

Tuning Simulated Annealing Using the Sequential Parameter Optimization Toolbox SPOT

Thomas Bartz-Beielstein
Department of Computer Science,
Cologne University of Applied Sciences,
51643 Gummersbach, Germany

October 24, 2010

Abstract

The sequential parameter optimization (SPOT) package for R (R Development Core Team, 2008) is a toolbox for tuning and understanding simulation and optimization algorithms. Model-based investigations are common approaches in simulation and optimization. Sequential parameter optimization has been developed, because there is a strong need for sound statistical analysis of simulation and optimization algorithms. SPOT includes methods for tuning based on classical regression and analysis of variance techniques; tree-based models such as CART and random forest; Gaussian process models (Kriging), and combinations of different meta-modeling approaches. This article exemplifies how an existing optimization algorithms, namely simulated annealing, can be tuned using the SPOT framework

1 Introduction

This article illustrates how an existing algorithm can be tuned using the SPOT framework. The SPOT package can be installed from within R using the

`install.packages("SPOT")`

command. Alternatively, SPOT can be downloaded from the comprehensive R archive network at <http://CRAN.R-project.org/package=SPOT>. The latter procedure is recommended for the experienced R user only. SPOT is one possible implementation of the *sequential parameter optimization* (SPO) framework introduced in Bartz-Beielstein (2006). For a detailed documentation of the functions from the SPOT package, the reader is referred to the package help manuals.

The performance of modern search heuristics such as *evolution strategies* (ES), *differential evolution* (DE), or *simulated annealing* (SANN) relies crucially on

their parametrizations—or, statistically speaking, on their factor settings. The term *algorithm design* summarizes factors that influence the behavior (performance) of an algorithm, whereas *problem design* refers to factors from the optimization (simulation) problem. Population size in ES is one typical factor which belongs to the algorithm design, the search space dimension belongs to the problem design.

2 The SPOT Interface to SANN

SPOT provides an interface to the SANN algorithm, which can be downloaded from the workshop’s web site¹. Note, `spotAlgStartSann.R` is also included in the SPOT package. The corresponding R code is shown here:

```
spotAlgStartSann <- function(spotConfig){
  io.apdFileName=spotConfig$io.apdFileName;
  io.desFileName=spotConfig$io.desFileName;
  io.resFileName=spotConfig$io.resFileName;
  x0<-NULL
  maxit<-NULL
  parscale<-NULL
  f<-NULL
  n<-NULL
  if (spotConfig$spot.fileMode){ ##Check if spotConfig was passed to the algorithm, if yes the
    writeLines(paste("Loading design file data from:", io.desFileName), con=stderr());
    ## read doe/dace etc settings:
    des <- read.table( io.desFileName, sep=" ", header = TRUE);
  }else{
    des <- spotConfig$alg.currentDesign; ##The if/else should not be necessary anymore, since de
  }
  source(io.apdFileName,local=TRUE)
  pNames <- names(des);
  config<-nrow(des);
  for (k in 1:config){
    for (i in 1:des$REPEATS[k]){
      if (is.element("TEMP", pNames)){
        temp <- des$TEMP[k]
      }
      if (is.element("TMAX", pNames)){
        tmax <- round(des$TMAX[k])
      }
      conf <- k
      if (is.element("CONFIG", pNames)){
        conf <- des$CONFIG[k]
      }
    }
  }
```

¹<http://advml.gm.fh-koeln.de/~bartz/SpotSannIntro.d/spotAlgStartSann.R>

```

spotStep<-NA
if (is.element("STEP", pNames)){
spotStep <- des$STEP[k]
}
seed <- des$SEED[k]+i-1
set.seed(seed)
y <- optim(x0, spotNoisyBraninFunction, method="SANN",
control=list(maxit=maxit, temp=temp, tmax=tmax, parscale=parscale))
res <- NULL
res <- list(Y=y$value, TEMP=temp, TMAX=tmax, FUNCTION=f, DIM=n, SEED=seed, CONFIG=conf)
if (is.element("STEP", pNames)){
res=c(res,STEP=spotStep)
}
res <-data.frame(res)
if (spotConfig$spot.fileMode){ ##Log the result in the .res file, only if user didnt set fileMode
colNames = TRUE
if (file.exists(io.resFileName)){
colNames = FALSE
}
write.table(res, file = io.resFileName, row.names = FALSE,
col.names = colNames, sep = " ", append = !colNames, quote = FALSE);
colNames = FALSE
}
spotConfig$alg.currentResult=rbind(spotConfig$alg.currentResult,res);#always log the results
}
}
return(spotConfig)
}

```

This interface is used to call SANN from SPOT.

3 SANN

Wikipedia introduces simulated annealing as follows (http://en.wikipedia.org/wiki/Simulated_annealing):

Simulated annealing (SANN) is a generic probabilistic metaheuristic for the global optimization problem of applied mathematics, namely locating a good approximation to the global optimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For certain problems, simulated annealing may be more effective than exhaustive enumeration—provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution.

Table 1: SANN parameters. The first two parameters belong to the algorithm design, whereas the remaining parameters are from the problem design.

Name	Symbol	Factor name
Initial temperature	t	TEMP
Number of function evaluations at each temperature	$t_{\max}$	TMAX
Starting point	$\vec{x}_0 = (10, 10)$	
Problem dimension	$n = 2$	
Objective function	Branin	
Quality measure	Expected performance, e.g., $E(y)$	
Initial seed	s	
Budget	maxit = 250	

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one. By analogy with this physical process, each step of the SANN algorithm replaces the current solution by a random nearby solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called the temperature), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when T is large, but increasingly downhill as T goes to zero. The allowance for uphill moves saves the method from becoming stuck at local optima—which are the bane of greedier methods.

The method was independently described by Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi in 1983, and by Vlado Cerny in 1985. The method is an adaptation of the Metropolis-Hastings algorithm, a Monte Carlo method to generate sample states of a thermodynamic system, invented by N. Metropolis et al. in 1953.

SANN parameters and related problem parameters are shown in Tab. 1.

4 SPOT files

SPOT generates new design points (parameter settings for the SANN) by building a meta model. These designs are based on information provided by the user. These information has to be specified in the following files:

- CONF
- ROI
- APD

Detailed information about these desing generators is given in the tutorial *Performing experiments in the SPOT environment*, which can be downloaded from the workshop’s web site². The following section present a short introduction into the automated tuning process.

4.1 Experimental Setup

SPOT allows the user to specify the region of interest ROI. In addition, SPOT can be configured (CONF) and additional parameters can be passed to the algorithm (APD).

4.2 Files Used During the Tuning Process

Each configuration file belongs to one SPOT project, if the same basename is used for corresponding files. SPOT uses simple text files as interfaces from the algorithm to the statistical tools.

1. The user has to provide the following files:
 - (i) *Region of interest* (ROI) files specify the region over which the algorithm parameters are tuned. Categorical variables such as the recombination operator in ES, can be encoded as factors, e.g., “intermediate recombination” and “discrete recombination.”
 - (ii) *Algorithm design* (APD) files are used to specify parameters used by the algorithm, e.g., problem dimension, objective function, starting point, or initial seed.
 - (iii) *Configuration* files (CONF) specify SPOT specific parameters, such as the prediction model or the initial design size.
2. SPOT will generate the following files:
 - (i) *Design* files (DES) specify algorithm designs. They are generated automatically by SPOT and will be read by the optimization algorithms.
 - (ii) After the algorithm has been started with a parametrization from the algorithm design, the algorithm writes its results to the *result file* (RES). Result files provide the basis for many statistical evaluations/visualizations. They are read by SPOT to generate prediction models. Additional prediction models can easily be integrated into SPOT.

²<http://advml.gm.fh-koeln.de/bartz/spotworkshop.html>

4.3 SPOT Configuration

A *configuration* (CONF) file, which stores information about SPOT specific settings, has to be set up. For example, the number of SANN algorithm runs, i.e., the available budget, can be specified via `auto.loop.nevals`. SPOT implements a sequential approach, i.e., the available budget is not used in one step. Evaluations of the algorithm on a subset of this budget, the so-called initial design, is used to generate a coarse grained meta model F . This meta model is used to determine promising algorithm design points which will be evaluated next. Results from these additional SANN runs are used to refine the meta model F . The size of the initial design can be specified via `init.design.size`. To generate the meta model, we use random forest (Breiman, 2001). This can be specified via `seq.predictionModel.func = "spotPredictRandomForest"`.

Random forest was chosen, because it is a robust method which can handle categorical and numerical variables.

4.3.1 Setup of the CONF for the SANN

The corresponding CONF file for the SANN looks as follows. Note, all SPOT options are summarized in the `spotGetOptions`³ file.

```
alg.func = "spotAlgStartSann"
alg.seed = 1235
auto.loop.steps = Inf;
auto.loop.nevals = 100;
init.design.func = "spotCreateDesignLhs";
init.design.size = 10;
init.design.repeats = 2;
io.columnSep = " ";
seq.design.maxRepeats = 10;
seq.design.size = 100
seq.predictionModel.func = "spotPredictRandomForest"
spot.fileMode=TRUE
spot.seed = 125
report.func = "spotReportSens"
report.io.pdf = TRUE
```

The settings can be explained as follows.

alg.func: Specify the name of the algorithm to be tuned. Type: STRING

alg.seed: Seed passed to the algorithm. Type: INT

auto.loop.steps: SPOT Termination criterion. Number of meta models to be build by SPOT. Type: INT

init.design.func: Name of the function to create an initial design. TYPE: STRING

³<http://advml.gm.fh-koeln.de/bartz/spotGetOptions.html>

init.design.size: Number of initial design points to be created. Type: INT

init.design.repeats: Number of repeats for each design point from the initial design. Type: INT

seq.design.maxRepeats: Maximum number of repeats for design points. Type: INT

seq.design.size: Number of design points evaluated by the meta model. Type: INT

seq.predictionModel.func: Meta model. Type: STRING

spot.seed: Seed used by SPOT, e.g., for generating LHD. Type: INT

report.func: Name of the report function

report.io.pdf: Write report to pdf.

The corresponding CONF file can be downloaded from the workshop's web site⁴.

4.4 The Region of Interest

A *region of interest* (ROI) file specifies algorithm parameters and associated lower and upper bounds for the algorithm parameters.

4.4.1 Setup of the ROI for the SANN

- Values for `temp` are chosen from the interval [150].
- Values for `tmax` are from the interval [1; 50].

The corresponding ROI file looks as follows.

```
name low high type
TEMP 1 50 FLOAT
TMAX 1 50 FLOAT
```

SPOT provides an ROI template, which can be downloaded from the workshop's web site⁵.

4.5 The Algorithm and Problem Design File

Parameters related to the algorithm or the optimization problem are stored in the APD file. This file contains information about the problem and might be used by the algorithm. For example, the starting point `x0 = (0,0)` can be specified in the APD file.

⁴<http://advml.gm.fh-koeln.de/bartz/SpotSannIntro.d/rfSann02.conf>

⁵<http://advml.gm.fh-koeln.de/bartz/SpotSannIntro.d/rfSann02.roi>

4.5.1 Setup of the APD for the SANN

```
f = "Branin0.0"
x0 = c(10,10)
parscale = c(1,1)
n = 2
maxit = 250
```

SPOT provides an SANNROI file, which can be downloaded from the workshop's web site⁶.

5 Running SPOT—Automatic Tuning of SANN

Now that the interface has been setup, and the experimental setup has been specified, the first SPOT run can be performed. Consider the following situation: The user has created a working directory for running the experiments, say `IWNICA`, which is a subfolder of his `Documents` folder. This directory contains the following files.

- SPOT configuration (CONF), e.g., `rfSann02.conf`⁷
- Region of interest (ROI), e.g., `rfSann02.roi`⁸
- Algorithm and problem parameters (APD) `rfSann02.apd`⁹

R should be started in the `IWNICA` directory. You can use R's menu to *change the working directory*. Alternatively, you can set the working directory using the R command:

```
> setwd("c:/users/yourname/documents/IWNICA")
```

The following command starts SPOT's automatic tuning procedure.

```
> library(SPOT)
> spot("rfSann02.conf")
```

Sometimes it is required to start a clean R session, because data from previous runs are in the workspace. Execute

```
rm(list=ls());
```

to perform a cleanup before SPOT is loaded and run.

If `spot.fileMode=TRUE` was set (as in our example), a result file (RES), which contains important information from the tuning process, has been written to the working directory. It can be downloaded as `rfSann02.res`¹⁰ from the workshop's web page. This RES file is a good starting point for further analyses of the SANN algorithm.

⁶<http://advml.gm.fh-koeln.de/bartz/SpotSannIntro.d/rfSann02.apd>

⁷<http://advml.gm.fh-koeln.de/bartz/SpotSannIntro.d/rfSann02.conf>

⁸<http://advml.gm.fh-koeln.de/bartz/SpotSannIntro.d/rfSann02.roi>

⁹<http://advml.gm.fh-koeln.de/bartz/SpotSannIntro.d/rfSann02.apd>

¹⁰<http://advml.gm.fh-koeln.de/bartz/SpotSannIntro.d/rfSann02.res>

6 Summary

SPOT can be used for tuning a simulated annealing algorithm, SANN. This article explains the basic functions and setup and serves as a starting point for further experiments.

The file `allInclusive.zip`¹¹ contains all files used in this workshop, so you do not have to download the files separately.

All SPOT options are summarized in the `spotGetOptions`¹² file.

Note, SPOT has a graphical user interface which can be started via

`spotGui()`

You can join the Internet Discussion Group: http://www.researchgate.net/group/Sequential_Parameter_Optimization_Toolbox/, it is free.

The book Bartz-Beielstein et al. (2010) might be a good starting point for further studies.

References

- Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation—The New Experimentalism*. Natural Computing Series. Springer, Berlin, Heidelberg, New York.
- Bartz-Beielstein, T., Chiarandini, M., Paquete, L., and Preuss, M., editors (2010). *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, Berlin, Heidelberg, New York. Im Druck.
- Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.
- R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

¹¹<http://advml.gm.fh-koeln.de/bartz/SpotPerforming.d/allInclusive.zip>

¹²<http://advml.gm.fh-koeln.de/bartz/spotGetOptions.html>