Flexible Global Optimization with Simulated-Annealing

by Kai Husmann, Alexander Lange (and Elmar Spiegel)

Abstract An abstract of less than 150 words.

Introduction

As early computer-based optimization methods developed contemporaneously with the first digital computers (Corana et al., 1987), nowadays numerous optimization methods for various purposes are available (Wegener, 2005). One of the main challenges in Operations Research is therefore to match the optimization problem with a reasonable method. According to Kirkpatrick et al. (1983), optimization procedures in general can be distinguished into exact methods and heuristics. In case the loss function of the optimization problem shows quite simple responses, 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. The *Linear Simplex-Method* (Dantzig et al., 1959) as an example only needs the loss-function and optional restrictions as model input. If, however, any of the model assumptions, e. g. linearity or unimodality, is violated, exact methods are unable to solve the problems validly. In practice, they thus lack applicability whenever a loss function is too complex. With developing computer power heuristics like the Savings-Algorithm (Clarke and Wright, 1964) and metaheuristics became more and more popular. Both enable solving optimization problems of more flexible loss functions. Metaheuristics are a generalization of heuristics with aim to be even more flexible and efficient (Blum and Roli, 2003). Depending on the method of choice, more or less 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 loss, optimization method and further parameterizations of the respective method. There is even no guaranty of approximating the actual optimum since the solution also depends, by contrast to exact methods, on parameterization (Blum and Roli, 2003). Heuristic and metaheuristic methods will thus always need additional parameters next to the loss function. Finding the proper model parameters is thus a crucial point when applying heuristic or metaheuristic algorithms. The complexity of parameterization will by trend increase with flexibility of the method while efficiency trends to decrease. Direct search methods like the *Nelder-Mead* (NM) algorithm are comparatively efficient methods which directly converge to the functions optimum and need relatively less settings (Geiger and Kanzow, 1999). Random search methods are able to cope with multimodal objective functions as they have a random component which allows leaving local optima. The efficiency and accuracy of such models is usually highly sensitive to their parameter specification (Corana et al., 1987).

Simulated Annealing (SA) (Kirkpatrick et al., 1983) is known to be one of the oldest and one of the most flexible and generalized metaheuristic method (Blum and Roli, 2003), though the term 'metaheuristic' was established after first publication of SA. It is known to be favorable against most other methods for multimodal loss functions with a very high number of covariates (Corana et al., 1987). The method was applied in many studies among several fields covering e. g. chemistry (Agostini et al., 2006), econometrics (Ingber, 1993) or forest sciences (Baskent and Jordan, 2002). 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 et al., 1989; Goffe et al., 1996) or make it more efficient (e. g. Xiang et al., 2013). It is basically a combination of systematic and a stochastic components. The stochastic part of the algorithm allows keeping worse responses in each iteration step during the optimization process. This enables searching for the global optimum even for multimodal loss functions. As it offers a lot of options, SA can be named as hybrid method between a general optimizer (when default values are chosen) and a problem specific optimization program (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 et al. (1984) suggest to decrease the search-domain of the random component with increasing number of iterations. The stochastic component in general is the most sensitive part of the method since it actually determines the loss variables modification during the iterations.

The R software environment provides platform for simple and effective distribution of statistical models to a huge 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 5 times. We, however, believe, a package which copes with very flexible loss functions and rules for the stochastic component could

be an advantageous extension for R. We present the package **optimization** where we implemented a modified version of SA. It is, as far as we know, the first function that combines SA with the extensions of Corana et al. (1987) and Pronzato et al. (1984). Main difference of our approach to existing random-search methods is its flexibility. It allows several user specifications which help to parameterize the model very problem-specific. Next to the high number of parameters that may influence accuracy and speed of the solution, remarkable novelties of our model are the ability of flexibly defining loss function as well as covariate changing rules. The user can thus directly influence the stochastic part of the algorithm. Visual post-hoc inspection of the model convergence facilitate assessment of the solution quality. We show in practical examples where our model can be useful and how it can be parameterized.

Method

Simulated Annealing and its derivations are well document in the scientific literature (e. g. in Hansen, 2012; Kirkpatrick et al., 1983; Xiang et al., 2013). We thus mainly concentrate in our methods description on modified and novel implementations. The classic SA is composed of an inner for and an outer while loop (Kirkpatrick et al., 1983). Since the basic idea of Simulated Annealing is derived from the physical process of metal annealing, the nomenclature of SA comes particularly from metallurgy. The number of iterations in both loops are user predefined.

initialize t, vf with user specifications
calculate $f(x_0)$ with initial parameter vector x_0
while $t > t_{min}$ **do**
  for *i in c(1:* n_{inner}*)* **do**
    $x_j \leftarrow x_{(i-1)}$
    call the variation function to generate x_{i*} in dependence of x_j, rf and t
    check if all entries in x_{i*} are within the boundaries
    if *all* x_i *valid* **then**
      calculate $f(x_{i*})$
    else
      while *any(*x_{i*} *invalid)* **do**
        call the variation function again
        count invalid combinations
      end
    end
    if $f(x_{i*}) < f(x_j)$ **then**
      $x_i \leftarrow x_{i*}; f(x_i) \leftarrow f(x_{i*})$
    else
      calculate Metropolis Probability in dependence of t
      if *random number* $<$ *Metropolis probability* **then**
        store x_j and $f(x_j)$
        $x_i \leftarrow x_{i*}; f(x_i) \leftarrow f(x_{i*})$
      else
        $x_i \leftarrow x_j; f(x_i) \leftarrow f(x_j)$
      end
    end
    if *threshold accepting criterion fulfilled* **then**
      break inner loop
    end
  end
  reduce t for the next iteration
  vf adaptation for the next iteration
end
return optimized parameter vector, function value and some additional informations

Algorithm 1: Pseudocode of the modified Simulated Annealing method in the **optimization** package exemplary for a minimization.

The inner loop of Algorithm 1 is a Markov-Chain in which the responses of different covariate combinations are calculated and compared. The loop is repeated n_{inner} times. After saving the covariate combination of the last inner iteration as x_j, the variation function is called to create a new temporary combination x_{i*}. In the classical SA approach, the covariates are changed by a uniform distributed random number around x_j (Kirkpatrick et al., 1983) which is also the default in our function. The variation function can be changed by the user. It is allowed to depend on a vector with random factors rf, x_j and the temperature t. By default, the entries in rf determine the lower and upper limit

of the uniformly distributed random number relative to the current expression of the covariate. A random factor of 0.1 and a covariate expression of 3 for x_j e.g. leads to a random number between 2.7 and 3.3 for x_{i*} if the covariate vector has one entry. The dynamic adaption of rf after Corana et al. (1987) and Pronzato et al. (1984) is one of the major novelties of our function. If all entries in x_{i*} after their variation are within the allowed 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 used for dynamical adjustment of the rf. The numbers of invalid trials are thus, distinctively for each covariate, counted and stored. If the return of current variables combination $f(x_{i*})$ is better than $f(x_j)$, x_{i*} and $f(x_{i*})$ are stored. Main idea of the SA is to cope with local optima. For this, even if the return of x_{i*} is worse than the return of x_j there is a chance of keeping x_{i*}. The former optimal covariate combination is stored before it is overwritten. The likelihood of keeping worse responses decreases with decreasing temperature t thus with increasing number of outer loop iterations. The detailed calculation of the Monte-Carlo approach by Metropolis et al. (1953) can i.a. be found in Kirkpatrick et al. (1983). Storing the development of covariates and response can be help improving the performance of SA (Lin et al., 1995; Hansen, 2012). We implemented a modification of the threshold accepting strategy (Dueck and Scheuer, 1990) which enables reducing the total number of calculations. This is archived by simply calculating and storing the improvement as absolute difference of $f(x_i)$ and $f(x_j)$. If the response oscillates for user defined number of repetitions within a user defined threshold, the inner loop breaks.

Main functions of the outer loop are calling the inner loop as well as specifying t and vf for the next iteration of the outer loop. As t obligatory decreases in our derivation of SA, the number of iterations is implicitly user predefined by initial temperature t_0 and minimum temperature t_{min}. t is necessary for the stochastic part within the inner loop (Kirkpatrick et al., 1983). As each covariate can have its own random factor, the vector with random factors rf is of the same dimension as x_i. Dividing the counted number of invalid trials of a covariate by the total number of trials of the respective covariate gives the ratio of invalid trials distinctively for each covariate. Summing up the valid and invalid trials of all inner loop repetitions gives the mean ratio of invalid trials per outer loop repetition. According to (Corana et al., 1987), this ratio can be used to find a trade-off between accuracy on the one and size of the search-domain on the other side. They argument that if there were only valid covariate combinations, the search domain could be too small for multimodal function. They suggest ratios between 0.4 and 0.6. If any observed ratio is < 0.4 or > 0.6, the respective random factors are modified following the suggested nonlinear equation by Corana et al. (1987) such that they are dynamically adjusted for the next iteration. Pronzato et al. (1984), who developed the Adaptive Random Search method, propose a time decreasing search domain. We combined the two methods by linearly shrinking the favorable range of ratios from 0.4-0.6 to 0.04-0.06. If the default variation function is chosen, the search domain of the covariates reduces with increasing number of outer loop iterations.

May an additional graphical explanation be useful?

The package optimization

The function optim_sa

As the function is mainly written in C++, it requires **Rcpp**. `optim_sa` shall be able to solve very specific optimization problems, several parameters can be defined by the user. Quality of solution and speed of convergence will thus substantially depend on accurate parametrization. In the following, we will explain each parameter briefly giving hints for useful specification:

- `fun`: loss function to be optimized. The function must depend on a vector of covariates and return one numeric value. There are no assumptions on covariates and return. They are not necessarily continuous. Missing (`NA`) or undefined (`NaN`) returns are allowed as well. Any restriction on the parameter space, e. g. specific invalid covariate values within the boundaries, can be integrated in the loss function directly by simply returning NA. Restrictions on the state space can be defined analogously. We will be more specific on this in the practical examples.
- `start`: obligatory numeric vector with initial covariate combination. It must be ensured that at least the initial covariate combination leads to a defined numeric response. The loss function at the start variables combination must therefore return a defined numeric value. This might be relevant when the starting values are determined stochastically.
- `maximization`: logical value with default `FALSE` which indicates whether the loss function is to be maximized or minimized.
- `trace`: logical value with default `FALSE`. If `TRUE`, the last inner loop iteration of each outer loop iteration is stored as row in the trace matrix. This might help evaluating the solutions quality.

However, storing interim results increases calculation time up to 10 %. Disabling `trace` can thus improve efficiency when the convergence of an optimization problem is known to be stable.

- `lower`: numeric vector with lower boundaries of the covariates. The boundaries are obligatory since the dynamic `rf` adjustment (Corana et al., 1987; Pronzato et al., 1984) depends on the number of invalid covariate combinations.
- `upper`: numeric vector with upper boundaries of the covariates.
- `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`: function that determines the variation of covariates during the iterations. It is allowed to depend on `rf`, `temperature` and a 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 `vf`. Per default, the covariates are changed by a continuous, uniform 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 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`: logical. Default is `TRUE` `rf` change dynamically over time to ensure increasing precision with increasing number of iterations. Default is `TRUE`. `dyn_rf` ensures a relatively wide search domain at the beginning of the optimization process that shrinks over time (Corana et al., 1987; Pronzato et al., 1984). 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 reasonable fixed `rf`. Self-specified `vf` may not depend on `rf`. In this cases activating `dyn_rf` makes no sense.
- `t0`: initial temperature. Default is 1000. The temperature directly influences the likelihood of accepting worse responses thus the stochastic part of the optimization. `t0` should be adopted to the loss function complexity. Higher temperatures lead to higher ability of coping with local optima on the one but also to more time-consuming function calls on the other hand.
- `t_min`: numeric value that determines the temperature where outer loop stops. Default is 0.1. As there is approximately 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 less function calls.
- `nlimit`: integer value which determines the maximum number of inner loops iterations. Default is 100. If the break criterion `stopac` is not fulfilled, `nlimit` is the exact number of inner loops repetitions.
- `r`: numeric value that determines the reduction of the temperature at the end of the outer loop. Slower temperature reduction leads to 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 continuously decreasing acceptance likelihood of worse responses.
- `k`: numeric value. Constant for the Metropolis function (Kirkpatrick et al., 1983). Default is 1. `k` determines the global, non-dynamic likelihood of accepting worse responses. The higher `k` the higher the likelihood of accepting worse responses.
- `stopac`: number of repetitions when the threshold accepting criterion is fulfilled. Can break the inner but not the outer loop.
- `ac_acc`: respective accuracy for the threshold accepting criterion in relation to the response. Very useful for optimization problems without high requirements on accuracy. Lower accuracy can speed-up the solution substantially.

Practical examples

We briefly explain where the `optim_sa` function might be advantageous at 4 differing in their complexity.

Himmelblau Function with continuous parameter space

We chose Himmelblau's function (Himmelblau, 1972) as initial example since it is a very simple multimodal equation which is widely known in Operations Research. It has 4 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} (Equation 1). In order to display the advantages and basic behavior of **optimization**, we compared it with 2 other SA methods from the packages **stats** and **GenSA**. As the function is relatively simple we also included the NM direct search optimization method (Nelder and Mead, 1965) which is default of `optim` from the **stats** package.

$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{1}$$

We performed 10,000 optimizations with each function in order investigate quality and speed of the solutions. We parameterized the `optim_sa` function for relatively simple optimization problems. We tried to find parameters for all other investigated functions such the problem was solved properly. We performed optimizations having the following parameters:

```
# stats package: optim (NM)
stats::optim(fn = hi, par = c(10, 10), method = "Nelder-Mead")

# optimization package: optim_sa
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
                )
        )

# stats package optim (SA)
stats::optim(fn = hi, par = c(10, 10), method = "SANN",
        control = list(tmax = 500, reltol = 0.1, temp = 50, trace = TRUE
                )
        )

# GenSA package: GenSA
GenSA::GenSA(fn = hi, par = c(10, 10), lower = c(-40, -40), upper = c(40, 40),
        control = list(temperature = 50, nb.stop.improvement = 30, maxit = 500)
        )
```

Since we have a multimodal optimization problem, we must evaluate response as well as covariate combinations to rate the quality of the algorithms. Although starting parameter combination are fixed, the methods should be able to find all possible 4 solution. We defined the problem to be solved with sufficient accuracy when the response was ≤ 0.01 after minimization. We also looked at the frequency distribution of the covariate combination. After investigation of quality, we also compared the efficiencies by measuring calculation times using **microbenchmark** with the same options except for the trace function.

It could be seen that parameterization of NM was quite easy. It only needed a vector with starting values. The other functions needed much more settings. With view to accuracy each method performed well. All functions returned in mean of 10,000 calls responses ≤ 0.01. Each function was thus generally able to find a proper solution. With view to frequency distribution the functions performed indifferent (Table 1). `optim_sa` and `optim (SA)` returned each possible solutions. The combination {-3.8, -3.3} was consistently least frequent. `GenSa` and `optim (NM)`, however, solely returned {-3.8, -3.3}. Further investigation revealed the solutions of `GenSa` and `optim (NM)` to be sensitive of the starting values. If `GenSa` and `optim (NM)` were parameterized with randomly drawn starting values, all 4 results would have been possible. Advantage of `optim_sa` and `optim (SA)` against `GenSa` and `optim (NM)` is, in this example, thus its independence from starting values. If an optimization problem has multiple equivalent solutions, `optim (SA)` is able to archive each possible solution. Practically, in case users are in doubt whether a problem has multiple solutions, the function can be repeated without re-parameterization. If well-specified, `optim_sa` will return all possible solutions within the parameter space.

As all functions were practically able to minimize equation 1, comparison of calculation times appears to be another important point for quality assessment. As expected, the direct search method `optim NM` was by far faster then all random search methods (Figure 1). The within-group ranking of random search methods revealed `GenSA` to be most efficient. It was in average 3.5 times faster than `optim_sa`. The 2 functions able coping with equally valued optima (`optim (SA)` and `optim_sa`) were slower than these returning only 1 solution (Table 1, Figure 1). Due to the multiple additional options

Table 1: Relative frequencies of covariate combinations in % after optimization for the 4 examined methods. Number of repetitions: 10,000.

		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
	GenSA	0.00	0.00	0.00	100.00
	optim (NM)	0.00	0.00	0.00	100.00

(which are all called several times in the code) `optim_sa` was by far slowest. This example clearly reveals that higher flexibility and generality on the one hand leads to significantly higher calculation times on the other hand.

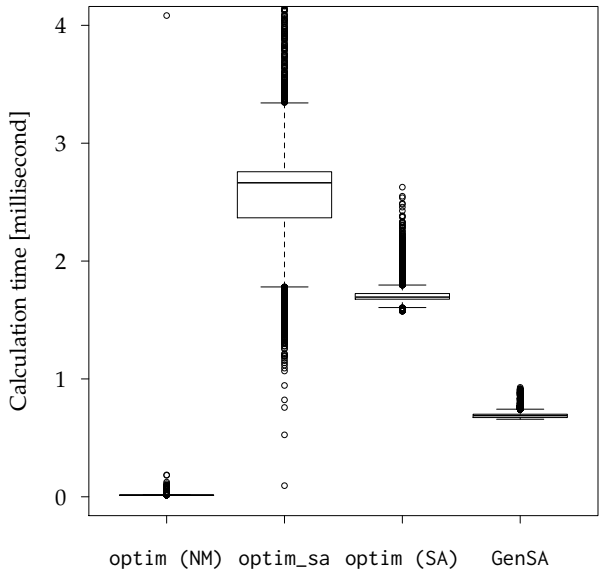

Figure 1: Calculation times of the 4 examined optimization algorithms. Note that the y-axis is truncated since every method showed outliers above 4 milliseconds. `optim_sa` and `optim (SA)` showed much outliers than `optim (NM)` and `GenSA`.

Himmelblau Function with discrete parameter space

A second example illustrating the option of flexibly defining the parameter space also bases on the function by Himmelblau (1972). In case a user is interested in integer covariate combinations only, the `vf` option can be used to restrict the parameter space accordingly. The following simple example shows a variation rule for integer programming problems. Mixed integer problems work similarly. `vf` must be defined as an R function. The simple `var_fun_int` from the example returns a vector of integer covariates varying in dimension of `vf` around the passed covariates `para_0`. The `fun_length` is passed explicitly, although it is implicitly given by the length of `para_0` because it might help simplifying the `vf`. `rf` and `temp` are optionally. If `vf` needs no dynamic part, they can separately be disabled by passing `NA`. If `vf` does not depend on `rf`, the option `dyn_rf` will not be necessary any more. Other restrictions may be defined analogously. `vf` can be used to flexibly restrict the parameter space.

```
# Define vf
var_fun_int <- function(para_0, fun_length, rf, temp = NA){
        ret_var_fun <- para_0 + sample.int(rf, fun_length, replace = TRUE) *
        ((rbinom(fun_length, 1, 0.5) * -2) + 1)
        return (ret_var_fun)
}

# Call optim_sa
```

```
int_programming <- 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, vf = var_func_int
                )
        )
```

Calling the minimization function always leaded to the only integer solution $f(3,2) = 0$. The generic plot function of `optimization` helps interpreting the convergence and thus the solution quality as well as the algorithm behavior. Since the used example is 2-dimensional, a `contour_plot` could be created. It became obvious that only integer covariate combinations were calculated during optimization (Figure 2, right). Figure 2 additionally helps explaining the stochastic part. It become clear that though the response of iteration 10 (parameter combination {2, 2}) was already quite near the optimum, iteration 11 ended with a relatively worse intermediate result. The likelihood of keeping worse solutions shrined over time till it became practically 0 after iteration 25 (Figure 2, left). Further hints for interpretation can be found in the package documentation.

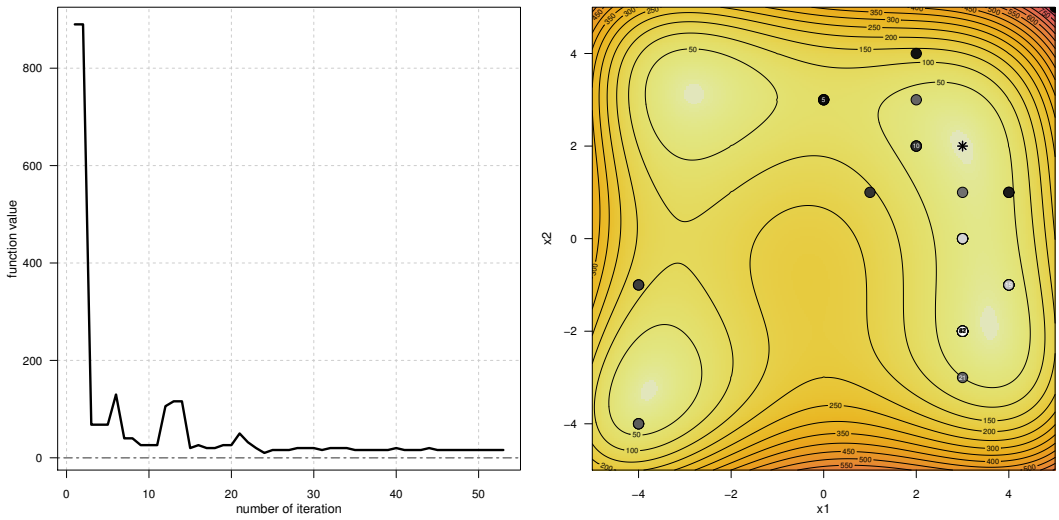

Figure 2: Examination plots created with the generic plot function. The left diagram shows the current optimal response over iteration of the outer loop. The left diagram displays the succession of the covariate values. The star points the covariate combination at optimum.

A complexer function?

SVAR

Alex

Forest harvesting schedule optimization

Popularity of optimizing forestry activities, a traditionally knowledge-based field with optimization methods playing only a minor role, is steadily rising. While *Linear Programs* are nowadays used in some states, e. g. Finland (Redsven et al., 2012), stochastic optimization programs are quite novel in forestry (Kangas et al., 2015). Optimization of forest harvesting planning thus represents an interesting and innovative practical example where `optim_sa` is recently used. **optimization** is integral part of the *ForestPlanner* decision support software for forest enterprises by Hansen and Nagel (2014). ForestPlanner is a user front end for *TreeGrowthOpenSourceSoftware* (TreeGrOSS) which is a complex Java written tree growth and yield simulation software used to forecast forest stands developments. 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 thus simulated individually. Optimization of forest activities is therefore not trivial since TreeGrOSS is a complex network of rules and functions which are predominantly nonlinear. The entire optimization process is composed of TreeGrOSS, **optimization** and 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 3 elements of the optimization system, must be composed of R, Java (using **rJava**) and SQL (using **RPostgreSQL**) code. 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 output reflects an optimizable response. 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 response is, accordingly, nonlinear and particularly non-continuous. Random search methods are hence favorable against all other types of optimization algorithms. Obligatory sustainability of forest stand treatment is also considered. Each function call returns, next to the actual response, a sustainability index. This index is used to restrict the optimization by simply defining `NA` responses whenever sustainability is violated.

It showed that forest treatment optimization was actually possible using **optimization**. We developed a loss function able translating the TreeGrOSS in- and output into interpretable variables for `optim_sa`. Sensitivity analysis using an exemplary forest enterprise comprised of 5 forest stands, 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 stands. The problem was solved within 12 hours using a cluster computer. Repeating the optimization 3 times leaded to equal responses but different harvesting volumes (thus covariate combinations). This again reveals the complexity of the problem and further reinforces the need for stochastic methods.

Discussion and outlook

It does not guarantee, of course, to find the global minimum, but if the function has many good near-optimal solutions, it should find one. In particular, this method is able to discriminate between gross behavior of the function and finer wrinkles. First, it reaches an area in the function domain where a global minimum should be present, following the gross behavior irrespectively of small local minima found on the way. It then develops finer details, finding a good, near-optimal local minimum, if not the global minimum itself.

Every method has advantages and disadvantages: GenSA for example is by far most efficient. It should be used for problems with 1 global optimum. optim_sa has its niche -> mult. equal opt. = felxible loss; NA returns -> enrichment of the optimization procedures in R This section may contain a figure such as Figure **??**. This file is only a basic article template. For full details of *The R Journal* style and information on how to prepare your article for submission, see the Instructions for Authors.

Bibliography

F. P. Agostini, D. D. O. Soares-Pinto, M. A. Moret, C. Osthoff, and P. G. Pascutti. Generalized simulated annealing applied to protein folding studies. *Journal of computational chemistry*, 27(11):1142–1155, 2006. [p1]

E. Z. Baskent and G. A. Jordan. Forest landscape management modeling using simulated annealing. *Forest Ecology and Management*, 165(1):29–45, 2002. [p1]

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003. [p1]

G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964. [p1]

A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm. *ACM Trans. Math. Softw.*, 13(3):262–280, 1987. [p1, 2, 3, 4]

G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. On a linear-programming, combinatorial approach to the traveling-salesman problem. *Operations Research*, 7(1):58–66, 1959. [p1]

W. S. DeSarbo, R. L. Oliver, and A. Rangaswamy. A simulated annealing methodology for clusterwise linear regression. *Psychometrika*, 54(4):707–736, 1989. [p1]

G. Dueck and T. Scheuer. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of computational physics*, 90(1):161–175, 1990. [p3]

C. Geiger and C. Kanzow. Das nelder-mead-verfahren. *Numerische Verfahren zur Lösung unregestrierter Optimierungsaufgaben*, 1999. [p1]

W. L. Goffe et al. Simann: A global optimization algorithm using simulated annealing. *Studies in Nonlinear Dynamics and Econometrics*, 1(3):169–176, 1996. [p1]

J. Hansen and J. Nagel. *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, 2014. ISBN 978-3-86395-149-8. [p7]

J. H. Hansen. *Modellbasierte Entscheidungsunterstützung für den Forstbetrieb: Optimierung kurzfristiger Nutzungsoptionen und mittelfristiger Strategien unter Verwendung metaheuristischer Verfahren und parallelen Rechnens*. Cuvillier, Göttingen, 1. aufl. edition, 2012. [p2, 3]

D. M. Himmelblau. *Applied nonlinear programming*. McGraw-Hill Companies, 1972. [p5, 6]

L. Ingber. Simulated annealing: Practice versus theory. *Mathematical and computer modelling*, 18(11): 29–57, 1993. [p1]

A. Kangas, M. Kurttila, T. Hujala, K. Eyvindson, and J. Kangas. *Decision Support for Forest Management*. Number 30 in Managing Forest Ecosystems. Springer International Publishing, Cham, 2nd ed edition, 2015. [p7]

S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598): 671–680, 1983. [p1, 2, 3, 4]

C. Lin, K. Haley, and C. Sparks. 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, 1995. [p3]

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953. [p3]

J. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7(4):308–313, 1965. [p5]

L. Pronzato, E. Walter, A. Venot, and J.-F. Lebruchec. A general-purpose global optimizer: Implimentation and applications. *Mathematics and Computers in Simulation*, 26(5):412–422, 1984. ISSN 03784754. doi: 10.1016/0378-4754(84)90105-8. [p1, 2, 3, 4]

V. Redsven, H. Hirvelä, K. Härkönen, O. Salminen, and M. Siitonen. *MELA2012 Reference Manual (2nd edition)*. The Finnish Forest Research Institute, 2012. [p7]

S. Theussl and H. W. Borchers. CRAN Task View: Optimization and Mathematical Programming. Oct. 2016. URL https://CRAN.R-project.org/view=Optimization. [p1]

I. Wegener. Simulated annealing beats metropolis in combinatorial optimization. In *International Colloquium on Automata, Languages, and Programming*, pages 589–601. Springer, 2005. [p1]

Y. Xiang, S. Gubian, B. Suomela, and J. Hoeng. Generalized simulated annealing for global optimization: the gensa package. *The R Journal Volume 5(1):13-29, June 2013*, 2013. [p1, 2]

Author One
Affiliation
Address
Country
author1@work

Author Two
Affiliation
Address
Country
author2@work

Author Three
Affiliation
Address
Country
author3@work