The R Package optimization: Flexible Global Optimization with Simulated-Annealing

Kai Husmann **Alexander Lange** **Elmar Spiegel**

Abstract

Standard numerical optimization approaches require several restrictions. So do exact optimization methods such as the Linear Programming approach appeal for linearity and Nelder-Mead for unimodality of the loss function. One method to relax these assumptions is the Simulated Annealing approach, which reduces the risk of getting trapped in a local optimum. However, the standard implementation still requires regular parameter spaces and continuous loss functions. To address this issue, we implemented a version of the Simulated Annealing method that is able to deal with irregular and complex parameter spaces as well as with non-continuous and sophisticated loss functions. Moreover, in order to gain fast but reliable solutions, we included steps to shrink the parameter space during the iterations. All these steps are summarized in the R package **optimization**, which we will introduce in the following article. We also included generic and real world applications in order to test our approach.

Keywords: optimization, Simulated Annealing, simulation, flexible.

1. Introduction

As early computer-based optimization methods developed simultaneously with the first digital computers (Corana, Marchesi, Martini, and Ridella 1987), numerous optimization methods for various purposes are available today (Wegener 2005). One of the main challenges in Operations Research is therefore to match the optimization problem with a reasonable method. The complexity of the problem determines possible methods. Optimization procedures in general can be distinguished into exact methods and heuristics (Kirkpatrick, Gelatt, Vecchi *et al.* 1983). For simple optimization problems, exact methods are often meaningful tools of choice. If all assumptions on model loss and restrictions are met, these methods will obligatorily find the exact solution without need for further parameters. They are the easiest way of solving optimization problems. The *Linear Simplex Method* (Dantzig, Fulkerson, and Johnson 1959) is one example that only needs the loss-function and optional restrictions as model input. If, however, any of the model assumptions, e.g. linearity, is violated, exact methods are unable to solve problems in a valid way. With developing computer power, heuristics like the *Savings-Algorithm* (Clarke and Wright 1964) and metaheuristics like *Simulated Annealing* (SA) (Kirkpatrick *et al.* 1983) became popular. They enable solving more complex optimization problems. Metaheuristics are a generalization of heuristics with aim to be even more flexible and efficient (Blum and Roli 2003) so that they can solve complex optimization problems, such as nonlinear problems. Direct search methods like the *Nelder-Mead* (NM) algorithm are comparatively efficient methods which directly converge to the functions optimum and need relatively few settings (Geiger and Kanzow 1999). Random search methods are able to cope with multimodal objective functions. On the one hand, depending on the method of choice, more or fewer assumptions on the loss function can be neglected. On the other hand, heuristics and metaheuristics will always solve problems approximately. Precision of the solution depends on the optimization method and further parameters. There is even no guarantee

of approximating the actual optimum since the solution also depends, contrary to exact methods, on parameterizations (Blum and Roli 2003). Defining proper parameters is thus a crucial point of those methods. The complexity of parameterization will by trend increase with the flexibility of the method while efficiency tends to decrease. The efficiency and accuracy of such models is strongly sensitive to their parameter specification (Corana *et al.* 1987). Heuristics are often programmed for multi-purpose usage such that there is a suitable method for many optimization problems. For complex optimization problems, however, multi-purpose optimizers often fail to find solutions. Additionally, multi-purpose optimizers are usually not suitable or not efficient for highly complex problems like problems with restricted parameter space. Whenever general-purpose optimizers are too restrictive or inflexible to solve a problem properly, specific methods are advantageous. They offer many variable parameters and can thus be parameterized in a way that is very specific to the optimization problem. They represent the most flexible and the most complex optimization methods (Blum and Roli 2003).

SA (Kirkpatrick *et al.* 1983) is known to be one of the oldest and most flexible metaheuristic methods, though the term metaheuristic was established after publication of SA (Blum and Roli 2003). It is known to be better suited to multimodal loss functions with a very high number of covariates than many other methods (Corana *et al.* 1987). The method has been applied in many studies of several fields such as chemistry (Agostini, Soares-Pinto, Moret, Osthoff, and Pascutti 2006), econometrics (Ingber 1993) or forest sciences (Baskent and Jordan 2002; Boston and Bettinger 1999). Since its first implementation by Kirkpatrick *et al.* (1983), many authors have modified the algorithm in order to adopt it for specific problems (e.g. DeSarbo, Oliver, and Rangaswamy 1989; Goffe *et al.* 1996) or more general applications (e.g. Xiang, Gubian, Suomela, and Hoeng 2013). It combines systematic and stochastic components, and thus enables escaping local optima. It is hence typically used for global optimization of multimodal functions. As it offers many options, SA can be seen a hybrid method between a general optimizer (when default values are chosen) and a problem specific optimization algorithm (Wegener 2005). Corana *et al.* (1987) developed a dynamic adoption method for the variation of the stochastic component during the optimization process. Their modification affects the efficiency as well as the accuracy of the SA algorithm. It has potential to substantially improve the method. Pronzato, Walter, Venot, and Lebruchec (1984) suggests decreasing the search-domain of the stochastic component with increasing number of iterations. The stochastic component in general is the most sensitive part of the method since it determines the loss variables modification during the iterations.

The R software environment provides a platform for the simple and effective distribution of statistical models to a large user community (Xiang *et al.* 2013). Thus, not surprisingly, several optimization packages of high quality can currently be purchased via *Comprehensive R Archive Network* (Theussl and Borchers 2016), where even the SA method is recently listed five times. However, we believe that there is need for a specific stochastic optimization package for complex optimization problems. A package coping with very flexible user definable loss functions with multiple options could be an advantageous extension for R. There is demand for specifically definable optimization problems. We therefore present the package **optimization** which is basically a modified version of SA. We used the properties of SA to program a stochastic optimization method for specific purposes. We therefore focused on flexibility and implemented many user specifiable parameters. Our method is, in its default configuration, usually not immediately efficient but flexibly adoptable to specific purposes. The main advantages of the package are the possibilities to specifically adjust covariate changing rules as well as the robustness of the loss function. For example, the changing rule allows the user to define an integer parameter space. The loss function can return any value; even `NA` or `NaN` are possible. Several further user specifications help to parameterize the model in a problem-specific way, so that the user can influence accuracy and speed in very detailed ways. It is also the first R function where the improvements of Corana *et al.* (1987) and Pronzato *et al.* (1984) are implemented into an SA based optimization software. This means that the search domain of the stochastic component of SA dynamically shrinks

with increasing iterations. We also implemented a generic plot function for post-hoc inspection of the model convergence assessment and the solution quality.

In the following, we briefly introduce the algorithm methodologically and explain the most relevant parameters. We show in four examples in which ways our model is favorable against standard methods and explain how it can be parameterized. We develop two examples illustrating the basic model behavior with a focus on the covariate changing rule. Additionally, we include suggestions on how to adopt the numerous options to specific problems. Two practical examples where our function is recently used underpin the relevance of our specific-purpose optimization method. One of them, the optimization of forest harvesting schedules, is a relatively complex example which cannot be solved with any other optimization function in the R framework.

2. The package optimization

In this section, we explain the theory of our SA interpretation and the resulting parameters.

2.1. Method

Since the basic idea of classic SA is derived from the physical process of metal annealing, the nomenclature of SA comes particularly from metallurgy. Just as the classic SA, our function is composed of an inner for loop and an outer while loop (Kirkpatrick *et al.* 1983). The number of iterations in both loops can be defined by the user. To enhance performance, the loops are written in C++ using **Rcpp** (Eddelbuettel 2013). For better overview, we displayed the important steps of our function in a pseudocode (Algorithm 1).

Inner loop

The function of the inner for loop (Algorithm 1, lines 4 to 29) is to draw covariate combinations stochastically and to compare the returns. The loop repeats n_{inner} times. The first operation of the inner loop (line 5) is saving the covariate combinations of the last inner iteration as x_j. In the first iteration x_j is the vector with user defined initial covariates.

In the second step (line 6), the covariates are changed. This changing process marks an essential difference between our approach and the other SA based method in the R framework. The shape and behavior of the variation process may be defined by the user and may be dynamic. Suggestions and examples for specifying this variation, which is a user defined R function, will be given in the following sections. The variation function *vf* is used to create a temporary vector of covariates x_{i*}. Besides the former covariate combination x_j, the variation function can depend on a vector with random factors *rf* and the temperature *t*. Since *rf* and *t* change over time, the variation function can have dynamic components. Adjustment of *rf* and *t* is done in the outer loop which will be explained explicitly in the following subsection. In the classical SA approach, the covariates x_{i*} are generated by adding or subtracting a uniformly distributed random number to x_j (Kirkpatrick *et al.* 1983). The range of the uniform random number is, in our function, determined by *rf* whereas *rf* is relative to x_j. A random factor of 0.1 and a covariate expression of three e.g. leads to a uniform random number between 2.7 and 3.3. This standard variation function is also default in our approach. A very simple exemplary modification of the variation function could be a normally distributed random number with mean x_j and standard deviation *rf*.

After generating x_{i*}, the boundaries are checked (lines 7 to 15). If all entries in x_{i*} are within their respective boundaries, the response is calculated. Otherwise the invalid entries of x_{i*} are drawn again until all entries are valid. According to Corana *et al.* (1987), the number of invalid trials can be useful

```
1  initialize t, vf with user specifications
2  calculate f(x_0) with initial parameter vector x_0
3  while t > t_min do
4      for i in 1: n_inner do
5          x_j ← x_{i-1}
6          call the variation function to generate x_{i*} in dependence of x_j, rf and t
7          check if all entries in x_{i*} are within the boundaries
8          if all x_i valid then
9              calculate f(x_{i*})
10         else
11             while any(x_{i*} invalid) do
12                 call the variation function again
13                 count and store invalid combinations
14             end
15         end
16         if f(x_{i*}) < f(x_j) then
17             x_i ← x_{i*}; f(x_i) ← f(x_{i*})
18         else
19             calculate Metropolis Probability M (Equation 1)
20             if uniformly distributed random number [0,1] < M then
21                 x_i ← x_{i*}; f(x_i) ← f(x_{i*})
22             else
23                 x_i ← x_j; f(x_i) ← f(x_j)
24             end
25         end
26         if threshold accepting criterion fulfilled then
27             break inner loop
28         end
29     end
30     reduce t for the next iteration
31     rf adaptation for the next iteration
32 end
33 return optimized parameter vector, function value and some additional information
```

Algorithm 1: Pseudocode of the `optim_sa` function in the optimization package exemplary for a minimization.

information in order to assess the quality of the search domain. The numbers of invalid trials are thus counted and stored (line 13) in order to make this information accessible for the outer loop. The count of valid and invalid trials is not reset after each inner loop repetition. Both are only initialized in iteration one of the inner loop and increase until the last iteration.

Next step is the comparison of loss function returns (lines 16 and 17). If the return of current variables combination $f(x_{i*})$ is better than $f(x_j)$, x_{i*} and $f(x_{i*})$ are stored into x_i and $f(x_i)$, so x_i are the initial covariates for the next iteration. The core idea of the classical SA approach is to cope with the problem of local optima. Thus, even if $f(x_{i*})$ is worse than $f(x_j)$, there is a chance of storing x_{i*} into x_i (lines 18 to 25). The likelihood of keeping worse responses depends on the Metropolis probability M (Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller 1953).

$$M = exp\left(-\frac{\mid f(i_*) - f(j) \mid}{kt}\right), \tag{1}$$

with k being a user definable constant. We adopted this strategy from classic SA without modifications. M decreases with decreasing temperature t (Equation 1). The likelihood of keeping worse responses thus depends on the same parameters for the whole inner loop (lines 19 and 24). t does not change during the entire inner loop. The likelihood of keeping worse values is thus equal for each response until the inner loop is completed. Modification of t is part of the outer loop which will be explained in the next paragraph. If a worse result is chosen the former optimal covariate combination is, of course, stored before it is overwritten since otherwise there is a sound chance of overwriting the actual global optimum. More details of the Metropolis probability can i.a. be found in Kirkpatrick *et al.* (1983) and Metropolis *et al.* (1953).

Storing the information on development of covariates and response can help improving the performance of SA (Lin, Haley, and Sparks 1995; Hansen 2012). We implemented a threshold accepting strategy (Dueck and Scheuer 1990) into our SA interpretation (lines 26 to 28). This criterion is the only module that allows reducing the inner loop repetitions without direct user influence. It is simply a vector where the absolute differences of $f(x_i)$ and $f(x_j)$ are stored. If the response oscillates for a user defined number of repetitions within a user defined threshold, the inner loop breaks.

Outer loop

The main functions of the outer while loop (Algorithm 1, lines 3 to 32) are calling the inner loop (lines 4 to 29) and modifying the parameters that are needed in the inner loop (lines 30 and 31). Therefore t and rf only change after completely finishing an inner loop. The outer loop repeats until t is smaller than the user defined minimum temperature t_0.

After finishing the inner loop, firstly t is adjusted (line 30). t is necessary for the stochastic part in the inner loop (line 19, Equation 1). In our function, t decreases per definition as it is calculated by multiplying the temperature of the current iteration by r which is a user defined real number between 0 and 1. The number of outer loop repetitions is thus implied by initial temperature t_0, t_{min} and r.

Afterwards rf changes (line 31). The dynamic adaption of rf after Corana *et al.* (1987) and Pronzato *et al.* (1984) is another major novelty of our function. As each covariate can have its own random factor, rf is a vector of the same size as x_i. rf is needed for the covariate variation in the inner loop (lines 6 and 12). Dividing the numbers of invalid trials distinctively for each covariate by the total number of trials of the respective covariate gives the ratio of invalid trials for each covariate. The numbers of invalid trials are counted in line 13. According to Corana *et al.* (1987), this ratio of invalid trials can be used to find a trade-off between accuracy and the size of the search domain. They argue that if only valid covariate combinations are drawn, the search domain could be too small for multimodal problems. For this, the ratio of invalid trials in the current iteration is used to generate the rf for the following outer

loop repetition. They suggest ratios between 0.4 and 0.6. If any observed ratio of invalid trials is < 0.4 or > 0.6, the respective random factors are modified following the suggested equation by Corana *et al.* (1987). This strategy allows an adjustment of *rf* for the next iteration. Pronzato *et al.* (1984) who developed the *Adaptive Random Search method*, propose a time decreasing search domain. Thus they suggest a search domain adjustment that does not depend on former information, as Corana *et al.* (1987) did, but on the number of iterations. They argue that the search domain should be wide at the beginning to give the algorithm the chance to cope with local optima, and small in the end to allow higher precisions. Later iterations thus require smaller search domains than earlier iterations. We integrated the idea of Pronzato *et al.* (1984) into the dynamic search domain adjustment of (Corana *et al.* 1987) by linearly shrinking the favorable range of ratios from the suggested ratios from {0.4, 0.6} to {0.04, 0.06}. As mentioned in the inner loop explanations, the variation function can be user defined (lines 6 and 12). The user thus has the option to define flexibly in which way *t* and *rf* influence the covariate variation. Per default, the search domain around the covariates shrinks by trend as the number of outer loop iterations increases.

As `optim_sa` shall be able to solve very specific optimization problems, several parameters can be defined by the user. The quality of solution and speed of convergence will thus substantially depend on accurate parametrization. In the following, we will explain the most important parameters briefly and make suggestions for useful specification. A complete parameter list can be found in the package help.

- `fun`: Loss function to be optimized. The statement is without default. The function must depend on a vector of covariates and return one numeric value. There are no assumptions on covariates and return. The covariates do not even need to be continuous. Missing (`NA`) or undefined (`NaN`) returns are also allowed. Any restriction on the parameter space, e.g. specific invalid covariate values within the boundaries, can be directly integrated into the loss function by simply returning `NA`. We will include more specific information on this in the practical examples.

- `start`: Numeric vector with initial covariate combination. This statement has no default. It must be ensured that at least the initial covariate combination leads to a defined numeric response. The loss function at the initial variables combination must therefore return a defined numeric value. This might be relevant when the starting values are determined stochastically.

- `trace`: If `TRUE`, the last inner loop iteration of each outer loop iteration is stored as a row in the trace matrix. This might help evaluating the solutions quality. However, storing interim results increases calculation time by up to 10 %. Disabling `trace` can thus improve efficiency when the convergence of an optimization problem is known to be stable.

- `lower, upper`: Numeric vector with lower boundaries of the covariates. The boundaries are needed since the dynamic `rf` adjustment (Corana *et al.* 1987; Pronzato *et al.* 1984) depends on the number of invalid covariate combinations.

- `control`: A list with optional further parameters.

All parameters in the list with `control` arguments have a default value. They are pre-parameterized for loss functions of medium complexity. `control` arguments are:

- `vf`: Variation function that allows the user to restrict the parameter space. This is one of the most important differences to classic SA. The function determines the variation of covariates during the iterations. It is allowed to depend on `rf`, `temperature` and the vector of covariates of the current iteration. The variation function is a crucial element of `optim_sa` which enables flexible

programming. It is (next to the loss function itself) the second possibility to define restrictions. The parameter space of the optimization program can be defined by `vf`. Per default, the covariates are changed by a continuous, uniformly distributed random number. It must be considered that defining specific `rf` can increase the calculation time. The default `rf` is a compiled C++ function whereas user specified `rf` must be defined as R functions. User specified `rf` are e.g. useful for optimization problems with non-continuous parameter space.

- `rf`: Numeric vector with random factors. The random factors determine the range of the random number in the variation function `vf` relative to the dimension of the function variables. The `rf` can be stated separately for each variable. Default is a vector of ones. If `dyn_rf` is enabled, the entries in `rf` change dynamically over time.

- `dyn_rf`: Boolean variable that indicates if the `rf` shall change dynamically over time to ensure increasing precision with increasing numbers of iterations. `rf` determines whether the adjustments of Corana *et al.* (1987) and Pronzato *et al.* (1984) are enabled (see method section for theoretical background). `dyn_rf` ensures a relatively wide search domain at the beginning of the optimization process that shrinks over time. Disabling `dyn_rf` can be useful when `rf` with high performance are known. The development of `rf` is documented in the `trace` matrix. Evaluation of former optimizations with dynamic `rf` can thus help finding efficient and reasonable fixed `rf`. Self-specified `vf` may not depend on `rf`. In this cases activating `dyn_rf` will not have any advantage.

- `t0`: Initial temperature. The temperature directly influences the likelihood of accepting worse responses and thus the stochastic part of the optimization. `t0` should be adopted to the loss function complexity. Higher temperatures lead to a higher ability of coping with local optima but also to more time-consuming function calls.

- `t_min`: Numeric value that determines the temperature where outer loop stops. As there is practically no chance of leaving local optima in iterations with low temperature `t_min` mainly affects accuracy of the solution. Higher `t_min` yields to lower accuracy and fewer function calls.

- `nlimit`: Integer value which determines the maximum number of inner loop iterations. If the break criterion in the inner loop is not fulfilled, `nlimit` is the exact number of inner loop repetitions. It is therefore an important parameter for determining the number of iterations.

- `r`: Numeric value that determines the reduction of the temperature at the end of each outer loop. Slower temperature reduction leads to an increasing number of function calls. It should be parameterized with respect to `nlimit`. High `nlimit` in combination with low `r` lead to many iterations with the same acceptance likelihood of worse responses. Low `nlimit` in combination with `r` near 1, by contrast, lead to a continuously decreasing acceptance likelihood of worse responses. It is thus the second crucial parameter for determining the number of iterations.

3. Examples

To show the benefits of our optimization package, we build four examples explaining where the `optim_sa` function can be advantageous.

3.1. Himmelblau Function with continuous parameter space

Himmelblau's function (Equation 2) (Himmelblau 1972) was chosen as an initial example since it is a very simple multimodal equation and widely known in operations research. It has four equal minimum values ($min(f(x_1,x_2)) = 0$)) at {-2.8, 3.1}, {3.0, 2.0}, {3.6, -1.8} and {-3.8, -3.3}. In order to display the basic behavior of `optim_sa`, it was compared with two other SA methods from the **stats** package (R Core Team 2017). Himmelblau's function is relatively simple. Therefore we also included the NM optimization method (Nelder and Mead 1965) which is a default of **optim** from the stats package, to examine the advantages of stochastic search against direct search.

$$f(x_1,x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{2}$$

We performed 10,000 repetitions with each function in order to investigate the quality and speed of the solutions using parameters for relatively simple optimization problems for all examined methods. The optimizations were performed having the following parameters:

```
R> # stats package: optim (NM)
R> stats::optim(fn = hi, par = c(10, 10), method = "Nelder-Mead")
R> # optimization package: optim_sa
R> optimization::optim_sa(fun = hi, start = (c(10, 10)), trace = TRUE,
+          lower = c(-40, -40), upper = c(40, 40),
+          control = list(t0 = 500, nlimit = 50, r = 0.85,
+                  rf = 3, ac_acc = 0.1, dyn_rf = TRUE))
R> # stats package: optim (SA)
R> stats::optim(fn = hi, par = c(10, 10), method = "SANN",
+          control = list(tmax = 500, reltol = 0.1, temp = 50, trace = TRUE))
```

Since we have a multimodal optimization problem with multiple equal solutions, the evaluation of solutions quality is composed of response accuracy and covariate combination. With fixed starting parameters, the methods should be able to find all possible solutions. We parameterized the functions from the example such that thy fulfill a defined accuracy threshold. We defined the optimization problem to be solved properly when the responses of the 10,000 repetitions were in mean ≤ 0.01. We also looked at the frequency distribution of the covariate combination after minimization. Subsequent to the investigation of quality, we also compared the efficiencies by measuring calculation times using **microbenchmark** (Mersmann 2015) and the iterations frequencies.

It became clear that parameterization of NM was quite simple. It only needed a vector with starting values. The other functions required a larger number of settings. With view to accuracy each method performed well. All functions returned in mean of 10,000 calls responses with values ≤ 0.01. Regarding frequency distribution, the functions performed differently (Table 1). `optim_sa` and `optim (SA)` returned all possible solutions. The combination {-3.8, -3.3} was consistently least frequent. As `optim (NM)` is a direct search method, it only returned one solution {-3.8, -3.3}. Further investigation revealed the solutions of `optim (NM)` to be sensitive to the starting values. If `optim (NM)` was parameterized with randomly drawn starting values, all four results would have been possible. Thus one advantage of `optim_sa` and `optim (SA)` against direct search methods, like `optim (NM)`, is its independence from starting values. Random search methods are advantageous for solving multimodal optimization problems.

As all functions were practically able to minimize Equation 2, comparison of calculation times appeared to be another important point for quality assessment. As expected, the direct search method `optim (NM)` was by far faster than all stochastic methods (Figure 1, left). The two functions able to cope with equally valued optima (`optim (SA)` and `optim_sa`; Table 1) were significantly slower (Figure 1, left). We parameterized the functions such that they fulfill our accuracy requirements. The

Table 1: Relative frequencies of covariate combinations in % after optimization of Example 5.1 for the four examined methods. Number of repetitions: 10,000. We used the parameters given in the example, only the `trace` options was deactivated.

		result (rounded)			
		{-2.8, 3.1}	{3.0, 2.0}	{3.6, -1.8}	{-3.8, -3.3}
method	optim_sa	22.19	33.49	28.05	16.27
	optim (SA)	25.91	30.89	24.08	19.12
	optim (NM)	0.00	0.00	0.00	100.00

parameterizations in the examples should thus be near to the most efficient parameterizations for the given problem with the expected accuracy.

Another way of comparing algorithms' efficiencies is to examine the frequency of necessary iterations solving the optimization problem. Again, `optim (NM)` performed best (Figure 1, right). `optim (SA)` required the most repetitions by far. Only `optim_sa` showed varying frequencies because `optim_sa` was, given the displayed parameters, the only function where the inner loop broke due to the threshold accepting criterion. `optim (SA)` always iterated till the user specified limit. It could be seen that `optim_sa` performed well when compared to established SA based optimization function. `optim_sa` is slower but it requires less iterations. The calculation of valid parameter combinations (Algorithm 1, lines 7 to 15) leads to multiple function calls within one iteration. For this, one iteration of `optim_sa` can last longer than one iteration of the other methods. The number of function calls is not necessarily equal to the number of iterations when `dyn_rf` is activated. The `dyn_rf` which adopts the search domain dynamically to specific loss functions on the one hand thus leads to longer calculation times on the other hand. Improved flexibility therefore again corresponds with longer calculation time.

To conclude, all functions were generally able to solve the problem. The flexible stochastic search-grid of `optim (SA)` and `optim_sa` enabled archiving each of the four solutions. Practically, if users are in doubt whether a problem has multiple solutions with equal responses, `optim (SA)` and `optim_sa` can simply be repeated without re-parameterization. If well-specified, they will return all possible solutions. They thus have advantages for complex multimodal problems of that kind but are also slower. Due to the multiple additional options, which all have to be called in the code, `optim_sa` is slowest. This example clearly reveals that higher flexibility and generality leads to significantly higher calculation times. Specific algorithms have advantages for more complex problems while more general functions are useful for optimization problems with simple responses.

3.2. Himmelblau Function with discrete parameter space

A second example illustrating the advantage of flexibly defining the parameter spaces also bases on the function by Himmelblau (1972). It is an exemplary optimization problem which cannot be solved with any other examined method. If a user is interested in integer covariate combinations only, simply rounding the solution is usually not satisfying since the solution could be non-feasible or non-optimal (Cardoso, Salcedo, de Azevedo, and Barbosa 1997). The flexible variation function `vf` of our models allows searching for integer temporary solutions only and thus searching for the global integer solution. `vf` can be used to restrict the parameter space to the specific problem. The following simple example shows a variation rule for an exemplary integer programming problem. The simple `var_fun_int` from the following example returns a vector of integer covariates varying in dimension of `vf` around the passed covariates `para_0`. The `fun_length` must be passed explicitly, although it is implicitly given by the length of `para_0` because it might help simplifying the function `vf`. Dependence of `rf`

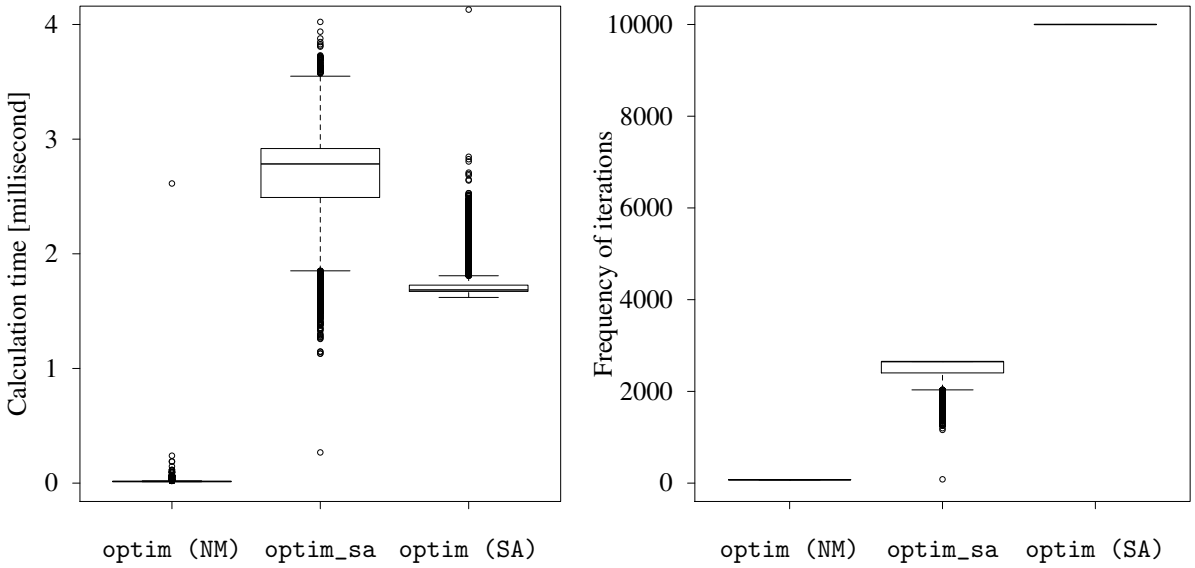

Figure 1: Calculation times and frequencies of iterations of the four examined optimization algorithms in Example 5.1. Note that the y-axis of the left diagram is truncated for reasons of presentation. `optim_sa` showed 101 and `optim (SA)` 70 outliers between four and seven milliseconds. The frequency of iterations represents the total number of iterations. Thus, for the SA methods, all inner loops repetitions are counted. The packages parameterization are shown in the example code.

and `temp` can be implemented. Optimization problems do not always need a dynamic component. If `vf` need not be dynamic, i.e. if the stochastic part shall be fixed during the entire iterations, `temp` and `rf` can separately be disabled by passing `NA` for the respective parameter. If `vf` does not depend on `rf`, the option `dyn_rf` will not be useful any more. It should be disabled in such cases to save computation time.

```
R> # stats package: optim (NM)
R> stats::optim(fn = hi, par = c(10, 10), method = "Nelder-Mead")
R> # optimization package: optim_sa
R> optimization::optim_sa(fun = hi, start = (c(10, 10)), trace = TRUE,
+          lower = c(-40, -40), upper = c(40, 40),
+          control = list(t0 = 500, nlimit = 50, r = 0.85,
+                  rf = 3, ac_acc = 0.1, dyn_rf = TRUE))
R> # stats package: optim (SA)
R> stats::optim(fn = hi, par = c(10, 10), method = "SANN",
+          control = list(tmax = 500, reltol = 0.1, temp = 50, trace = TRUE))
```

Repeating the minimization function with the parameters from the example 10,000 times led to the only integer solution {3, 2} in 95.6 % of the repetitions. The generic plot function of the optimization package (Figure 2) helps interpreting the convergence and thus the solution quality as well as the algorithm behavior. Since the example is 2-dimensional, a `contour_plot` (Figure 2, right) could be created. It shows the state space of the Himmelblau function in continuously changing colors for parameter spaces between -4 and 4. The results at the end of each outer loop iteration are shown as points within the parameter space. It became obvious that only integer covariate combinations were calculated during the entire optimization (Figure 2, right). Figure 2 additionally helps explaining the stochastic part of the SA method. It becomes clear that though the response of iteration 10 (parameter combination {0, 0}) was already quite near 0, following iteration ended with relatively worse intermediate results. With response > 100, iteration 19 (parameter combination {2, -3}) was much worse

than iteration 10. The likelihood of keeping worse solutions shrank over time till it became practically 0 after iteration 40 (Figure 2, left). For problems of such kind, 40 iterations therefore should be far enough. This information could help the user parameterizing the function for the next problem of such kind.

`optim (SA)` also provides possibility of defining a variation function via the `gr` statement. Using the variation function from the example code with fixed `rf`, however, it was not possible to define robust parameters for solving an integer programming problem. `optim_sa` is thus advantageous against all other examined functions when the parameter space of the optimization problem underlies extended restrictions. It is the only function that enables global optimum integer search. To achieve this, the user must define a changing rule in form of a R function. Users must thus examine their optimization problem very carefully and translate it into suitable functions. Parameterization of `optim_sa` is quite complex and time extensive but enables adopting the optimizer to problem specific needs. Other restrictions such as mixed integer problems may be defined analogously. `vf` can thus be used to flexibly restrict the parameter space. Further suggestions for interpretation can be found in the package documentation.

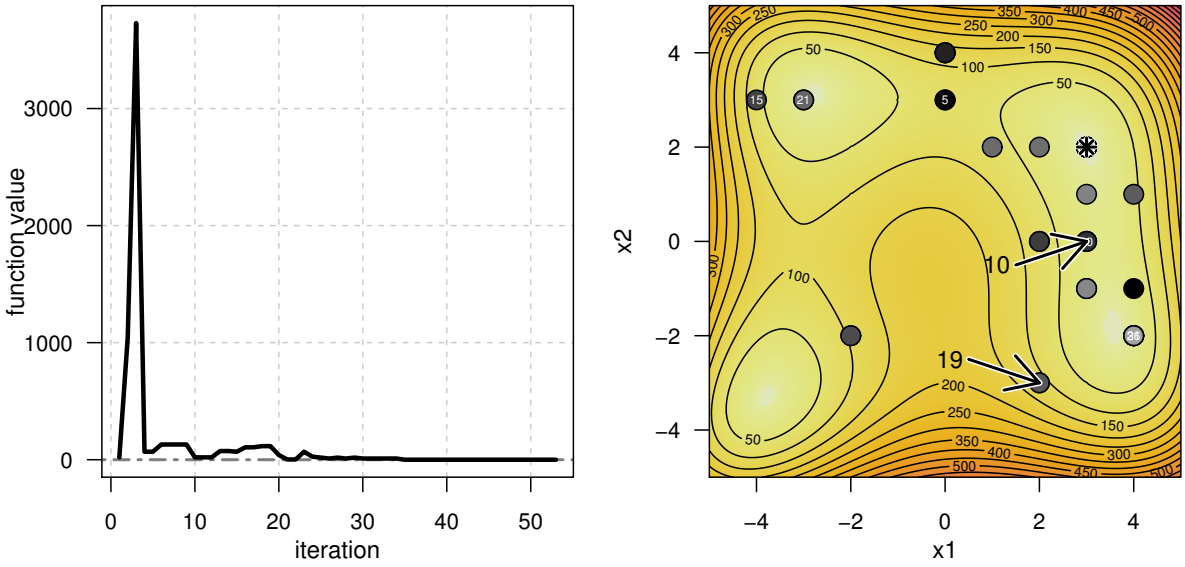

Figure 2: Exemplary examination plots created with the generic plot function. The left diagram shows the current optimal response over iteration of the outer loop. The left diagram displays the succession of the covariate values. The star points the covariate combination at optimum. The actual parameter space of the optimization is reduced for presentation purposes.

3.3. Structural vector autoregressive models with least dependent innovations

Vector autoregressive (VAR) models are used to examine dynamic structures of several time series endogenously within one model. Usually, impulse response functions come to use for investigating structural shocks in those multivariate time series systems. Those impulse responses need a decomposition of the covariance matrix. The covariance matrix decomposition, however, is not unique. The interaction between the time series variables in a VAR model are not directly calculable without further assumptions or information. The idea behind *structural* VAR (SVAR) is to define strategies to overcome this problem and enable unique impulse response definitions (Lütkepohl 2006).

Least dependent innovations is one of plenty possibilities to obtain unique decompositions of the covariance matrix. The idea is to minimize a Cramer-von-Mises test statistic for stochastically inde-

pendent structural errors (Genest, Quessy, and Remillard 2007). Basically, the structural errors are multiplied with $K(K-1)/2$ Givens rotation matrices with angles θ between 0 and 2π, until the test statistic from equation 3 becomes minimal. K is the number of time series in the SVAR model, T the number of observations and $\tilde{\varepsilon}$ are the rotated structural errors. An advantage of this method is, that it is very liberal in sense of distribution assumptions (Herwartz 2014).

$$\mathscr{B}_\theta = \int_{(0,1)^K} \left[\sqrt{T} \left(C(\tilde{\varepsilon}) - \prod_{i=1}^{K} U(\tilde{\varepsilon}_i) \right) \right]^2 d\tilde{\varepsilon} \tag{3}$$

However, optimizing the test statistic is challenging, because the loss function has a large and unpredictable number of local optima and an irregular state space. Moreover, the function shows a relatively broad parameter space with very sensitive responses. It often reacts dramatically to small changes in the rotation angles. One approach to solve the optimization problem is the *grid optimization* (Herwartz and Plödt 2015). Grid optimization is, however, inefficient in large samples and – depending on the step size – sometimes unable to find the global optimum. Monte Carlo simulations have shown that direct search and gradient based optimization algorithms usually fail to find the optimal rotation angles. Performing several direct search optimizations sequentially with randomly changing starting values can be lead to satisfying solutions. The strategy is, however, also very inefficient.

The irregular response pattern and the high demands on the solution precision are the main challenges. The combination of SA with the adaptive variation of the search seems to be promising for the optimization problem. The relatively broad parameter variation at the beginning of the optimization process should ensure adequate search-grid size while the stochastic component of the SA should be able to tackle the irregular responses. Dynamically decreasing parameter variation is particular advantageous for optimizing the rotation angles as sufficient parameter accuracy is crucial for the reliability of the solution.

Practical application of `optim_sa` revealed its advantages. Using parameterizations for low performance and high precision, like an initial temperature of 100,000, a maximum number of 10,000 inner loops and temperature reductions between 0.7 and 0.9, still converged faster than all other investigated methods. Sensitivity analysis of exemplary optimization problems confirmed the general applicability of `optim_sa`. The function was able to find the actual global optimum sufficiently often.

3.4. Forest harvesting schedule optimization

Forestry is traditionally a knowledge-based field with optimization playing only a minor role. However, optimization methods are very interesting for intermediate-term forest planning. Their popularity is increasing (Möhring 2010). While Linear Programs are nowadays used in some states, e.g. Finland (Redsven, Hirvelä, Härkönen, Salminen, and Siitonen 2012), stochastic optimization programs are quite novel in forestry for optimization of harvesting intensity (Kangas, Kurttila, Hujala, Eyvindson, and Kangas 2015). Our function is an integral part of the first stochastic optimization software of forest operation in Germany and one of the first software solutions worldwide on single tree scale. Optimization of forest harvesting planning represents an interesting and innovative practical example where `optim_sa` is recently used. The `optim_sa` function is part a innovative forest development simulation-optimization software for decision support of forest enterprises. The software is an optimization add-on for the widely-used forest development simulation software *WaldPlaner* by Hansen and Nagel (2014). WaldPlaner is a user front end for *Tree Growth Open Source Software* (*TreeGrOSS*), which is a complex Java written tree growth and yield simulation software used to forecast the developments of forest management areas (forest stands) developed by Nagel (1996). It is a tool able to simulate forest enterprises with hundreds of forest stands simultaneously where the smallest simulation element is the single tree. Each tree in the system is simulated individually. Optimization of

forest activities is, accordingly, not trivial since TreeGrOSS is a complex network of rules and functions which are predominantly nonlinear. The entire optimization process is composed of TreeGrOSS, `optim_sa`, an interface function to enable communication between TreeGrOSS and the optimizer as well as a data warehouse. This multi-factorial composition implies high demand for flexibility of the optimization function. The loss function, which represents in this example the interface between the three elements of the optimization system, must be composed of R, Java (using **rJava**; Urbanek 2016) and SQL (using **RPostgreSQL**; Conway, Eddelbuettel, Nishiyama, Prayaga, and Tiffin 2016). Main tasks of the interface are enabling communication between TreeGrOSS and optimization algorithm and rating the TreeGrOSS output in terms of costs and revenue such that the TreeGrOSS output is translated into a response readable by the optimization algorithm. Flexibility of loss and variation functions is hence a prerequisite for forest harvesting schedule optimization via TreeGrOSS simulations. Each loss function call causes a TreeGrOSS simulation and a database operation. In order to save time, parts of the loss are therefore programmed parallel. The concept of sustainability plays a central role in forestry. Forests must be treated such that recent and further generations can benefit from their economic, ecologic and social functions. Particular harvesting operations must not exceed the sustainable wood growth. Each function call returns, next to the actual response, a sustainability index. This index is used to restrict the optimization by returning `NA` responses whenever sustainability is violated. Sustainability is hence an important restriction forest schedule optimization that further increases complexity of the loss function as the loss partially returns invalid values.

The example again reveals the complexity of the problem and further reinforces the need for specific stochastic methods. Only `optim_sa` was able to solve the complex loss function of the forest harvesting schedule problem. Sensitivity analysis using an exemplary forest enterprise comprised of five forest sites, with known global maximum, reinforced reliability of `optim_sa` for harvesting optimization. A solution sufficiently near the global maximum was found in arguable time on a standard personal computer. To test the practical usability, the optimization system was additionally tested on a real forest enterprise with 100 forest sites. The problem was solved using a high performance cluster computer. The simulation-optimization approach calculated reasonable and reliable solutions.

4. Discussion and outlook

In conclusion, SA methods have considerable advantages over classical optimization procedures if the loss function is non-linear, multimodal and partially undefined. Our simulated annealing approach optimizes these functions quite well. It might not be faster than the standard simulated annealing approach of the stats package (R Core Team 2017), but that was not its purpose. However, due to the shrinkage of the parameter space and the extra stopping criteria it manages to require fewer steps than the standard approach. Additionally, the main advantage of our new approach is to deal with irregular parameter spaces, as well as with non-continuous and sophisticated loss functions, where the standard approach reaches its limits. In our examples we show that problems with these characteristics exist and that our algorithm solves these problems. Furthermore there are several examples of loss functions in natural science where some combinations of variables are restricted and our approach shows its benefits.

Acknowledgements

The authors are grateful to the Project Management Jülich of the Federal Ministry of Education and Research for funding this research through the Cluster *Bio-Economy* (031A294). We thank Michael Hill for reviewing the language of the manuscript.

References

Agostini FP, Soares-Pinto DDO, Moret MA, Osthoff C, Pascutti PG (2006). "Generalized simulated annealing applied to protein folding studies." *Journal of computational chemistry*, **27**(11), 1142–1155.

Baskent EZ, Jordan GA (2002). "Forest landscape management modeling using simulated annealing." *Forest Ecology and Management*, **165**(1), 29–45.

Blum C, Roli A (2003). "Metaheuristics in combinatorial optimization: Overview and conceptual comparison." *ACM Computing Surveys (CSUR)*, **35**(3), 268–308.

Boston K, Bettinger P (1999). "An analysis of Monte Carlo integer programming, simulated annealing, and tabu search heuristics for solving spatial harvest scheduling problems." *Forest science*, **45**(2), 292–301.

Cardoso MF, Salcedo R, de Azevedo SF, Barbosa D (1997). "A simulated annealing approach to the solution of MINLP problems." *Computers & chemical engineering*, **21**(12), 1349–1364.

Clarke G, Wright JW (1964). "Scheduling of vehicles from a central depot to a number of delivery points." *Operations research*, **12**(4), 568–581.

Conway J, Eddelbuettel D, Nishiyama T, Prayaga SK, Tiffin N (2016). *RPostgreSQL: R interface to the PostgreSQL database system*. R package version 0.4-1, URL https://CRAN.R-project.org/package=RPostgreSQL.

Corana A, Marchesi M, Martini C, Ridella S (1987). "Minimizing Multimodal Functions of Continuous Variables with the "Simulated Annealing" algorithm." *ACM Trans. Math. Softw.*, **13**(3), 262–280.

Dantzig GB, Fulkerson DR, Johnson SM (1959). "On a linear-programming, combinatorial approach to the traveling-salesman problem." *Operations Research*, **7**(1), 58–66.

DeSarbo WS, Oliver RL, Rangaswamy A (1989). "A simulated annealing methodology for clusterwise linear regression." *Psychometrika*, **54**(4), 707–736.

Dueck G, Scheuer T (1990). "Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing." *Journal of computational physics*, **90**(1), 161–175.

Eddelbuettel D (2013). *Seamless R and C++ Integration with Rcpp*. Springer, New York. ISBN 978-1-4614-6867-7.

Geiger C, Kanzow C (1999). "Das Nelder-Mead-Verfahren." *Numerische Verfahren zur Lösung unregestrierter Optimierungsaufgaben*.

Genest C, Quessy JF, Remillard B (2007). "Asymptotic local efficiency of Cramer-von Mises tests for multivariate independence." *The Annals of Statistics*.

Goffe WL, *et al.* (1996). "SIMANN: A global optimization algorithm using simulated annealing." *Studies in Nonlinear Dynamics and Econometrics*, **1**(3), 169–176.

Hansen J, Nagel J (2014). *Waldwachstumskundliche Softwaresysteme auf Basis von TreeGrOSS: Anwendung und theoretische Grundlagen*. Beiträge aus der Nordwestdeutschen Forstlichen Versuchsanstalt / Nordwestdeutsche Forstliche Versuchsanstalt. - Göttingen : Univ.-Verl. Göttingen, 2007- ; 11Universitätsdrucke Göttingen. Univ.-Verl., Göttingen. ISBN 978-3-86395-149-8.

Hansen JH (2012). *Modellbasierte Entscheidungsunterstützung für den Forstbetrieb: Optimierung kurzfristiger Nutzungsoptionen und mittelfristiger Strategien unter Verwendung metaheuristischer Verfahren und parallelen Rechnens*. 1. aufl. edition. Cuvillier, Göttingen.

Herwartz H (2014). "Structural analysis with independent innovations." *cege Discussion Papers*.

Herwartz H, Plödt M (2015). "The macroeconomic effects of oil price shocks: Evidence from a statistical identification approach." *Journal of International Money and Finance*.

Himmelblau DM (1972). *Applied nonlinear programming*. McGraw-Hill Companies.

Ingber L (1993). "Simulated annealing: Practice versus theory." *Mathematical and computer modelling*, **18**(11), 29–57.

Kangas A, Kurttila M, Hujala T, Eyvindson K, Kangas J (2015). *Decision Support for Forest Management*. Number 30 in Managing Forest Ecosystems, 2nd ed edition. Springer International Publishing, Cham.

Kirkpatrick S, Gelatt CD, Vecchi MP, *et al.* (1983). "Optimization by simulated annealing." *science*, **220**(4598), 671–680.

Lin C, Haley K, Sparks C (1995). "A comparative study of both standard and adaptive versions of threshold accepting and simulated annealing algorithms in three scheduling problems." *European Journal of Operational Research*, **83**(2), 330–346.

Lütkepohl H (2006). "Structural vector autoregressive analysis for cointegrated variables." In *Modern Econometric Analysis*, pp. 73–86. Springer.

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

Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953). "Equation of state calculations by fast computing machines." *The journal of chemical physics*, **21**(6), 1087–1092.

Möhring B (2010). "Optimierung forstlicher Produktion unter Beachtung von finanziellen Restriktionen | Optimisation of forest production under financial restrictions." *Schweizerische Zeitschrift fur Forstwesen*, **161**(9), 346–354. ISSN 0036-7818. doi:10.3188/szf.2010.0346. URL http://szf-jfs.org/doi/abs/10.3188/szf.2010.0346.

Nagel J (1996). "Anwendungsprogramm zur Bestandesbewertung und zur Prognose der Bestandesentwicklung." *Forst und Holz*, **51**(3), 76–78.

Nelder J, Mead R (1965). "A simplex method for function minimization." *Computer Journal*, **7**(4), 308–313.

Pronzato L, Walter E, Venot A, Lebruchec JF (1984). "A general-purpose global optimizer: Implimentation and applications." *Mathematics and Computers in Simulation*, **26**(5), 412–422. ISSN 03784754. doi:10.1016/0378-4754(84)90105-8.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Redsven V, Hirvelä H, Härkönen K, Salminen O, Siitonen M (2012). *MELA2012 Reference Manual (2nd edition)*. The Finnish Forest Research Institute.

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

Urbanek S (2016). *rJava: Low-Level R to Java Interface*. R package version 0.9-8, URL https://CRAN.R-project.org/package=rJava.

Wegener I (2005). "Simulated annealing beats metropolis in combinatorial optimization." In *International Colloquium on Automata, Languages, and Programming*, pp. 589–601. Springer.

Xiang Y, Gubian S, Suomela B, Hoeng J (2013). "Generalized Simulated Annealing for Global Optimization: the GenSA Package." *The R Journal Volume 5(1):13-29, June 2013*.

Affiliation:

Kai Husmann
University of Goettingen
Department of Forest Economics and Forest Management
Buesgenweg 3
37077 Goettingen, Germany
E-mail: khusman1@gwdg.de

Alexander Lange
University of Goettingen
Chair of Econometrics
Humboldtallee 3
37073 Goettingen, Germany
E-mail: alexander.lange@uni-goettingen.de

Elmar Spiegel
University of Goettingen
Chair of Statistics
Humboldtallee 3
37073 Goettingen, Germany
E-mail: espiege@uni-goettingen.de