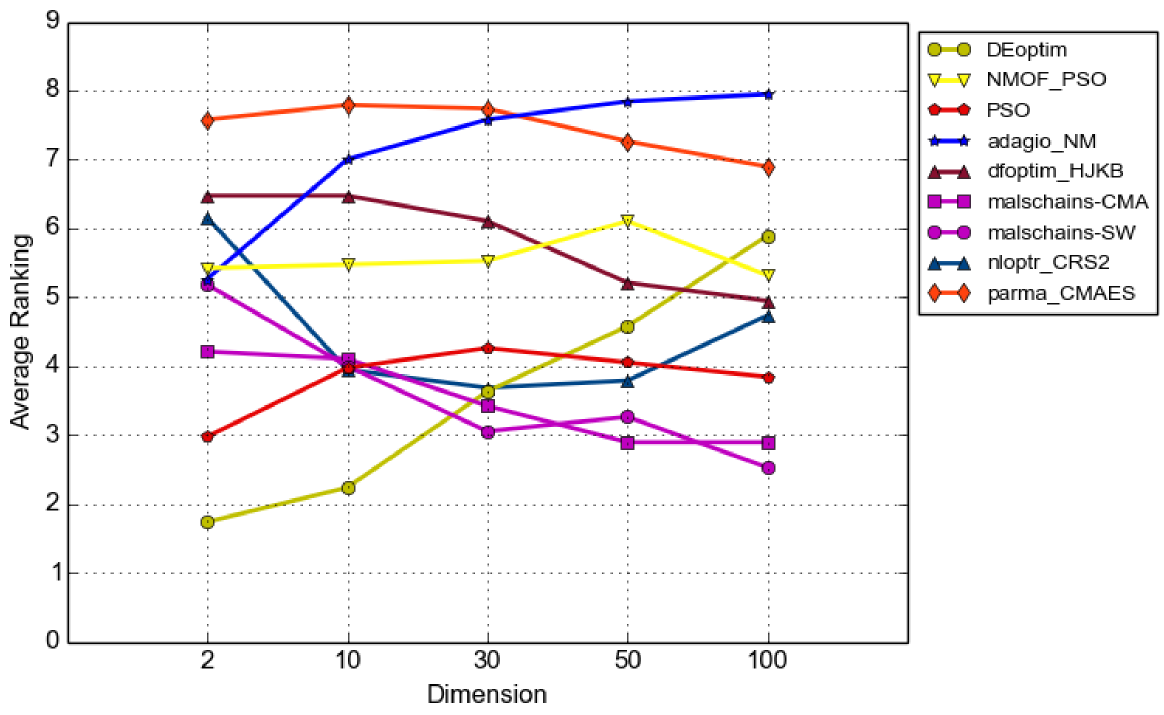


Figure 3: Average ranking for algorithms and dimension ≤ 100.

- The CMA-ES implementations available in packages **cmaes** and **adagio** are not used, as the implementation in the **parma** package is more sophisticated, because it is a direct translation of the code recommended by the original authors as "productive code."
- The packages **Rsolnp** and **alabama** are not included, because our focus is not to solve problems with (nonlinear) constraints. We used them in preliminary experiments, but not surprisingly, their performance was not good on our benchmark problems.
- Finally, the packages **rgenoud** and **GenSA** are not included, because their memory requirements prevented obtaining results for dimensions greater than 30.

Some packages require an initial solution, for these algorithms we generate a solution randomly within the parameter domains (using `runif`), and pass this initial solution to these functions.

5.3. Results in average error

In this section, we study the average error for medium dimension ($D \leq 100$). Results with higher dimensions (up to 1000) are analyzed in Section 5.5. For each function we rank the algorithms according to the average error (the tabulated results can be found in Appendix A). The algorithms with best results have the lowest ranks, i.e., the lower the ranking, the better the algorithm.

Figure 3 shows the average ranking for all the algorithms considered for dimension ≤ 100. An interesting conclusion we can draw from the figure is that **DEoptim** (and with the same results **RcppDE**) is initially the algorithm with best results, but with increasing dimensionality, the

Algorithm	Dimension			
	5	10	30	50
`adagio_NM`	68.06	254.56	13954.30	31057.45
DEoptim	402.30	770.45	2727.22	5138.34
RcppDE	287.83	322.06	1044.56	2515.88
`nloptr_CRS2`	322.90	413.01	2450.26	6668.90
`parma_CMAES`	844.36	2481.49	11397.93	22843.63
`dfoptim_HJKB`	9.51	22.26	59.07	100.07
`malschains-CMA`	44.85	137.69	888.95	7188.50
`malschains-SW`	29.14	108.08	440.32	1085.85
`PSO`	1200.48	1427.22	2002.28	2611.18
`NMOF_PSO`	489.87	1041.79	1858.67	2427.66
	100	200	500	1000
`adagio_NM`	123799.00	679064.70	–T–	–
DEoptim	12972.36	37580.78	177020.90	656181.60
RcppDE	4917.35	14383.89	85628.93	361631.10
`nloptr_CRS2`	29349.60	140109.90	–T–	–
`parma_CMAES`	89434.72	–T–	–	–
`dfoptim_HJKB`	592.74	1809.29	4615.43	58617.32
`malschains-CMA`	47237.20	352899.50	–T–	–
`malschains-SW`	5693.48	17961.84	121082.20	570921.00
`PSO`	3934.63	6655.85	15833.53	35383.74
`NMOF_PSO`	3686.72	6498.25	12251.77	26849.33

Table 3: Time (in ms) for each optimization package. T: time limit was reached.

algorithm performs worse. With respect to our package, we can observe that the **Rmalschains** methods perform best for dimensions 50 and 100. **PSO** and `parma_CMAES` also perform well. Results obtained by package **PSO** are better than the ones obtained by `NMOF_PSO`.

5.4. Analysis of computation time

Though computation time depends greatly on implementation details (e.g., if the whole algorithm is implemented in pure R, or if C or C++ code is used), from a user perspective, when a package has to be chosen for a concrete application, such an analysis can be very valuable.

For each package and function we run once the optimization function and we measure its computation time (using the **microbenchmark** package; Mersmann 2015) removing every I/O operation (by function `capture.output` from the **utils** package). Table 3 shows the average computation time in milliseconds. This is graphically illustrated in Figure 4 (note that in the analysis of computation time **DEoptim** and **RcppDE** are shown separately).

Figure 4 shows that `dfoptim_HJKB` is the fastest, and `parma_CMAES` and `adagio_NM` are the slowest methods for higher dimensions. **PSO** has an average computation time, and it is the algorithm whose running time increases the least with the dimension.

The `malschains-CMA` algorithm has the same increasing ratio as `parma_CMAES` but it maintains for dimension ≤ 100 a moderate computation time. The `malschains-SW` method has better performance.

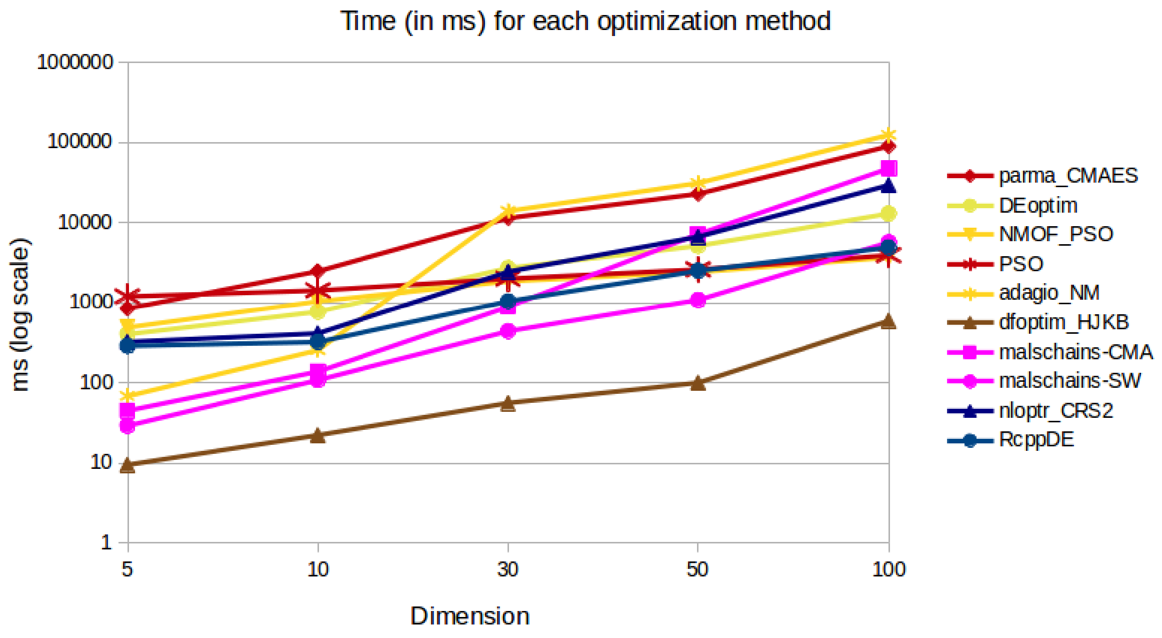


Figure 4: Average computation time for each package (in ms, log scale).

5.5. Scalability

Many real problems require optimization of large numbers of variables. Thus, an important aspect of an algorithm is its scalability. Unfortunately, many algorithms have serious limitations in the number of dimensions they can handle. The main reason is usually the increasing computation time (as seen in Section 5.4), but there are others, such as memory requirements, program errors, or accumulated error with the dimension.

In this section, we study the scalability of the different packages. First, we identify the packages that have scalability problems, considering Table 3:

- Method `parma_CMAES` with its default parameters is not very scalable, since the computational complexity of the CMA-ES algorithm is $O(n^3)$. From dimension 10 on it is the slowest algorithm, and reaches the time limit at dimension 200.
- Package **Rmalschains** with the CMA-ES method requires a lower computation time than `parma_CMAES`, but with a similar increasing velocity.
- The most scalable algorithms are `dfoptim_HJKB`, `PSO`, `NMOF_PSO`, `malschains-SW`, **DEoptim**, and **RcppDE**.

In terms of accuracy, Figure 5 shows the ranking for the algorithms that could be executed up to dimension 1000 (the tabulated results can be found in Appendix A). We can observe that `malschains-SW` obtains the best results for high-dimensional problems.

Results for the execution time are shown in Figure 6. We can observe that the differences in time between the majority of algorithms, except **PSO** and `dfoptim_HJKB`, are very similar. `dfoptim_HJKB` is the algorithm with lowest computation time, but taking into account its

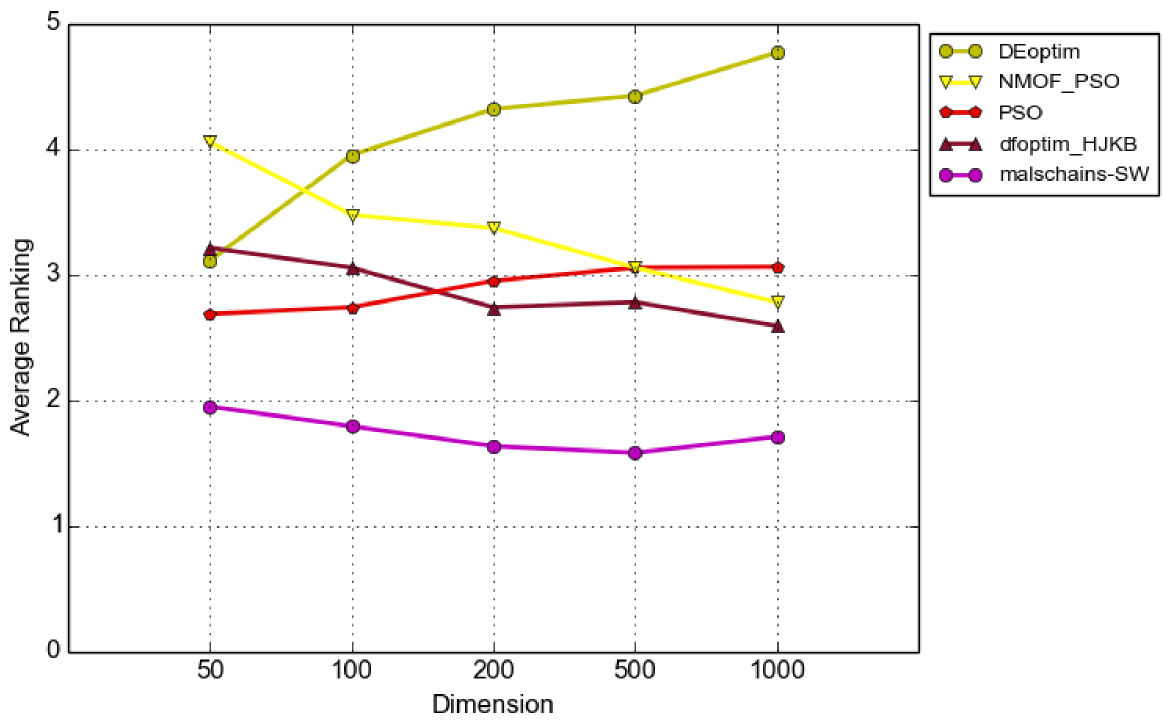


Figure 5: Average ranking for scalable algorithms and dimension ≤ 1000.

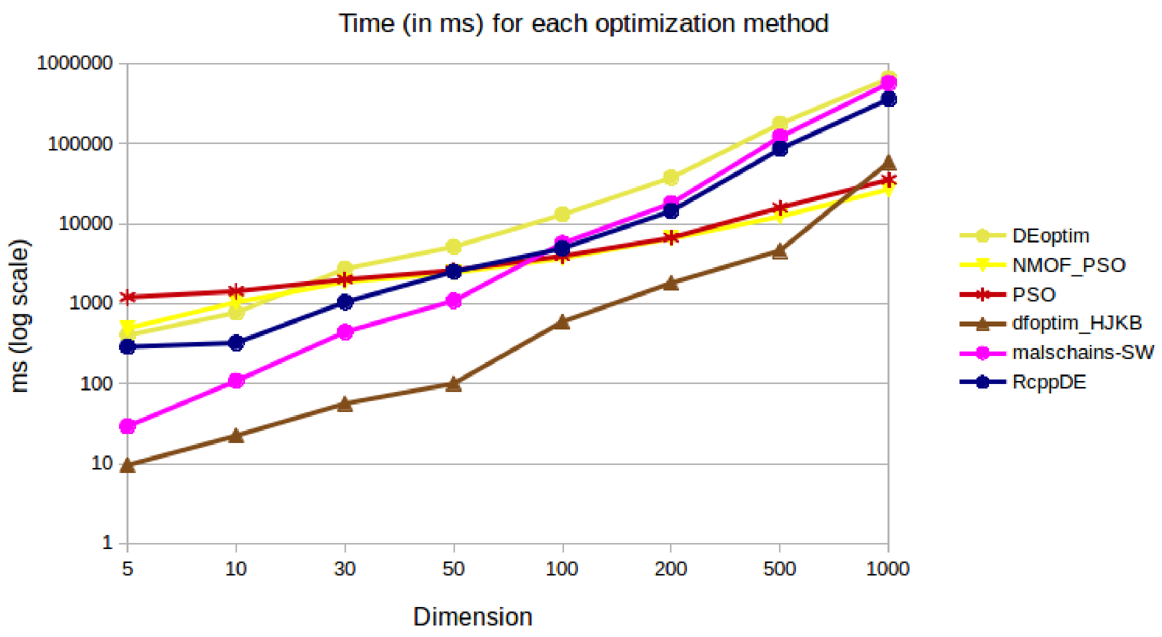


Figure 6: Average computation time for large dimension (in ms, log scale).

fast increase in computation time for dimension 1000, it may perform similar as the other methods for dimensions higher than 1000. **PSO** is the algorithm that maintains the lowest increase in computation time with the dimension.

5.6. Study of accuracy per function and dimension

So far we performed graphical analyses of the average results in ranking. In this section, we complement the analysis showing the results in fitness for several functions, using boxplots. As this analysis is pretty verbose, we show only selected results here, namely for the two functions F_4, Rastrigin's function, and F_{18}, as an example of combined functions. The full results can be found at `http://sci2s.ugr.es/dicits/software/Rmalschains`.

In Figures 7 and 8 we can see the results for function F_4 in medium and high dimensions. We can observe that there is a big difference between algorithms. Initially, `malschains-SW` and `malschains-CMA` obtain bad results in lower dimensions, but when the dimensionality increases, their results are more competitive: In dimension 100, only `parma_CMAES` and `dfoptim_HJKB` perform better on average, and for higher dimensions, only `dfoptim_HJKB` obtains better results.

In Figures 9 and 10 we can see the results for function F_{18} in medium and high dimensions. In that function, `malschains-SW` obtains better results than `dfoptim_HJKB`, and both of them maintain the best results. We can observe that the algorithms that maintain the best results also have low variance in the reached fitness, so that they robustly find better solutions than the other algorithms.

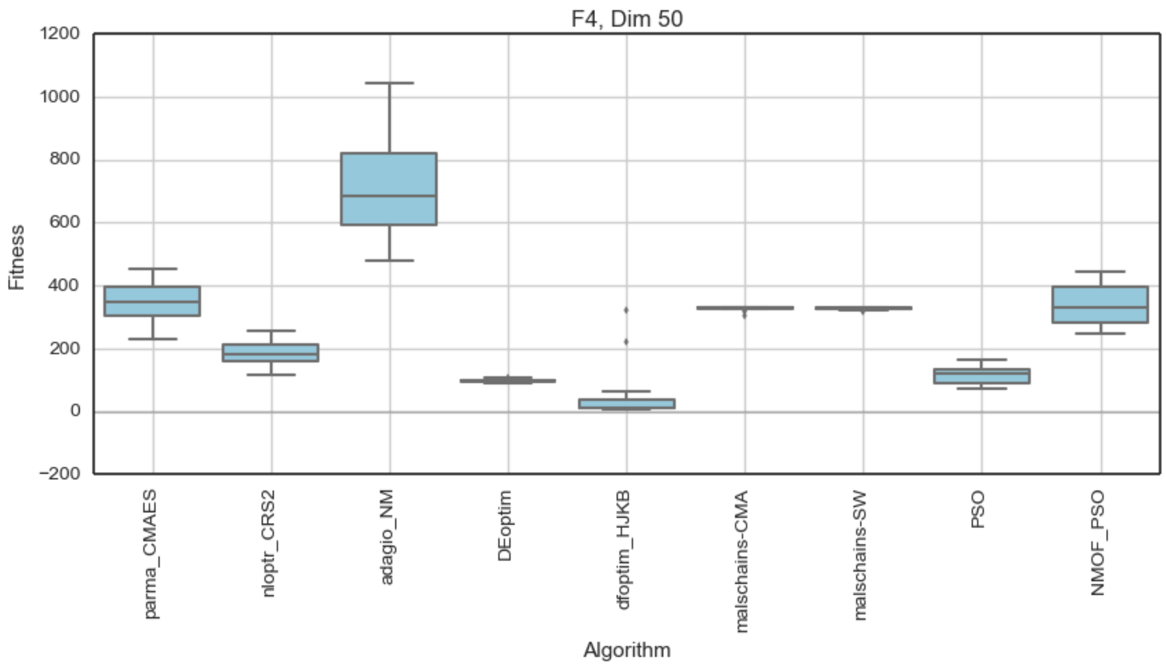
6. Conclusions

We have presented the R package **Rmalschains**. It implements the MA-LS-Chains algorithm, which is an algorithm framework of memetic algorithms with local search chains. The framework uses a steady-state genetic algorithm in combination with an LS method. Different LS methods are implemented. The algorithm chooses the individual on which to run the LS according to its fitness and its possibility to be enhanced with the LS. The LS is run for a fixed number of iterations, with the possibility to be continued on in a later stage of the algorithm. The algorithm has good performance, especially for high-dimensional problems. This was demonstrated in various optimization competitions, and in our experiments.

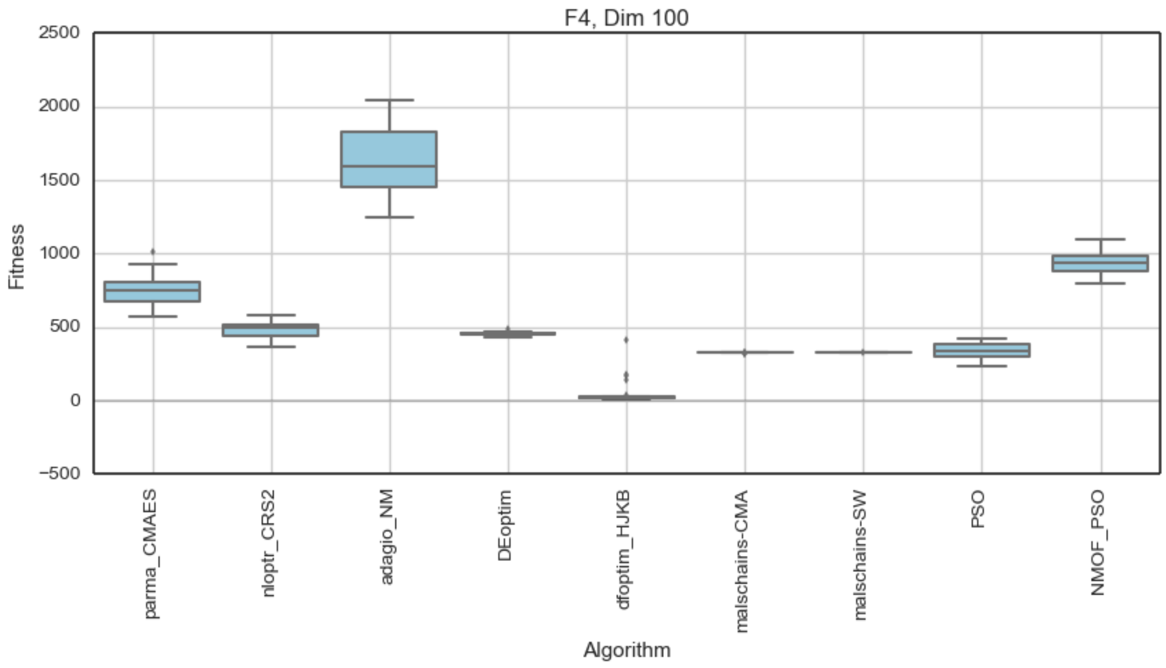
With presenting an implementation in R, we make the algorithm available to the R community and facilitate its use in general. We performed a rigorous experimental study comparing it to other general purpose optimizers already available in R, both with respect to quality of the results, as with respect to execution time. The study showed that, while in lower dimensions there exist competitive methods in R, often outperforming **Rmalschains**, for higher-dimensional problems the algorithm is very effective.

Acknowledgments

This work was supported in part by the Spanish Ministry of Science and Innovation (MICINN) under Project TIN-2009-14575. The work was performed while C. Bergmeir held a scholarship from the Spanish Ministry of Education (MEC) of the "Programa de Formación del Profesorado Universitario (FPU)".

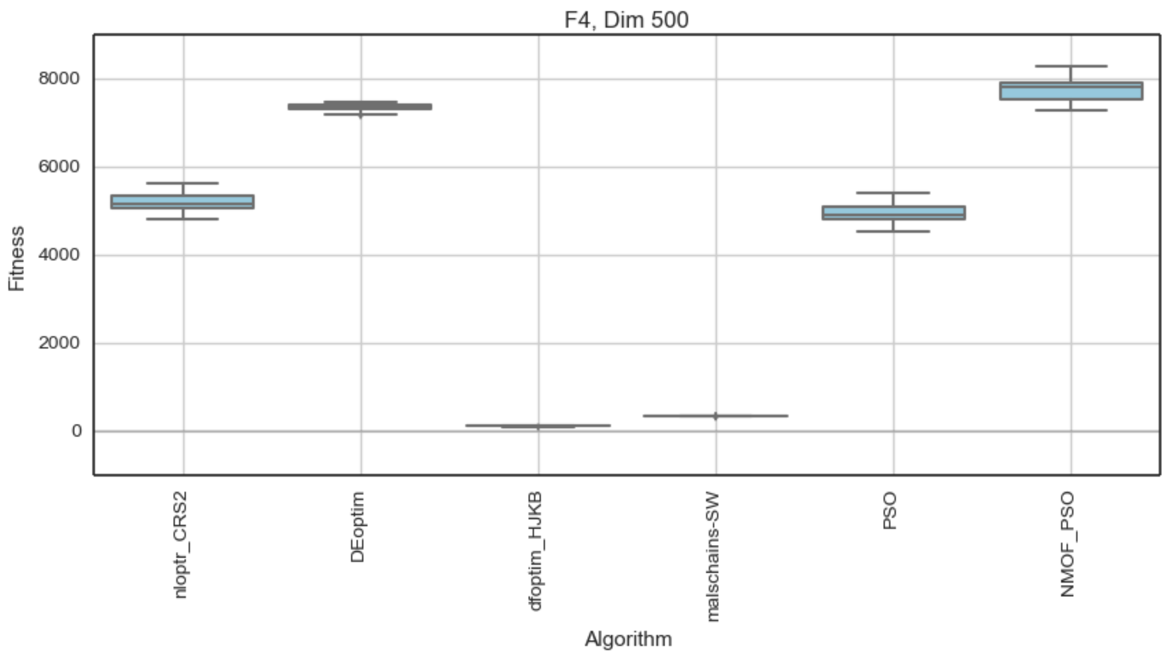


(a) Dimension 50

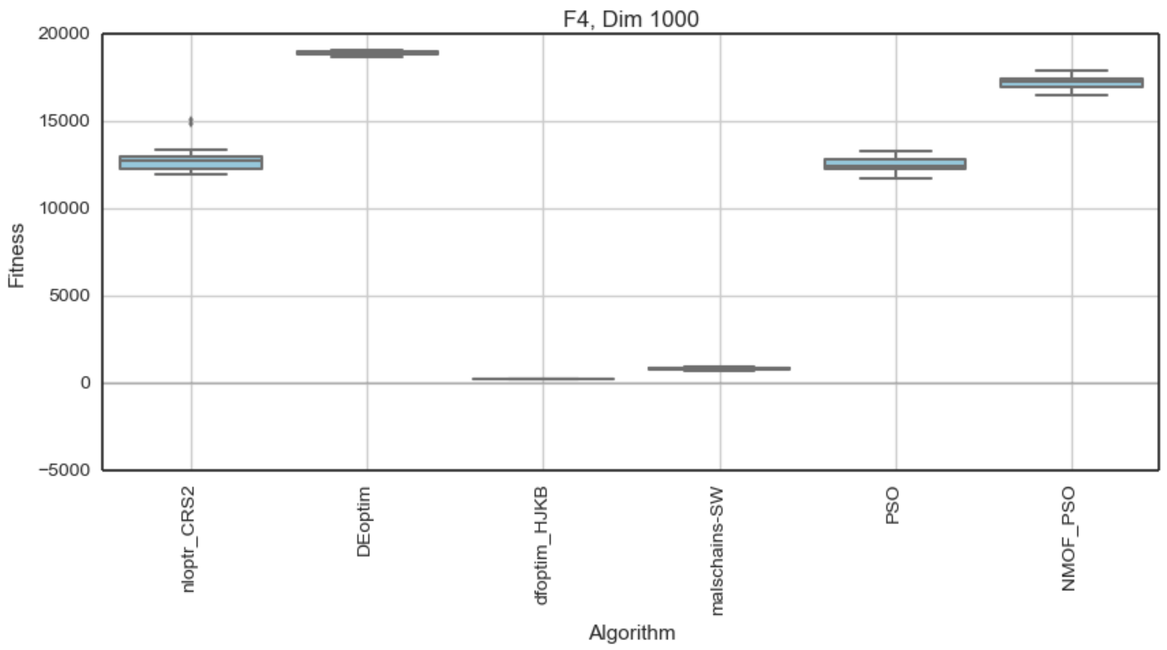


(b) Dimension 100

Figure 7: Results obtained by the packages for function F_4 for dimension 50, 100.

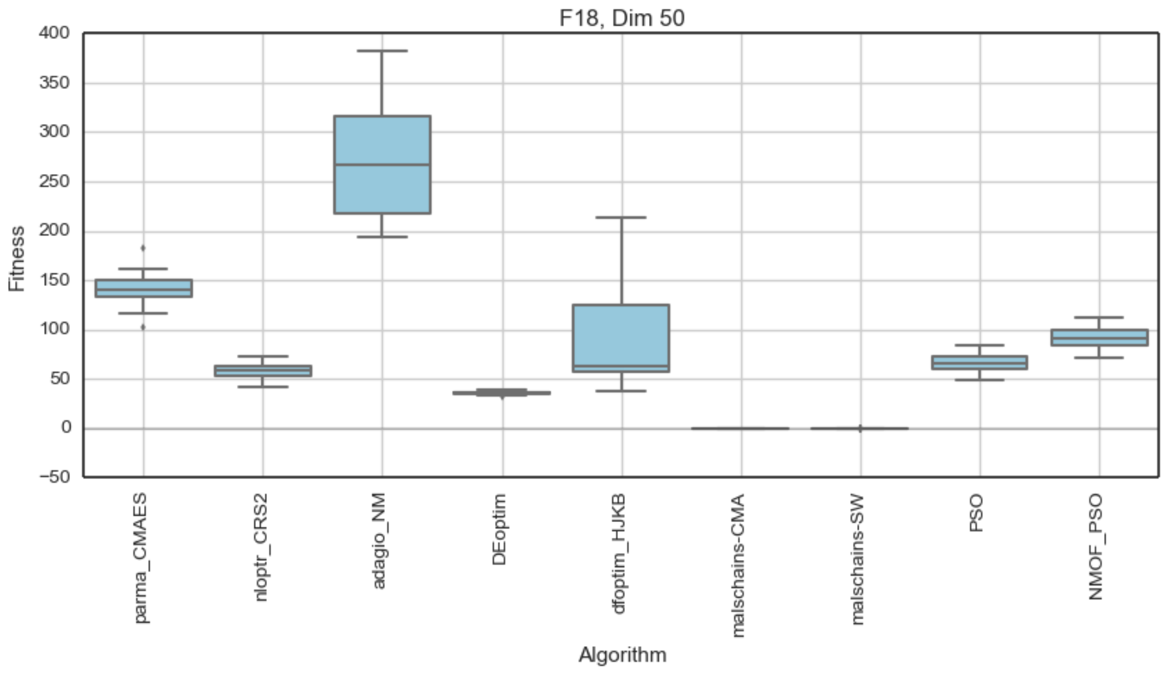


(a) Dimension 500

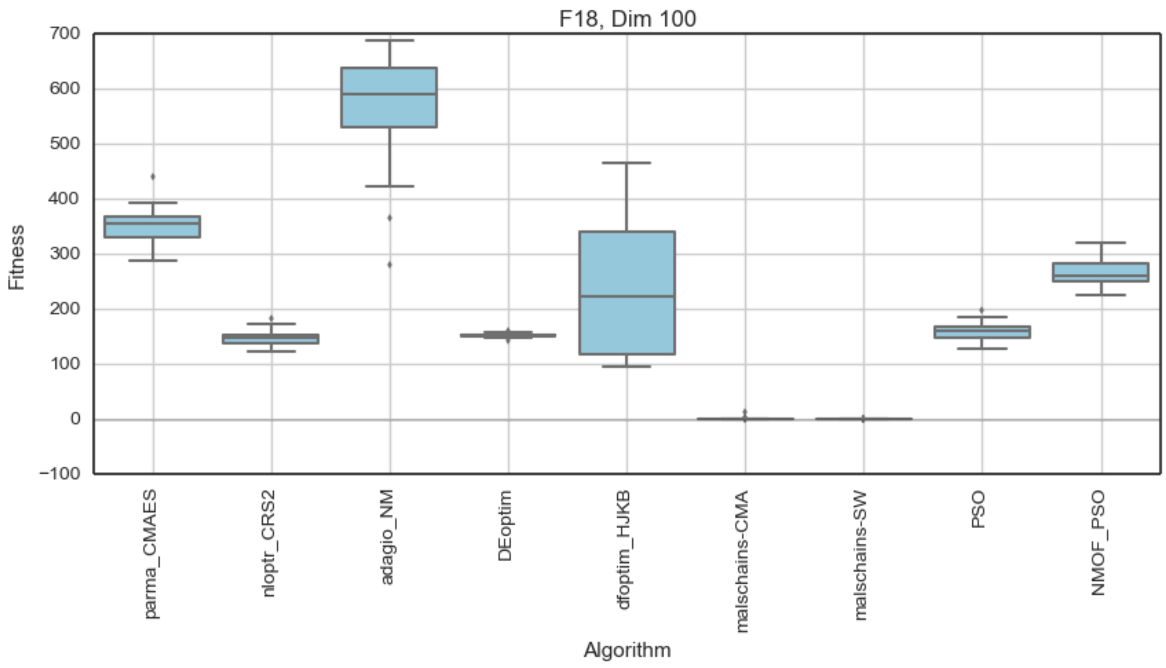


(b) Dimension 1000

Figure 8: Results obtained by the packages for function F_4 for dimension 500, 1000.

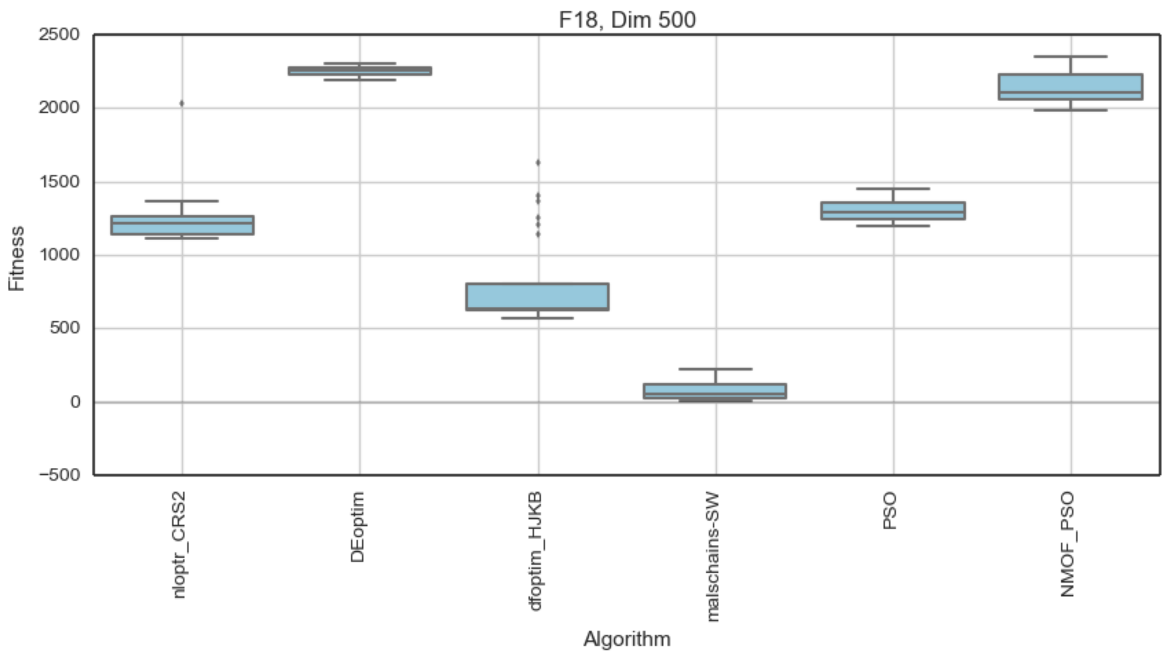


(a) Dimension 50

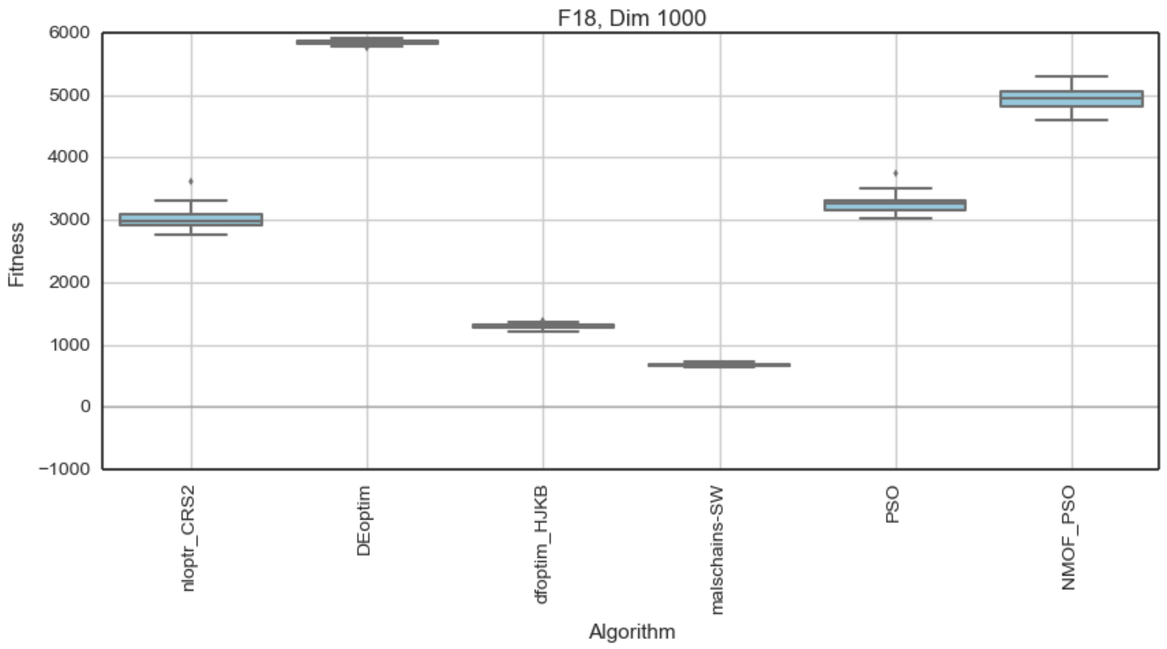


(b) Dimension 100

Figure 9: Results obtained by the packages for function F_{18} for dimension 50, 100.



(a) Dimension 500



(b) Dimension 1000

Figure 10: Results obtained by the packages for function F_{18} for dimension 500, 1000.

References

Ardia D, Boudt K, Carl P, Mullen K, Peterson B (2011). "Differential Evolution with **DEoptim**." *The R Journal*, **3**(1), 27–34. URL `https://journal.R-project.org/archive/2011-1/RJournal_2011-1_Ardia~et~al.pdf`.

Bäck T, Fogel D, Michalewicz Z (eds.) (1997). *Handbook of Evolutionary Computation*. IOP Publishing, Bristol, UK. `doi:10.1201/9781420050387`.

Bendtsen C (2012). ***pso**: Particle Swarm Optimization*. R package version 1.0.3, URL `https://CRAN.R-project.org/package=pso`.

Bergmeir C, Molina D, Benítez J (2016). ***Rmalschains**: Continuous Optimization Using Memetic Algorithms with Local Search Chains (MA-LS-Chains) in R*. R package version 0.2-3, URL `https://CRAN.R-project.org/package=Rmalschains`.

Bihorel S, Baudin M (2015). ***neldermead**: R Port of the **Scilab*** `neldermead` *Module*. R package version 1.0-10, URL `https://CRAN.R-project.org/package=neldermead`.

Borchers H (2016). ***adagio**: Discrete and Global Optimization Routines*. R package version 0.6.5, URL `https://CRAN.R-project.org/package=adagio`.

Burns P (2012). "A Comparison of Some Heuristic Optimization Methods." *Technical report*, Burns Statistics. URL `http://www.portfolioprobe.com/2012/07/23/a-comparison-of-some-heuristic-optimization-methods`.

Deb K, Suganthan PN (2005). "Special Session on Real-Parameter Optimization. 2005 IEEE CEC, Edinburgh, UK, Sept 2–5, 2005." In *Proceedings of the IEEE Conference on Evolutionary Computation, CEC*.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. `doi:10.18637/jss.v040.i08`.

Eshelman LJ, Schaffer JD (1993). "Real-Coded Genetic Algorithms in Genetic Algorithms by Preventing Incest." *Foundation of Genetic Algorithms 2*, pp. 187–202. `doi:10.1016/b978-0-08-094832-4.50018-0`.

Fernandes C, Rosa A (2001). "A Study on Non-Random Mating and Varying Population Size in Genetic Algorithms Using a Royal Road Function." In *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC*, volume 1, pp. 60–66.

García S, Molina D, Lozano M, Herrera F (2009). "A Study on the Use of Non-Parametric Tests for Analyzing the Evolutionary Algorithms' Behaviour: A Case Study on the CEC'2005 Special Session on Real Parameter Optimization." *Journal of Heuristics*, **15**(6), 617–644. `doi:10.1007/s10732-008-9080-4`.

Ghalanos A, Pfaff B (2016). ***parma**: Portfolio Allocation and Risk Management Applications*. R package version 1.5-3, URL `https://CRAN.R-project.org/package=parma`.

Ghalanos A, Theussl S (2016). ***Rsolnp**: General Non-Linear Optimization Using Augmented Lagrange Multiplier Method*. R package version 1.16, URL `https://CRAN.R-project.org/package=Rsolnp`.

Goldberg D, Deb K (1991). "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms." In G Rawlins (ed.), *Foundations of Genetic Algorithms*, volume 1, pp. 69–93. Morgan Kaufmann Publishers, San Mateo. doi:10.1016/b978-0-08-050684-5.50008-2.

Goldberg D, Voessner S (1999). "Optimizing Global-Local Search Hybrids." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 1999*, pp. 220–228.

Hankin R (2006). "Special Functions in R: Introducing the **gsl** Package." *R News*, **6**(4), 24–26. URL https://CRAN.R-project.org/doc/Rnews/Rnews_2006-4.pdf.

Hansen N (2012). *cmaes.m: Evolution Strategy with Covariance Matrix Adaptation for Nonlinear Function Minimization.* MATLAB code version 3.60, URL https://www.lri.fr/~hansen/count-cmaes-m.php?Down=cmaes.m.

Hansen N, Auger A, Ros R, Finck S, Pošík P (2010). "Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009." In *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference, GECCO '10 – Companion Publication*, pp. 1689–1696.

Hansen N, Müller S, Koumoutsakos P (2003). "Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)." *Evolutionary Computation*, **1**(11), 1–18. doi:10.1162/106365603321828970.

Johnson S (2012). *The **NLopt** Nonlinear-Optimization Package.* URL http://ab-initio.mit.edu/nlopt.

Krasnogor N, Smith J (2005). "A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues." *IEEE Transactions on Evolutionary Computation*, **9**(5), 474–488. doi:10.1109/tevc.2005.850260.

Lozano M, Molina D, Herrera F (2011). "Editorial Scalability of Evolutionary Algorithms and Other Metaheuristics for Large-Scale Continuous Optimization Problems." *Soft Computing*, **15**(11), 2085–2087. doi:10.1007/s00500-010-0639-2.

Mebane Jr WR, Sekhon JS (2011). "Genetic Optimization Using Derivatives: The **rgenoud** Package for R." *Journal of Statistical Software*, **42**(11), 1–26. doi:10.18637/jss.v042.i11.

Mersmann O (2015). ***microbenchmark**: Sub Microsecond Accurate Timing Functions.* R package version 1.4-2.1, URL https://CRAN.R-project.org/package=microbenchmark.

Michalewicz Z, Fogel DB (2004). *How to Solve It: Modern Heuristics.* Springer-Verlag. doi:10.1007/978-3-662-07807-5.

Molina D (2012). ***librealea**: An Implementation of Evolutionary Algorithms for Real Optimisation.* URL https://bitbucket.org/dmolina/librealea.

Molina D, Lozano M, García-Martínez C, Herrera F (2010). "Memetic Algorithms for Continuous Optimisation Based on Local Search Chains." *Evolutionary Computation*, **18**(1), 27–63. doi:10.1162/evco.2010.18.1.18102.

Molina D, Lozano M, Sánchez AM, Herrera F (2011). "Memetic Algorithms Based on Local Search Chains for Large Scale Continuous Optimisation Problems: MA-SSW-Chains." *Soft Computing*, **15**(11), 2201–2220. doi:10.1007/s00500-010-0647-2.

Moscato P (1999). "Memetic Algorithms: A Short Introduction." In D Corne, M Dorigo, F Glover, D Dasgupta, P Moscato, R Poli, K Price (eds.), *New Ideas in Optimization*, pp. 219–234. McGraw-Hill, Maidenhead.

Mühlenbein H, Schlierkamp-Voosen D (1993). "Predictive Models for the Breeder Genetic Algorithm – I. Continuous Parameter Optimization." *Evolutionary Computation*, **1**(1), 25–49. doi:10.1162/evco.1993.1.1.25.

Mühlenbein H, Schomisch M, Born J (1991). "The Parallel Genetic Algorithm as Function Optimizer." *Parallel Computing*, **17**(6–7), 619–632. doi:10.1016/s0167-8191(05)80052-3.

Mullen K (2014). "Continuous Global Optimization in R." *Journal of Statistical Software*, **60**(6), 1–45. doi:10.18637/jss.v060.i06.

Mullen KM, Ardia D, Gil DL, Windover D, Cline J (2011). "**DEoptim**: An R Package for Global Optimization by Differential Evolution." *Journal of Statistical Software*, **40**(6), 1–26. doi:10.18637/jss.v040.i06.

Nash J (2014). "On Best Practice Optimization Methods in R." *Journal of Statistical Software*, **60**(2), 1–14. doi:10.18637/jss.v060.i02.

Nelder J, Singer S (2009). "Nelder-Mead Algorithm." *Scholarpedia*, **4**(2), 2928.

Nelder JA, Mead R (1965). "A Simplex Method for Function Minimization." *The Computer Journal*, **7**(4), 308–313. doi:10.1093/comjnl/7.4.308.

Nomura T, Shimohara K (2001). "An Analysis of Two-Parent Recombinations for Real-Valued Chromosomes in an Infinite Population." *Evolutionary Computation*, **9**(3), 283–308. doi:10.1162/106365601750406000.

Press W, Teukolsky S, Vetterling W, Flannery B (2007). *Numerical Recipes: The Art of Scientific Computing*. 3rd edition. Cambridge University Press, Cambridge.

Price KV, Storn RM, Lampinen JA (2005). *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. Springer-Verlag.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Schumann E (2016). ***NMOF**: Numerical Methods and Optimization in Finance.* R package version 0.40-0, URL https://CRAN.R-project.org/package=NMOF.

Smith JE (2002). "Genetic Algorithms." In PM Pardalos, HE Romeijn (eds.), *Handbook of Global Optimization*, volume 62 of *Nonconvex Optimization and Its Applications*, pp. 275–362. Springer-Verlag. doi:10.1007/978-1-4757-5362-2_9.

Solis FJ, Wets RJB (1981). "Minimization by Random Search Techniques." *Mathematics of Operations Research*, **6**(1), 19–30. doi:10.1287/moor.6.1.19.

Tang K (2008). "Summary of Results on CEC'08 Competition on Large Scale Global Optimization." *Technical report*, Nature Inspired Computation and Application Lab (NICAL). URL http://nical.ustc.edu.cn/papers/CEC2008_SUMMARY.pdf.

Tang K, Li X, Suganthan PN (2010). "Session on Large Scale Global Optimization. 2010 IEEE World Congress on Computational Intelligence (CEC@WCCI-2010), July 18–23, 2010, Barcelona, Spain." In *Proceedings of the IEEE Conference on Evolutionary Computation, CEC*.

The MathWorks Inc (2014). *MATLAB – The Language of Technical Computing, Version R2014b*. Natick. URL http://www.mathworks.com/products/matlab/.

Theussl S, Borchers HW (2016). "CRAN Task View: Optimization and Mathematical Programming." Version 2016-06-06, URL https://CRAN.R-project.org/view=Optimization.

Trautmann H, Mersmann O, Arnu D (2011). ***cmaes**: Covariance Matrix Adapting Evolutionary Strategy*. R package version 1.0-11, URL http://CRAN.R-project.org/package=cmaes.

Tsallis C, Stariolo D (1996). "Generalized Simulated Annealing." *Physica A: Statistical Mechanics and Its Applications*, **233**(1–2), 395–406. doi:10.1016/s0378-4371(96)00271-3.

Varadhan R (2015). ***alabama**: Constrained Nonlinear Optimization*. R package version 2015.3-1, URL http://CRAN.R-project.org/package=alabama.

Varadhan R, Borchers H (2016). ***dfoptim**: Derivative-Free Optimization*. R package version 2016.7-1, URL http://CRAN.R-project.org/package=dfoptim.

Venables W, Ripley B (2002). *Modern Applied Statistics with S*. 4th edition. Springer-Verlag, New York. doi:10.1007/978-0-387-21706-2.

Whitley D (1989). "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials Is Best." *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 116–121.

Wolpert DH, Macready WG (1997). "No Free Lunch Theorems for Optimization." *IEEE Transactions on Evolutionary Computation*, **1**(1), 67–82. doi:10.1109/4235.585893.

Xiang Y, Gubian S, Suomela B, Hoeng J (2013). "Generalized Simulated Annealing for Efficient Global Optimization: The **GenSA** Package for R." *The R Journal*, **5**(1), 13–28. URL https://journal.r-project.org/archive/2013-1/xiang-gubian-suomela-etal.pdf.

Ypma J, Borchers H, Eddelbuettel D (2014). ***nloptr**: R Interface to **NLopt***. R package version 1.0.4, URL https://CRAN.R-project.org/package=nloptr.

A. Average accuracy result tables

In Tables 4, 5, 6, 7, 8, 9, 10, and 11, we present the average errors, i.e., the differences between global minimum and achieved value, averaged over 25 test runs, for each package and function for dimension 10, 30, 50, 100, 200, 500, and 1000, respectively. In the last tables, for dimension 500 and 1000, we can observe that for functions F_7 and F_{15} the results of several packages are reported as infinity (Inf). Because this function is defined as

$$\sum_{i=1}^{D} |x_i| + \prod_{i=1}^{D} |x_i|,$$

for high dimensions, numeric overflows may occur.

Affiliation:

Christoph Bergmeir
Faculty of Information Technology
Monash University
3800 VIC, Melbourne, Australia
E-mail: christoph.bergmeir@monash.edu

Daniel Molina
Computer Science and Engineering
University of Cádiz
Cádiz, Spain
Email: daniel.molina@uca.es

José M. Benítez
Department of Computer Science and Artificial Intelligence
E.T.S. de Ingenierías Informática y de Telecomunicación
CITIC-UGR, University of Granada
18071 Granada, Spain
E-mail: j.m.benitez@decsai.ugr.es
URL: http://dicits.ugr.es, http://sci2s.ugr.es

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
December 2016, Volume 75, Issue 4 *Submitted:* 2012-07-24
doi:10.18637/jss.v075.i04 *Accepted:* 2015-09-06

Function	`nloptr`	`parma`	`adagio`	**DEoptim**	`dfoptim`	`malsch`	`malsch`	**PSO**	`NMOF`
	`_CRS2`	`_CMAES`	`_NM`		`_HJKB`	`ains-CMA`	`ains-SW`		`_PSO`
F_1	1.887e+01	4.347e+01	0.000e+00	0.000e+00	2.328e–12	2.100e+02	2.473e+02	9.075e–01	4.169e–05
F_2	0.000e+00	2.851e–12	1.188e–11	0.000e+00	1.206e–06	1.322e+01	1.994e+01	0.000e+00	2.315e–03
F_3	1.516e+03	1.704e+04	8.795e–08	0.000e+00	3.017e+03	0.000e+00	2.175e+00	1.810e+00	3.094e+00
F_4	1.214e+00	4.423e+00	1.870e+00	2.547e–13	1.194e–01	7.943e+00	1.874e+01	3.582e–01	1.033e–04
F_5	3.342e–01	4.558e+01	3.148e–01	2.959e–04	8.858e+01	3.137e+00	1.189e+01	6.268e–04	7.621e–01
F_6	1.704e+00	1.514e+01	5.184e+00	0.000e+00	4.128e–01	1.150e+01	1.389e+01	0.000e+00	2.712e–03
F_7	1.084e–03	5.741e–01	3.733e–12	0.000e+00	1.906e–06	6.653e–09	5.605e–09	0.000e+00	7.021e–04
F_8	2.090e–02	8.476e–17	1.317e–12	0.000e+00	4.062e–12	6.079e–09	5.168e–09	0.000e+00	2.387e–06
F_9	1.066e+01	1.244e+01	1.193e+01	1.059e+01	1.247e+01	1.062e+01	1.069e+01	1.059e+01	1.076e+01
F_{10}	1.305e–01	3.759e–02	7.062e–02	0.000e+00	1.880e–02	4.787e–09	5.267e–09	0.000e+00	1.168e–05
F_{11}	7.982e–02	8.086e+00	6.805e+00	2.146e–04	9.050e+00	0.000e+00	2.146e–04	0.000e+00	6.271e–02
F_{12}	3.097e+01	4.863e+01	0.000e+00	0.000e+00	2.866e–12	4.217e–09	4.960e–09	1.210e+00	3.451e–05
F_{13}	1.070e+03	1.788e+05	1.283e–08	3.004e–06	8.673e+01	6.040e–09	3.333e–02	6.482e–01	8.261e–01
F_{14}	1.349e+00	4.247e+00	4.138e+00	0.000e+00	1.990e–01	4.634e–09	3.947e–09	3.047e–01	1.013e–04
F_{15}	9.142e–04	3.654e–01	3.644e–12	0.000e+00	1.885e–06	7.007e–09	6.391e–09	0.000e+00	8.242e–04
F_{16}	3.076e+00	6.794e+00	5.504e+00	5.481e–38	4.174e+02	7.020e–09	7.426e–09	2.420e+00	3.807e–02
F_{17}	7.096e–03	1.789e+01	1.243e+01	1.218e–24	1.811e+01	8.193e–09	7.383e–05	4.440e–19	1.193e–01
F_{18}	3.949e–03	9.650e–01	2.836e+00	1.744e–25	1.851e+00	7.927e–09	8.006e–09	9.516e–20	4.038e–02
F_{19}	1.000e+00	1.037e+00	1.170e+00	1.000e+00	1.113e+00	9.600e–01	1.000e+00	1.000e+00	1.000e+00

Table 4: Results for each package for dimension 2.

Function	`nloptr` `_CRS2`	`parma` `_CMAES`	`adagio` `_NM`	**DEoptim**	`dfoptim` `_HJKB`	`malsch` `ains-CMA`	`malsch` `ains-SW`	**PSO**	`NMOF` `_PSO`
F_1	1.593e–09	3.881e+02	2.679e–05	0.000e+00	1.324e–11	4.042e+02	3.958e+02	0.000e+00	8.802e–05
F_2	0.000e+00	1.498e+01	4.512e+00	2.090e–03	2.409e+00	8.175e+01	7.720e+01	1.232e–10	1.765e–02
F_3	7.973e–01	5.711e+07	2.350e+01	3.970e+00	2.987e+07	1.664e–11	2.547e+00	3.171e+01	3.031e+02
F_4	1.099e+01	3.053e+01	7.029e+01	1.368e+00	1.313e+00	1.608e+02	1.389e+02	7.044e+00	7.809e+00
F_5	4.410e–02	5.014e+00	4.417e+00	1.150e–02	3.624e+00	1.413e+02	1.459e+02	6.589e–02	7.019e+01
F_6	8.730e–06	1.630e+01	1.687e+01	4.243e–09	4.495e–06	2.027e+01	2.008e+01	5.237e–01	1.176e+00
F_7	8.119e–06	5.288e+00	1.805e–03	3.288e–10	9.011e–06	9.184e–09	9.655e–09	0.000e+00	4.901e–03
F_8	2.576e–10	3.086e+02	1.274e–06	2.754e–01	3.604e+03	7.847e–09	9.863e–09	3.646e–12	4.781e–03
F_9	6.323e+01	6.648e+01	6.638e+01	5.682e+01	6.996e+01	5.830e+01	5.762e+01	5.869e+01	6.294e+01
F_{10}	1.452e–09	6.371e–01	4.190e+00	2.842e–18	1.205e+00	7.934e–09	8.951e–09	3.038e–01	3.322e–04
F_{11}	9.570e–02	8.009e+01	7.322e+01	2.042e–02	8.644e+01	4.922e–04	1.646e–08	9.205e–02	2.284e+01
F_{12}	4.060e+00	1.875e+02	2.102e+01	1.469e–03	8.898e+02	8.693e–09	9.264e–09	3.707e+00	8.036e+00
F_{13}	4.893e+00	1.390e+06	3.379e+01	2.750e+00	2.734e+08	4.991e+00	2.117e+00	2.473e+01	1.515e+01
F_{14}	9.004e+00	2.885e+01	6.280e+01	1.268e+00	5.839e+00	8.280e–09	4.353e–02	7.481e+00	6.648e+00
F_{15}	1.000e–07	5.432e+00	5.136e–01	9.753e–11	7.519e–02	9.019e–09	9.630e–09	1.880e–02	3.718e–03
F_{16}	1.534e+01	8.994e+01	4.642e+01	6.853e–04	4.223e+01	1.717e–03	9.550e–09	1.095e+01	1.444e+01
F_{17}	2.105e+01	7.772e+01	6.845e+01	7.136e–01	1.982e+04	8.185e+00	3.805e+00	6.199e+00	2.597e+01
F_{18}	1.929e+00	1.513e+01	2.651e+01	1.672e–02	1.347e+01	8.819e–09	1.376e–08	1.264e+00	7.555e–01
F_{19}	1.175e–10	6.890e+00	3.234e+00	3.429e–14	9.902e–01	8.328e–09	9.017e–09	7.958e–02	1.292e–03

Table 5: Results for each package for dimension 10.

Function	`nloptr` `_CRS2`	`parma` `_CMAES`	`adagio` `_NM`	**DEoptim**	`dfoptim` `_HJKB`	`malsch` `ains-CMA`	`malsch` `ains-SW`	**PSO**	`NMOF` `_PSO`
F_1	4.127e–09	3.658e+03	1.808e+03	2.984e–02	3.790e–11	4.330e+02	4.389e+02	4.483e–10	2.156e–01
F_2	6.648e+00	4.122e+01	5.284e+01	1.505e+01	1.512e+01	1.281e+02	1.246e+02	1.288e+01	4.547e–01
F_3	3.166e+01	2.501e+08	8.178e+08	1.777e+02	1.941e+09	9.421e+00	4.634e+01	2.940e+02	5.463e+02
F_4	7.257e+01	1.361e+02	4.075e+02	2.752e+01	1.600e+01	3.234e+02	3.209e+02	5.140e+01	1.263e+02
F_5	5.215e–03	3.687e+01	2.562e+01	1.015e–01	1.319e–01	1.761e+02	1.740e+02	2.236e–02	5.773e+02
F_6	1.758e–01	1.753e+01	1.988e+01	9.988e–02	4.381e–06	2.118e+01	2.111e+01	1.019e+00	1.905e+01
F_7	1.654e–05	1.309e+01	8.088e+00	4.042e–02	2.912e–05	9.205e–09	9.957e–09	8.510e–07	7.247e–01
F_8	2.384e–09	2.369e+03	1.246e+03	2.933e+03	3.423e+04	1.250e–08	1.018e–08	1.938e+02	7.068e+00
F_9	1.912e+02	1.961e+02	1.969e+02	1.773e+02	2.024e+02	1.769e+02	1.742e+02	1.826e+02	1.924e+02
F_{10}	3.079e+00	5.295e+01	3.473e+01	2.981e–02	3.355e+00	8.540e–09	9.841e–09	3.485e+00	3.168e+00
F_{11}	3.283e+01	2.668e+02	2.562e+02	1.800e+01	2.677e+02	7.918e–02	1.010e–02	1.427e+01	1.341e+02
F_{12}	5.400e+01	1.441e+03	7.199e+01	3.494e+00	6.420e+01	6.312e–03	9.723e–09	3.609e+01	3.952e+01
F_{13}	4.617e+01	6.683e+07	4.986e+02	9.896e+01	1.940e+09	6.871e+01	3.423e+01	1.097e+02	9.643e+01
F_{14}	5.290e+01	1.200e+02	3.379e+02	2.242e+01	4.162e+01	2.836e–01	3.199e–01	4.509e+01	9.139e+01
F_{15}	1.291e–01	1.059e+02	2.482e+01	4.455e–02	7.659e–01	1.549e–07	9.891e–09	3.677e–01	8.702e–01
F_{16}	1.129e+02	5.767e+02	1.466e+02	8.872e+00	5.669e+02	2.404e–02	9.707e–09	5.752e+01	8.461e+01
F_{17}	1.020e+02	3.448e+05	2.114e+02	5.847e+01	2.726e+09	3.785e+00	2.423e+00	1.286e+02	1.426e+02
F_{18}	2.203e+01	5.739e+01	1.153e+02	9.061e+00	4.499e+01	6.417e–03	9.848e–09	2.925e+01	3.667e+01
F_{19}	1.427e+00	7.501e+01	1.041e+01	2.524e–02	3.155e+00	8.732e–09	9.547e–09	2.528e+00	2.039e+00

Table 6: Results for each package for dimension 30.

Function	`nloptr`	`parma`	`adagio`	**DEoptim**	`dfoptim`	`malsch`	`malsch`	**PSO**	`NMOF`
	`_CRS2`	`_CMAES`	`_NM`		`_HJKB`	`ains-CMA`	`ains-SW`		`_PSO`
F_1	9.778e–09	3.956e+03	1.029e+04	2.824e+01	5.968e–11	4.355e+02	4.428e+02	3.781e–04	1.651e+00
F_2	2.920e+01	5.096e+01	7.994e+01	4.779e+01	2.471e+01	1.433e+02	1.420e+02	3.818e+01	3.248e+01
F_3	7.596e+01	7.045e+08	5.236e+09	1.653e+04	5.650e+09	4.895e+01	4.599e+01	2.570e+02	5.579e+02
F_4	1.836e+02	2.363e+02	7.172e+02	9.742e+01	4.083e+01	3.269e+02	3.274e+02	1.168e+02	3.378e+02
F_5	9.797e–03	3.719e+01	3.240e+01	1.216e+00	2.552e–01	1.752e+02	1.748e+02	1.639e–02	1.060e+03
F_6	2.214e+00	1.848e+01	1.985e+01	3.122e+00	4.333e–06	2.126e+01	2.125e+01	2.068e+00	1.924e+01
F_7	2.939e–04	2.591e+01	1.069e+02	1.913e+00	4.759e–05	9.615e–09	1.000e–08	1.259e–02	3.599e+00
F_8	3.408e+01	4.998e+03	3.694e+03	1.465e+04	1.361e+05	1.008e+00	1.040e–08	4.053e+03	1.465e+02
F_9	3.246e+02	3.308e+02	3.299e+02	3.081e+02	3.302e+02	2.867e+02	2.950e+02	3.158e+02	3.274e+02
F_{10}	1.330e+01	6.281e+01	2.150e+02	1.604e+01	6.264e+00	8.564e–09	9.880e–09	8.447e+00	2.077e+01
F_{11}	1.203e+02	4.547e+02	4.404e+02	9.541e+01	4.672e+02	0.000e+00	1.677e–02	5.029e+01	2.652e+02
F_{12}	1.030e+02	3.101e+03	3.132e+03	5.101e+01	1.046e+02	1.790e–08	1.002e–08	7.057e+01	7.493e+01
F_{13}	1.358e+02	3.383e+08	6.656e+08	2.791e+03	6.509e+09	2.237e+00	3.818e+01	2.271e+02	3.222e+02
F_{14}	1.225e+02	1.787e+02	5.449e+02	7.195e+01	5.267e+01	4.657e–09	7.562e–01	9.430e+01	2.354e+02
F_{15}	3.953e–01	3.007e+02	5.283e+02	3.185e+00	1.349e+00	4.323e–09	9.964e–09	1.137e+00	5.580e+00
F_{16}	2.100e+02	1.314e+03	1.419e+03	7.682e+01	2.136e+03	2.070e–03	4.923e–03	1.240e+02	1.545e+02
F_{17}	2.249e+02	1.174e+07	3.618e+02	2.669e+02	1.678e+09	0.000e+00	2.151e+01	3.340e+02	3.844e+02
F_{18}	5.801e+01	1.210e+02	2.722e+02	3.583e+01	9.064e+01	0.000e+00	4.292e–04	6.656e+01	9.191e+01
F_{19}	6.735e+00	2.258e+02	2.394e+02	7.005e+00	3.896e+00	7.657e–09	9.304e–09	7.388e+00	1.983e+01

Table 7: Results for each package for dimension 50.

Function	nloptr	parma	adagio	**DEoptim**	dfoptim	malsch	malsch	**PSO**	NMOF
	_CRS2	_CMAES	_NM		_HJKB	ains-CMA	ains-SW		_PSO
F_1	8.423e+02	1.140e+04	3.586e+04	5.679e+03	1.223e–10	4.343e+02	4.446e+02	2.311e+01	1.820e+01
F_2	6.153e+01	7.660e+01	1.033e+02	9.912e+01	3.136e+01	1.587e+02	1.552e+02	7.518e+01	6.232e+01
F_3	2.263e+07	1.208e+09	2.317e+10	9.450e+07	2.130e+09	1.188e+02	1.147e+02	6.429e+05	8.941e+03
F_4	4.830e+02	5.191e+02	1.650e+03	4.550e+02	5.437e+01	3.283e+02	3.297e+02	3.347e+02	9.321e+02
F_5	7.861e+00	9.322e+01	2.327e+02	4.884e+01	1.959e–01	1.747e+02	1.780e+02	1.152e+00	2.288e+03
F_6	6.630e+00	1.987e+01	1.988e+01	1.132e+01	4.366e–06	2.138e+01	2.135e+01	4.073e+00	1.982e+01
F_7	1.206e+01	7.030e+01	2.867e+30	4.469e+01	9.612e–05	9.773e–09	1.010e–08	2.705e+00	2.291e+01
F_8	5.128e+03	1.203e+04	1.155e+04	8.600e+04	1.007e+06	1.803e+03	6.989e–04	3.423e+04	4.169e+03
F_9	6.488e+02	6.317e+02	6.298e+02	6.190e+02	6.401e+02	5.675e+02	5.668e+02	6.128e+02	6.424e+02
F_{10}	5.819e+01	8.868e+01	1.787e+03	3.334e+02	1.204e+01	1.135e–08	9.919e–09	3.278e+01	8.810e+01
F_{11}	3.806e+02	9.132e+02	9.165e+02	4.520e+02	9.675e+02	2.346e+00	1.357e–01	3.253e+02	5.987e+02
F_{12}	2.846e+02	7.074e+03	2.372e+04	2.516e+03	1.170e+04	2.538e+00	2.020e+00	1.874e+02	1.746e+02
F_{13}	5.189e+05	3.570e+08	1.470e+10	2.090e+07	2.820e+10	1.195e+02	7.711e+01	4.764e+03	1.992e+03
F_{14}	3.634e+02	4.057e+02	1.219e+03	3.257e+02	9.099e+01	2.640e+00	4.127e+00	2.517e+02	6.561e+02
F_{15}	7.260e+01	5.896e+02	2.242e+03	8.723e+01	3.409e+00	7.333e–06	9.950e–09	9.711e+00	6.329e+01
F_{16}	4.452e+02	2.729e+03	1.111e+04	7.047e+02	1.898e+04	2.859e+00	1.698e+00	3.626e+02	3.361e+02
F_{17}	6.631e+02	7.313e+07	8.679e+02	4.291e+04	1.500e+10	2.219e+02	2.265e+02	8.250e+02	9.072e+02
F_{18}	1.479e+02	2.448e+02	5.686e+02	1.521e+02	2.313e+02	9.421e–01	1.260e–01	1.600e+02	2.689e+02
F_{19}	5.433e+01	9.881e+02	1.165e+03	1.496e+02	9.157e+00	1.457e–08	8.684e–09	2.203e+01	8.886e+01

Table 8: Results for each package for dimension 100.

Function	`nloptr` `_CRS2`	**DEoptim**	`dfoptim` `_HJKB`	`malsch` `ains-SW`	`adagio` `_NM`	**PSO**	`NMOF` `_PSO`
F_1	9.214e+04	1.237e+05	2.390e–10	4.477e+02	8.724e+04	3.805e+03	1.452e+03
F_2	7.802e+01	1.330e+02	3.253e+01	1.696e+02	1.193e+02	9.715e+01	7.881e+01
F_3	8.230e+09	1.818e+10	5.505e+05	2.167e+02	1.135e+11	8.114e+08	2.049e+06
F_4	1.360e+03	1.696e+03	1.264e+02	3.296e+02	3.095e+03	1.133e+03	2.468e+03
F_5	7.193e+02	1.012e+03	1.152e–01	1.790e+02	7.572e+02	4.085e+01	5.042e+03
F_6	1.544e+01	1.836e+01	4.375e–06	2.143e+01	1.987e+01	9.610e+00	2.025e+01
F_7	1.588e+02	3.086e+02	1.914e–04	1.023e–08	6.781e+76	7.089e+01	1.193e+02
F_8	3.593e+04	3.771e+05	3.849e+06	4.766e+00	5.036e+04	1.831e+05	3.698e+04
F_9	1.398e+03	1.346e+03	1.287e+03	1.191e+03	1.268e+03	1.279e+03	1.344e+03
F_{10}	7.295e+02	5.299e+03	2.489e+01	9.986e–09	4.016e+03	4.851e+02	3.168e+02
F_{11}	9.918e+02	1.394e+03	1.914e+03	6.989e–01	1.846e+03	1.162e+03	1.308e+03
F_{12}	3.927e+04	6.711e+04	1.210e+05	3.331e+01	6.468e+04	1.336e+03	6.099e+02
F_{13}	1.546e+09	6.365e+09	7.181e+10	3.500e+02	6.681e+10	5.097e+07	5.576e+04
F_{14}	9.292e+02	1.223e+03	2.385e+02	2.731e+01	2.494e+03	7.873e+02	1.816e+03
F_{15}	7.902e+02	1.443e+03	5.847e+00	9.943e–09	1.079e+48	1.326e+02	3.411e+02
F_{16}	5.305e+03	1.979e+04	7.854e+04	9.839e+00	4.409e+04	1.102e+03	7.710e+02
F_{17}	5.516e+03	1.340e+08	5.063e+10	7.692e+01	4.388e+09	2.249e+03	1.756e+03
F_{18}	3.928e+02	5.283e+02	3.847e+02	2.518e–01	1.155e+03	3.860e+02	6.726e+02
F_{19}	7.931e+02	2.655e+03	1.822e+01	8.385e–08	7.097e+03	2.828e+02	2.991e+02

Table 9: Results for each package for dimension 200.

Function	**DEoptim**	`dfoptim _HJKB`	`malsch ains-SW`	**PSO**	`NMOF _PSO`
F_1	1.248e+06	6.012e–10	4.490e+02	1.673e+05	1.152e+05
F_2	1.614e+02	4.974e+01	1.805e+02	1.176e+02	9.175e+01
F_3	7.654e+11	2.554e+11	5.055e+02	6.247e+10	4.607e+09
F_4	7.349e+03	1.225e+02	3.301e+02	4.946e+03	7.770e+03
F_5	1.037e+04	1.780e–01	1.796e+02	1.548e+03	1.328e+04
F_6	2.074e+01	4.384e–06	2.149e+01	1.827e+01	2.056e+01
F_7	3.401e+113	Inf	1.043e–08	7.542e+02	1.621e+41
F_8	2.333e+06	1.850e+07	2.207e+03	1.214e+06	2.790e+05
F_9	3.753e+03	3.224e+03	3.036e+03	3.447e+03	3.619e+03
F_{10}	4.964e+04	6.528e+01	8.398e–02	7.783e+03	2.163e+03
F_{11}	4.732e+03	4.841e+03	9.419e+01	3.830e+03	3.578e+03
F_{12}	8.341e+05	3.315e+05	4.134e+02	6.081e+04	4.227e+04
F_{13}	4.559e+11	5.983e+11	9.248e+02	2.073e+10	6.367e+08
F_{14}	5.428e+03	2.983e+02	2.350e+02	3.435e+03	5.658e+03
F_{15}	6.215e+70	1.557e+01	8.771e–09	2.414e+03	1.432e+24
F_{16}	4.267e+05	6.175e+05	7.094e+02	1.216e+04	7.976e+03
F_{17}	4.927e+10	2.927e+11	2.705e+02	9.704e+05	5.943e+04
F_{18}	2.253e+03	8.097e+02	7.229e+01	1.301e+03	2.140e+03
F_{19}	2.765e+06	4.757e+01	9.046e–09	5.039e+03	2.836e+03

Table 10: Results for dimension 500. "Inf" means that no candidate solution was found.

Function	**DEoptim**	`dfoptim _HJKB`	`malsch ains-SW`	**PSO**	`NMOF _PSO`
F_1	3.897e+06	1.217e–09	4.496e+02	1.018e+06	7.858e+05
F_2	1.758e+02	4.363e+01	1.864e+02	1.272e+02	9.742e+01
F_3	3.508e+12	7.968e+11	9.932e+02	4.524e+11	9.417e+10
F_4	1.889e+04	2.388e+02	8.290e+02	1.249e+04	1.719e+04
F_5	3.485e+04	1.178e–01	1.798e+02	9.271e+03	2.960e+04
F_6	2.119e+01	4.430e–06	2.153e+01	2.080e+01	2.081e+01
F_7	Inf	Inf	Inf	Inf	Inf
F_8	8.711e+06	5.036e+07	3.156e+04	5.384e+06	1.198e+06
F_9	8.110e+03	6.516e+03	6.203e+03	7.550e+03	7.721e+03
F_{10}	1.580e+05	1.300e+02	5.039e–01	3.267e+04	1.213e+04
F_{11}	1.088e+04	9.693e+03	5.398e+02	8.429e+03	7.738e+03
F_{12}	2.797e+06	1.208e+06	1.234e+03	4.922e+05	4.083e+05
F_{13}	2.462e+12	2.061e+12	2.078e+03	2.156e+11	3.353e+10
F_{14}	1.423e+04	5.873e+02	6.725e+02	9.017e+03	1.278e+04
F_{15}	Inf	Inf	Inf	Inf	1.040e+120
F_{16}	1.669e+06	1.011e+06	2.618e+03	1.432e+05	1.220e+05
F_{17}	4.686e+11	5.119e+11	1.552e+03	1.158e+09	1.879e+07
F_{18}	5.847e+03	1.294e+03	6.735e+02	3.262e+03	4.941e+03
F_{19}	3.460e+67	1.003e+02	1.680e–01	2.230e+04	2.945e+04

Table 11: Results for dimension 1000. "Inf" means that no candidate solution was found.