Performing Experiments Using the Sequential Parameter Optimization Toolbox

SPOT

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

July 8, 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 metamodeling approaches. This article exemplifies how experiments can be performed using the SPOT framework.

1 Introduction

This article describes the experimental setup which is necessary to perfom experiments using the SPOT framework. The SPOT package can be downloaded from the comprehensive R archive network at `http://CRAN.R-project.org/package=SPOT`. 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

Procedure 1: (1+1)-ES()

$t := 0$;
initialize($\vec{x}$, σ);
$y_p := f(\vec{x_p})$;
repeat
  $\vec{x_o} := \vec{x_p} + \sigma(\mathcal{N}(0,1), \mathcal{N}(0,1), \ldots, \mathcal{N}(0,1))^T$;
  $y_o := f(\vec{x_o})$;
  if $y_o \leq y_p$ **then**
    $\vec{x_p} := \vec{x_o}$;
    $y_p = y_o$;
  end
  modify σ according to 1/5th rule;
  $t := t + 1$;
until `TerminationCriterion()` ;
return $(\vec{x_p}, y_p)$

which belongs to the algorithm design, the search space dimension belongs to the problem design.

The paper is structured as follows: Section 2 presents an example how the experimental setup for an optimization algorithm written in JAVA can be specified in the SPOT framework.

2 JAVA Algorithms

2.1 1+1 Evolution Strategy

2.1.1 1+1 Basics

We consider a simple *evolution strategy* (ES), the so-called (1+1)-ES, see Procedure 1. The 1/5th rule states that σ should be modified according to the rule

$$\sigma(t+1) := \begin{cases} \sigma(t)a, & \text{if } P_s > 1/5 \\ \sigma(t)/a, & \text{if } P_s < 1/5 \\ \sigma(t), & \text{if } P_s = 1/5 \end{cases} \tag{1}$$

where the factor a is usually between 1.1 and 1.5 and P_s denotes the success rate (Beyer, 2001). The factor a depends particularly on the measurement period g, which is used to estimate the success rate P_s. During the measurement period, g remains constant. For $g = n$, where n denotes the problem dimension, Schwefel (1995) calculated $1/a \approx 0.817$. Beyer (2001) states that the "choice of a is relatively uncritical" and that the 1/5th rule has a "remarkable validity domain." He also mentions limits of this rule.

Based on these theoretical results, we can derive certain scientific hypotheses. One might be formulated as follows: *Given a spherical fitness landscape, the*

Table 1: $(1+1)$-ES parameters. The first three parameters belong to the algorithm design, whereas the remaining parameters are from the problem design

Name	Symbol	Factor name in the algorithm design
Initial stepsize	$\sigma(0)$	SIGMANULL
Stepsize multiplier	a	VARA
History	$g = n$	VARG
Name	**Symbol**	**Name in the APD file[1]**
Starting point	$\vec{x_p}$	xp0
Problem dimension	n	n
Objective function	$f(\vec{x}) = \sum x_i^2$	f
Quality measure	Expected performance, e.g., $E(y)$	-
Initial seed	s	seed
Budget	$t_{\max}$	steps

(1+1)-ES performs optimally, if the step-sizes σ is modified according to the $1/5$*th rule as stated in Eq. 1.* This statement is related to the primary model.

In the experimental model, we relate primary questions or statements to questions about a particular type of experiment. At this level, we define an objective function, a starting point, a quality measure, and parameters used by the algorithm. These parameters are summarized in Table 1.

Note, the quality measure is defined in the CONF file.

2.1.2 The JAVA Implementation of the 1+1 ES

We are using a JAVA implementation of the $(1+1)$ ES described in Sect. 2.1.1. The corresponding `jar` file can be downloaded from the workshop's web site[2].

The JAVA (1+1)-ES algorithm uses the parameters from Tab. 2. The (1+1)-ES can be started using the `jar` file from the command line with the following arguments.

```
java -jar simpleOnePlusOneES.jar 1 100 1.0E-6
     de.fhkoeln.spot.objectivefunctions.Ball
     3 "c(1.0,1.0,1.0)" 1.0 1.2239 3 0 2
```

The following command-line parameters were used:

1. seed = 1;
2. the algorithm has a budget of one hundred function evaluations;
3. it terminates, if the function value is smaller than 1e-6;
4. the sphere function is used as the objective function;

[2]http://advm1.gm.fh-koeln.de/ bartz/simpleOnePlusOneES.jar

Table 2: JAVA ($1 + 1$)-ES: Parameters as reported by the algorithm

Name	Parameter
seed	random seed (e.g. 12345)
steps	maximum number of evolution steps (e.g. 10000)
target	objective function threshold for preliminary evolution end (e.g. 0.0001)
f	objective function class name (e.g. de.fhkoeln.spot.objectivefunctions.Ball)
n	problem dimension (e.g.12)
xp0	starting point (uniform = uniformly distributed random vector from $[0.0, 1.0]^n$, gaussian = normally distributed random vector from N(0,1), $c(xp0_0, ..., xp0_n)$ = the vector $[xp0_0, ..., xp0_n]$)
sigma0	initial step size (e.g. 1.0)
a	step size muliplier (e.g. 1.2239)
g	history length (e.g. 12 = n)
px	individual printing mode (0 = do not print individuals, 1 = only print best individual, 2 = only print improving step numbers and individuals, 3 = print every individual)
py	objective function value printing mode (0 = do not print objective function values, 1 = only print best objective function value, 2 = only print improving step numbers and objective function values, 3 = print every objective function value)

5. a three dimensional search space is used;
6. $(1, 1, 1)$ was chosen as the starting point;
7. the initial step size was set to one;
8. as a step size multiplier, the value 1.2239 was chosen;
9. the history length was set to three;
10. no information about individuals is printed;
11. and the best objective function value is reported at the end.

This algorithm run produces the following output:

```
1 0.3732544130302741
13 0.2268318386083562
20 0.19052464589633564
25 0.17090575193950355
31 0.14554127695687402
37 0.08943630492465122
38 0.07890216216826802
47 0.07318808722843884
53 0.0573032759515119
61 0.001451451919883614
68 0.0010101618142669604
79 1.89432721043702E-4
93 8.645160644753755E-5
```

2.2 CMA-ES

2.2.1 CMA-ES **Basics**

We are using Hansen's CMA-ES, see `http://www.lri.fr/~hansen/javadoc/index.html` for details.

3 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).

3.1 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.

3.2 SPOT Configuration

A *configuration* (CONF) file, which stores information about SPOT specific settings, has to be set up. For example, the number of (1+1)-ES 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 (1+1)-ES 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.

3.2.1 Setup of the CONF for the (1+1)-ES

The corresponding CONF file for the (1+1)-ES looks as follows. Note, all SPOT options are summarized in the spotGetOptions[3] file.

[3] http://advm1.gm.fh-koeln.de/ bartz/spotGetOptions.html

```
alg.path="."
alg.func = "spotInterfacingTemplate"
alg.seed = 1235

spot.seed = 125

auto.loop.steps = 50;

init.design.func = "spotCreateDesignLhd";
init.design.size = 10;
init.design.repeats  = 1;

seq.design.maxRepeats = 5;
seq.design.size = 250
seq.predictionModel.func = "spotPredictRandomForest"

io.verbosity=3
```

The settings can be explained as follows.

alg.path: Specify the path to the algorithm to be tuned. Type: STRING

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

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

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

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

io.verbosity: Level of verbosity of the programm. TYPE: INT.

SPOT provides an CONF template, which can be downloaded from the workshop's web site[4].

[4] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/java0.conf

3.2.2 Setup of the CONF for the CMA-ES

The corresponding CONF file for the CMA-ES looks as follows. Note, all SPOT options are summarized in the spotGetOptions[5] file.

```
alg.path="."
alg.func = "spotAlgStartCmaEsJava"


alg.seed = 1235

spot.seed = 125

auto.loop.steps = 50;

init.design.func = "spotCreateDesignLhd";
init.design.size = 10;
init.design.repeats  = 1;

seq.design.maxRepeats = 5;
seq.design.size = 250
seq.predictionModel.func = "spotPredictRandomForest"

io.verbosity=3
```

The settings can be explained as follows.

alg.path: Specify the path to the algorithm to be tuned. Type: STRING

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

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

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

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

[5] http://advm1.gm.fh-koeln.de/ bartz/spotGetOptions.html

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

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

io.verbosity: Level of verbosity of the programm. TYPE: INT

SPOT provides an CONF template, which can be downloaded from the workshop's web site[6].

3.3 The Region of Interest

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

3.3.1 Setup of the ROI for the (1+1)-ES

- Values for `sigmanull` are chosen from the interval $[.1; 5]$.
- Values for `vara` are from the interval $[1; 2]$ and
- Values for `varg` are from the interval $[2; 100]$.

The corresponding ROI file looks as follows.

```
name low high type
SIGMANULL 0.1 5 FLOAT
VARA 1 2 FLOAT
VARG 2 100 INT
```

SPOT provides an ROI template, which can be downloaded from the workshop's web site[7].

3.3.2 Setup of the ROI for the CMA-ES

The JAVACMA-ES algorithm uses the parameters from Tab. 3. Note, that there are constraints on the ls and us parameters:

$$ls > 0 \text{ and } us > ls. \tag{2}$$

Therefore, we introduce a new variable, namely usfprop, which is defined as follows:

$$us = usprop \times (1.0 + ls). \tag{3}$$

These parameters are related to the algorithm design and will be specified in SPOT's ROI file.

The corresponding ROI file for the CMA-ES looks as follows.

[6] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/cmaEs0.conf
[7] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/java0.roi

Table 3: JAVACMA-ES: Algorithm parameters as reported by the algorithm

Name	Parameter	Name in the ROI file
s	population size (e.g. 9)	S
restarts	number of restarts (e.g. 1)	RESTARTS
ipsf	increase population size factor (e.g. 2.0)	IPSF
isd	initial standard deviations (e.g. 0.3)	ISD
ls	lower standard deviations (step sizes)	LS
us	upper standard deviations (step sizes)	-
usprop	upper standard deviations (step sizes) factor	USPROP

```
name low high type
S 2 50 INT
RESTARTS 1 10 INT
IPSF 1 5 FLOAT
ISD 0.1 3 FLOAT
LS 0 1 FLOAT
USPROP 1 2 FLOAT
```

SPOT provides an ROI template, which can be downloaded from the workshop's web site[8].

3.4 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 `xp0 = "[1.0,0.0,1.0,0.0,1.0,0.0,1.0,0.0,1.0,0.0]"` can be specified in the APD file.

3.4.1 Setup of the APD for the (1+1)-ES

```
px = 0
py = 1
steps = 100
target = 1e-10
f = "de.fhkoeln.spot.objectivefunctions.Ball"
n = 10
xp0 = "[1.0,0.0,1.0,0.0,1.0,0.0,1.0,0.0,1.0,0.0]"
seed = 123

sigma0 = 1
a = 1.2
g = 10
```

[8] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/cmaEs0.roi

Table 4: JAVACMA-ES: Parameters as reported by the algorithm

Name	Parameter	Parameter name used in the CMA-ESAP
seed	random seed (e.g. 12345)	seed
d	search space dimension (e.g. 22)	d
evals	maximum number of fitness function evaluations (e.g. 100000)	evals
f	objective function index (e.g. 50, 10 is Sphere)	f
frot	objective function rotation (e.g. 0)	frot
arot	objective function axis rotation (e.g. 0.0)	arot
tx	typical X (e.g. 0.5)	tx

SPOT provides an (1+1)-ESAPD template, which can be downloaded from the workshop's web site[9].

3.4.2 Setup of the APD for the CMA-ES

The APD file for the CMA-ES contains information about the problem design. The JAVACMA-ES algorithm uses the parameters from Tab. 4.

```
# cmaEs0.apd: cmaEs problem design parameters
seed = 123
d = 10
evals = 100
f = 10
frot = 0
arot = 0.0
tx = 1.0
```

SPOT provides an CMA-ESAPD template, which can be downloaded from the workshop's web site[10].

4 Running SPOT

Now that the interface has been setup, and the experimental setup has been specified, the first SPOT run can be performed.

4.1 Automatic Tuning of the (1+1)-ES

Consider the following situation: The user has created a working directory for running the experiments, say `MyJavaExperiments`. This directory contains the following files.

[9] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/java0.apd
[10] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/cmaEs0.apd

- SPOT configuration (CONF), e.g., java0.conf[11]
- Region of interest (ROI), e.g., java0.roi[12]
- Algorithm and problem parameters (APD) java0.apd[13]
- The interface to the algorithm. spotInterfacingTemplate.R[14]
- The algorithm, i.e., the `jar` file. simpleOnePlusOneES.jar[15]

R is started in the working directory. The following command starts SPOT's automatic tuning procedure.

```
> library(SPOT)
> spot("java0.conf")
```

Sometimes is it 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.

The output from Fig. 1 is shown by default during the tuning process.

The tuning process terminates with the following output:

```
Best solution found with 507 evaluations:
Y          SIGMANULL VARA     VARG COUNT CONFIG
0.1029542  0.445661 1.050709   72     5    110
```

A short summary is also shown:

```
       y              SIGMANULL           VARA            VARG
 Min.   :0.01184   Min.   :0.1035   Min.   :1.023   Min.   : 3.0
 1st Qu.:0.11606   1st Qu.:0.2885   1st Qu.:1.064   1st Qu.:18.0
 Median :0.19034   Median :0.4112   Median :1.096   Median :50.0
 Mean   :0.25662   Mean   :0.6167   Mean   :1.118   Mean   :43.7
 3rd Qu.:0.31951   3rd Qu.:0.7337   3rd Qu.:1.145   3rd Qu.:64.0
 Max.   :1.68859   Max.   :4.5725   Max.   :1.984   Max.   :98.0
```

A result file (RES), which contains important information from the tuning process, has been written to the working directory. It can be downloaded as java0.res[16] from the workshop's web page.

[11] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/java0.conf
[12] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/java0.roi
[13] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/java0.apd
[14] http://advm1.gm.fh-koeln.de/ bartz/SpotInterfacing.d/spotInterfacingTemplate.R
[15] http://advm1.gm.fh-koeln.de/ bartz/simpleOnePlusOneES.jar
[16] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/java0.res

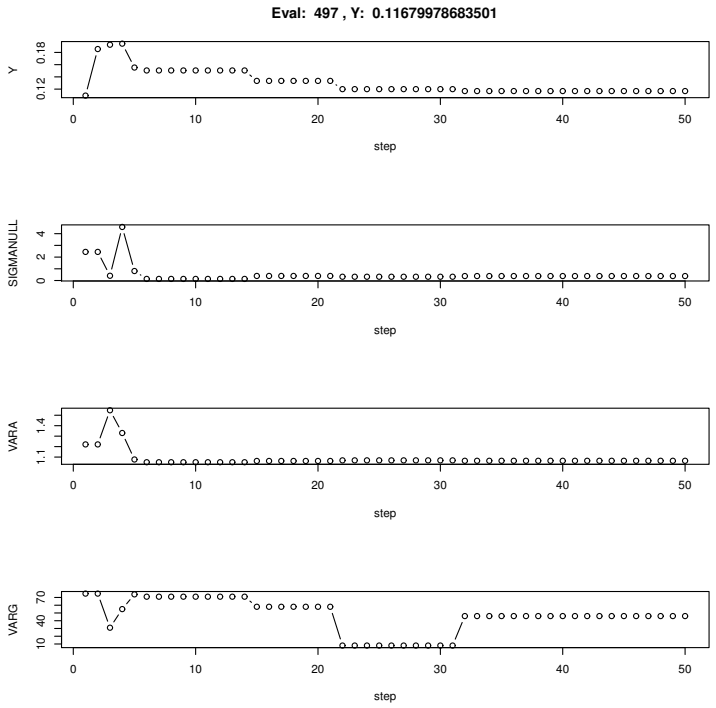

Figure 1: Default output during the optimization run

4.2 Automatic Tuning of the CMA-ES

Consider the following situation: The user has created a working directory for running the experiments, say `MyJavaExperiments`. This directory contains the following files.

- SPOT configuration (CONF), e.g., cmaEs0.conf[17]
- Region of interest (ROI), e.g., cmaEs0.roi[18]
- Algorithm and problem parameters (APD) cmaEs0.apd[19]
- The interface to the algorithm. spotAlgStartCmaEsJava.R[20]
- The algorithm, i.e., the `jar` file. cmaEs.jar[21]

R is started in the working directory. The following command starts SPOT's automatic tuning procedure.

[17] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/cmaEs0.conf
[18] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/cmaEs0.roi
[19] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/cmaEs0.apd
[20] http://advm1.gm.fh-koeln.de/ bartz/SpotInterfacing.d/spotAlgStartCmaEsJava.R
[21] http://advm1.gm.fh-koeln.de/ bartz/cmaEs.jar

```
> library(SPOT)
> spot("cmaEs0.conf")
```

Sometimes is it 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.

The output from Fig. 2 is shown by default during the tuning process.

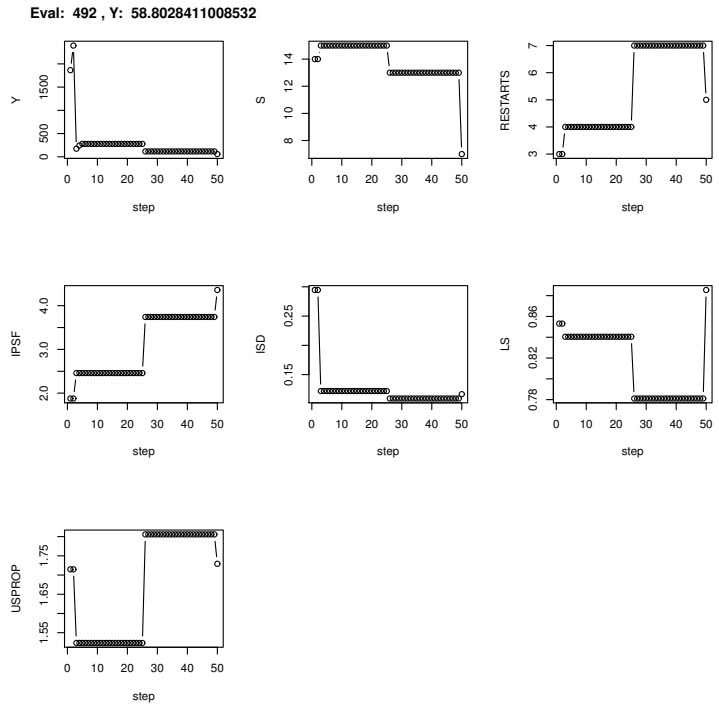

Figure 2: Default output during the optimization run

The tuning process terminates with the following output:

```
Best solution found with 502 evaluations:
          Y  S RESTARTS     IPSF       ISD        LS   USPROP COUNT CONFIG
82 117.2177 10        3 2.790516 0.1132285 0.7860233 1.846407     5     82
```

A short summary is also shown:

```
      y                  S            RESTARTS          IPSF
 Min.   :    27.96   Min.   : 2.00   Min.   : 1.000   Min.   :1.103
 1st Qu.:   819.95   1st Qu.: 9.00   1st Qu.: 4.000   1st Qu.:1.865
```

```
Median :  2445.59   Median :19.00   Median : 6.000   Median :2.648
Mean   : 10084.99   Mean   :22.90   Mean   : 5.703   Mean   :2.625
3rd Qu.:  9162.28   3rd Qu.:36.00   3rd Qu.: 7.000   3rd Qu.:3.221
Max.   :430195.03   Max.   :49.00   Max.   :10.000   Max.   :4.859
     ISD               LS              USPROP
Min.   :0.1006   Min.   :0.05975   Min.   :1.040
1st Qu.:0.1717   1st Qu.:0.67164   1st Qu.:1.610
Median :0.2611   Median :0.78602   Median :1.682
Mean   :0.3317   Mean   :0.75616   Mean   :1.684
3rd Qu.:0.4081   3rd Qu.:0.86056   3rd Qu.:1.774
Max.   :2.7564   Max.   :0.97134   Max.   :1.980
```

A result file (RES), which contains important information from the tuning process, has been written to the working directory. It can be downloaded as cmaEs0.res[22] from the workshop's web page.

5 A Closer Look at SPOT's Tasks

In Sect. 4, SPOT was run as an automatic tuner. Steps from the automatic mode can be used in an interactive manner. SPOT can be started with the command

```
spot (<configurationfile>,  <task>)
```

where `configurationfile` is the name of the SPOT configuration file (String) and `task` is a STRING and can be one of the tasks `init`, `seq`, `run`, `rep` or `auto`. SPOT can also be run in a `meta` mode to perform tuning over a set of problem instances.

SPOT provides tools to perform the following tasks:

1. *Initialize.* An initial design is generated. This is usually the first step during experimentation. The employed parameter region (ROI) and the constant algorithm parameters (APD) have to be provided by the user. SPOT's parameters are specified in the CONF file. Although it is recommended to use the same basename for CONF, ROI, and APD files in order to define a project, this is not mandatory. SPOT allows a flexible combination of different filenames, e.g., one APD file can be used for different projects.

2. *Run.* This is usually the second step. The optimization algorithm is started with configurations of the generated design. Additionally information about the algorithms problem design are used in this step. The algorithm writes its results to the result file.

3. *Sequential step.* A new design, based on information from the result file, is generated. A prediction model is used in this step. Several generic prediction models are available in SPOT by default. To perform an efficient

[22] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/cmaEs0.res

analysis, especially in situations when only few algorithms runs are possible, user-specified prediction models can easily be integrated into SPOT.

4. *Report*. An analysis, based on information from the result file, is generated. Since all data flow is stored in files, new report facilities can be added very easily. SPOT contains some scripts to perform a basic regression analysis and plots such as histograms, scatter plots, plots of the residuals, etc.

5. *Automatic* mode. In the automatic mode, the steps *run* and *sequential* are performed after an initialization for a predetermined number of times.

6. *Meta* mode. In the `meta` mode, the tuning process is repeated for several configurations. For example, tuning can be performed for different starting points $\vec{x}_0$, several dimensions, or randomly chosen problem instances.

5.1 The Init Task

The command

```
> spot("java0.conf", "init")
```

invokes SPOT's init step. Based on information from the CONF and ROI files, a design file (DES) is generated. This file is shown here:

```
SIGMANULL VARA VARG CONFIG REPEATS STEP SEED
4.50189721501898 1.98423853812274 81 1 1 0 1235
4.57246683350997 1.33011866996530 55 2 1 0 1235
0.845961183127947 1.40626377388835 63 3 1 0 1235
3.36020042322343 1.68839243145194 15 4 1 0 1235
1.87136912634131 1.17582192602567 8 5 1 0 1235
2.44399118965725 1.22097277157009 75 6 1 0 1235
0.406022209986113 1.5474003639305 31 7 1 0 1235
3.66735229385784 1.89343050755560 43 8 1 0 1235
1.20762449005153 1.72775872701313 98 9 1 0 1235
2.7426001026202 1.0844767608447 40 10 1 0 1235
```

5.1.1 Choosing an Alternative Design for SPOT's Init Task

Many designs generators are available in R. This is one of the main reasons why SPOT is implemented in R. The user can use state of the art design generators for tuning his algorithm. Or, he can write his own design plugin and use it as a plugin for SPOT.

The default SPOT installation contains several design plugins (and further design plugins will be added in forthcoming versions). Table 5 summarizes design plugins from the current SPOT version (0.1.888). The command `spotVersion()` displays the actual version of your local SPOT package.

A Latin hypercube design was chosen as the default initial design, because it is easy to implement and understand.

Table 5: SPOT initial design plugins

Type	Name of the SPOT plugin	R pa
Fractional factorial design (Resolution III)	`spotCreateDesignDoe3`	FrF2
Factorial design	`spotCreateDesignBasicDoe`	AlgD
Latin hypercube	`spotCreateDesignLhd`	SPOT
Latin hypercube	`spotCreateDesignLhs`	lhs

These plugins should be considered as templates. They were implemented in order to demonstrate how the interfaces should look like. We strongly recommend an adaptation of these plugins to your specific needs.

Automatic Tuning of the (1+1)-ES with Gaussian process models To select a Gaussian process model for the tuning process, simply modify the line

```
seq.predictionModel.func = "spotPredictRandomForest"
```

to

```
seq.predictionModel.func = "spotPredictMlegp"
```

Note, SPOT options are summarized in the spotGetOptions[23] file. Meta models are use the prefix `spotPredict*`.

5.2 The Run Task

The command

```
> spot("java0.conf", "run")
```

invokes SPOT's run step. The algorithm, e.g., `simpleOnePlusOneES.jar` is executed. Each line of the DES file contains one parameter configuration for the (1+1)-ES. The corresponding results are written to the result file (RES). The result file from executing the DES file from above looks as follows.

```
Y SIGMANULL VARA VARG Function MAXITER DIM TARGET SEED CONFIG STEP
1.6878752317589363 4.50189721501898 1.98423853812274 81 Ball 100 10 1e-10 1236 1 0
0.19435389671821338 4.57246683350997 1.3301186699653 55 Ball 100 10 1e-10 1236 2 0
0.19614961385471452 0.845961183127947 1.40626377388835 63 Ball 100 10 1e-10 1236 3 0
1.1336280907965832 3.36020042322343 1.68839243145194 15 Ball 100 10 1e-10 1236 4 0
0.20147814905430297 1.87136912634131 1.17582192602567 8 Ball 100 10 1e-10 1236 5 0
0.10938976591467603 2.44399118965725 1.22097277157009 75 Ball 100 10 1e-10 1236 6 0
0.19260197240195384 0.406022209986113 1.5474003639305 31 Ball 100 10 1e-10 1236 7 0
1.628667713635999 3.66735229385784 1.8934305075556 43 Ball 100 10 1e-10 1236 8 0
0.5800269409368702 1.20762449005153 1.72775872701313 98 Ball 100 10 1e-10 1236 9 0
0.4935194326825984 2.7426001026202 1.0844767608447 40 Ball 100 10 1e-10 1236 10 0
```

[23] http://advm1.gm.fh-koeln.de/ bartz/spotGetOptions.html

5.3 The Sequential Task

Now that results have been written to the result file, the meta model can be build.

```
spot("java0.conf","seq")
```

The sequential call generates a new *design file* (DES), which is shown here.

```
SIGMANULL VARA VARG CONFIG REPEATS repeatsLastConfig STEP SEED
2.44399118965725 1.22097277157009 75 6 1 1 1 1236
0.893540025975183 1.16118876079656 65.1960384295396 11 2 2 1 1235
0.946482443374582 1.38916710264608 68.2101951118447 12 2 2 1 1235
```

In order to improve confidence, the best solution found so far is evaluated again. To enable fair comparisons, new configurations are evaluated as many times as the best configuration found so far. Note, other update schemes are possible.

If SPOT's budget is not exhausted, the new configurations are evaluated, i.e., `run` is called again, which updates the result file. In the following step, `seq` is called again etc.

To support exploratory data analysis, SPOT also generates a best file. The content of the BST file after the run is finished is shown here.

```
Y SIGMANULL VARA VARG COUNT CONFIG
0.109389765914676 2.44399118965725 1.22097277157009 75 1 6
0.185522132969328 2.44399118965725 1.22097277157009 75 2 6
0.192601972401954 0.406022209986113 1.5474003639305 31 1 7
0.194353896718213 4.57246683350997 1.3301186699653 55 1 2
0.155270353688288 0.810298690075241 1.07694991343189 74 5 18
0.150488510690188 0.137167671956401 1.05063784049824 71 5 20
0.150488510690188 0.137167671956401 1.05063784049824 71 5 20
0.150488510690188 0.137167671956401 1.05063784049824 71 5 20
0.150488510690188 0.137167671956401 1.05063784049824 71 5 20
0.150488510690188 0.137167671956401 1.05063784049824 71 5 20
0.150488510690188 0.137167671956401 1.05063784049824 71 5 20
0.150488510690188 0.137167671956401 1.05063784049824 71 5 20
0.150488510690188 0.137167671956401 1.05063784049824 71 5 20
0.150488510690188 0.137167671956401 1.05063784049824 71 5 20
0.133442747019774 0.390411197773647 1.06283169919532 58 5 38
0.133442747019774 0.390411197773647 1.06283169919532 58 5 38
0.133442747019774 0.390411197773647 1.06283169919532 58 5 38
0.133442747019774 0.390411197773647 1.06283169919532 58 5 38
0.133442747019774 0.390411197773647 1.06283169919532 58 5 38
0.133442747019774 0.390411197773647 1.06283169919532 58 5 38
0.133442747019774 0.390411197773647 1.06283169919532 58 5 38
0.119941727061402 0.322729802686907 1.06848478622735 8 5 51
0.119941727061402 0.322729802686907 1.06848478622735 8 5 51
0.119941727061402 0.322729802686907 1.06848478622735 8 5 51
```

```
0.119941727061402 0.322729802686907 1.06848478622735 8 5 51
0.119941727061402 0.322729802686907 1.06848478622735 8 5 51
0.119941727061402 0.322729802686907 1.06848478622735 8 5 51
0.119941727061402 0.322729802686907 1.06848478622735 8 5 51
0.119941727061402 0.322729802686907 1.06848478622735 8 5 51
0.119941727061402 0.322729802686907 1.06848478622735 8 5 51
0.119941727061402 0.322729802686907 1.06848478622735 8 5 51
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.116799786835010 0.380922899061162 1.06452080560755 46 5 71
0.102954222434145 0.445660976006463 1.05070910971053 72 5 110
```

The best file is updated after each SPOT iteration and can be used for an on-line visualization tool, e.g., to illustrate search progress or stagnation. The variable COUNT reports the number of REPEATS used for this specific configuration.

5.3.1 Choosing an Alternative Meta Model for SPOT's Sequential Task

SPOT processes data sequentially, i.e., starting from a small initial design, further design points (parameter settings for the algorithm) are generated using a meta model. Many meta models are available in R. This is one of the main reasons why SPOT is implemented in R. The user can use state of the art meta models for tuning his algorithm. Or, he can write his own meta model and use it as a plugin for SPOT.

The default SPOT installation contains several meta models (and further meta models will be added in forthcoming versions). Table 6 summarizes meta models from the current SPOT version (0.1.888). The command `spotVersion()` displays the actual version of your local SPOT package.

`randomForest` was chosen as the default meta model, because it is quite robust and can handle categorical and numerical values.

Table 6: SPOT meta models

Type	Name of the SPOT plugin	R package
Linear model	`spotPredictLm`	base
Response surface methodology	`spotPredictLmOptim`	`rsm`
Regression trees	`spotPredictTree`	`rpart`
Random forest	`spotPredictRandomForest`	`random forest`
Gaussian processes (Kriging)	`spotPredictMlegg`	`mlegp`
Tree based Gaussian processes	`spotPredictTgp`	`tgp`

These plugins should be considered as templates. They were implemented in order to demonstrate how the interfaces should look like. We strongly recommend an adaptation of these plugins to your specific needs.

Automatic Tuning of the (1+1)-ES with Gaussian process models To select a Gaussian process model for the tuning process, simply modify the line

```
seq.predictionModel.func = "spotPredictRandomForest"
```

to

```
seq.predictionModel.func = "spotPredictMlegp"
```

Note, SPOT options are summarized in the spotGetOptions[24] file. Meta models are use the prefix `spotPredict*`.

Automatic Tuning of the (1+1)-ES with User-defined meta models The user can define additional meta models (and she will hopefully send her model to the SPOT developers).

To set the path to the user defined meta model, the variable `seq.design.path` can be modified.

5.4 The Report Task

If SPOT's termination criterion is fulfilled, a report is generated. By default, SPOT provides as simple report function which reads data from the RES file and produces the following output:

```
 Best solution found with 507 evaluations:
Y          SIGMANULL VARA VARG COUNT CONFIG
0.1029542  0.445661 1.050709   72     5    110
```

The best solution found by SPOT has an average function value $Y = 0.1029542$. The automatic tuning process found an initial stepwidth SIGMANULL = 0.445661, the value VARA = 1.050709 for the stepsize multiplier, and the history length VARG = 72 . The average function value $Y = 0.1029542$ is based on five

[24] http://advm1.gm.fh-koeln.de/ bartz/spotGetOptions.html

(COUNT = 5) algorithm runs. This configuration is the 110th configuration used in the SPOT run.

The output from Fig. 3 is shown by default after the tuning process is finished.

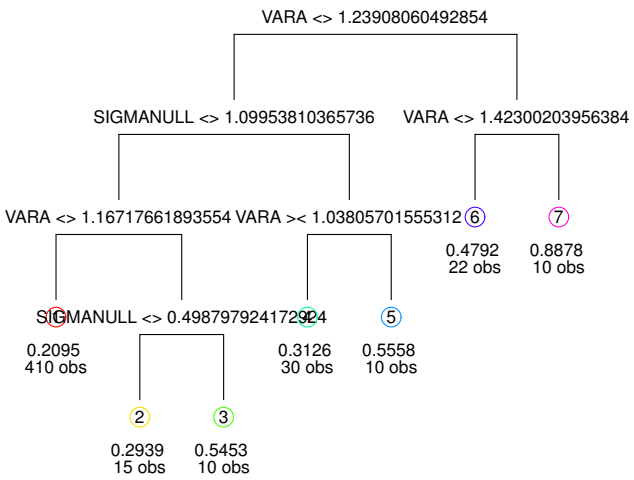

Figure 3: Default output after the tuning process is finished

The user can choose alternative report functions or even add new report functions to SPOT.

5.4.1 Choosing an Alternative Report Function

The user can change the name of the default report function in the CONF file, e.g., `java0.conf`. SPOT options are summarized in the spotGetOptions[25] file. The variable `report.func` has the default value `spotReportDefault`.

Table 7 summarizes report plugins from the current SPOT version (0.1.888). The command `spotVersion()` displays the actual version of your local SPOT package.

`spotReportDefault` determines a regression tree, because it is quite robust and can handle categorical and numerical values.

[25] http://advm1.gm.fh-koeln.de/ bartz/spotGetOptions.html

Table 7: SPOT report plugins

Type	Name of the SPOT plugin	R package
Regression trees	`spotReportDefault`	`rpart`
Random forest	`spotPredictSens`	`random forest`

These plugins should be considered as templates. They were implemented in order to demonstrate how the interfaces should look like. We strongly recommend an adaptation of these plugins to your specific needs.

Report Generation with Random Forest The user can change the value of the `report.func` from `spotReportDefault` to `spotReportSens` in the `java0.conf` file. The corresponding line in the `java0.conf` looks as follows:

```
report.func = "spotReportSens"
```

Executing the command

```
> spot("java0.conf", rep)
```

produces the result shown in Fig. 4

Note, the `spotReportSens` is included in the SPOT package.

Integrating New Report Plugins The user can define her own report plugins as follows. The default report plugin can be used as a starting point. Entering the name of the default report plugin at R's command line will display the R source code. So,

```
> spotReportDefault
```

produces the following output.

```
spotReportDefault <- function(spotConfig) {
spotWriteLines(spotConfig,2,"  Entering spotReportDefault");
rawB <- spotGetRawDataMatrixB(spotConfig);
print(summary(rawB));
mergedData <- spotPrepareData(spotConfig)
mergedB <- spotGetMergedDataMatrixB(mergedData, spotConfig);
C1 = spotWriteBest(mergedData, spotConfig);
C1 = C1[C1$COUNT==max(C1$COUNT),];    # choose only among the solutions with high repeat
cat(sprintf("\n Best solution found with %d evaluations:\n",nrow(rawB)));
print(C1[1,]);
fit.tree <- rpart(y ~ ., data= rawB)
if (!is.null(fit.tree$splits)){
if(spotConfig$report.io.pdf==TRUE){ #if pdf should be created
pdf(spotConfig$io.pdfFileName) #start pdf creation
spotPlotBst(spotConfig)
par(mfrow=c(1,1))
```

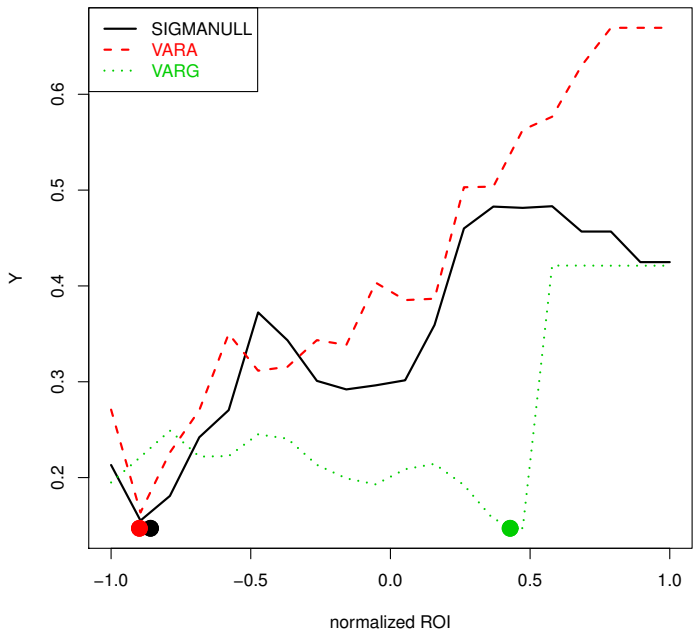

Figure 4: User specified output from the `report` task after the tuning process is finished

```
draw.tree(fit.tree, digits=4)
dev.off() #close pdf device
}
if(spotConfig$report.io.screen==TRUE) #if graphic should be on screen
{
x11()
par(mfrow=c(1,1), xpd=NA)
draw.tree(fit.tree, digits=4)
}
}
}
```

This output can be copied and pasted into a new R file. First, we choose a new name for this plugin, say `myReport`. This file has to be saved as `myReport.R`. We choose the actual working directory, i.e., where the `java0.*` project files reside and where R is started, to save this file. SPOT's function `spotGetRawDataMatrixB` is used to read the data from the RES file.

```
myReport <- function(spotConfig) {
```

```
rawB <- spotGetRawDataMatrixB(spotConfig);
...
}
```

Assuming the user wants to fit a linear regression model to the data, she has to modify the report plugin as follows. In addition to the standard plot of the linear model, she adds a plot of the effects. This plot requires R functions from the `effects` package, which is loaded via `library effects`.

```
myReport <- function(spotConfig) {
rawB <- spotGetRawDataMatrixB(spotConfig);
my.lm <- lm(y ~ ., data= rawB)
print(summary(my.lm))
par(mfrow=c(2,2))
plot(my.lm)
library(effects)
x11()
plot(allEffects(my.lm), ask=FALSE)
}
```

Since the new report plugin is located in the actual working directory, the user has to add the line `report.path = "."` to the CONF file `java0.conf`. The corresponding lines in the `java0.conf` has to be modified as follows:

```
report.func = "myReport"
report.path = "."
```

That's all. For your convenience, the plugin `myReport.R` can be downloaded from the workshop's web site[26].

Executing the command

```
> spot("java0.conf", rep)
```

produces the following output.

```
Call:
lm(formula = y ~ ., data = rawB)

Residuals:
     Min       1Q   Median       3Q      Max
-0.55087 -0.11698 -0.04086  0.07470  1.11687

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.7773069  0.1107214  -7.020 7.20e-12 ***
SIGMANULL    0.0668264  0.0154612   4.322 1.86e-05 ***
VARA         0.8809139  0.1009097   8.730  < 2e-16 ***
```

[26] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/myReport.R

```
VARG          0.0001705  0.0003798    0.449     0.654
---
Residual standard error: 0.1982 on 503 degrees of freedom
Multiple R-squared: 0.2538,     Adjusted R-squared: 0.2493
F-statistic: 57.02 on 3 and 503 DF,  p-value: < 2.2e-16
```

Fig. 5 shows the output from the generic diagnostic plot on the linear model.

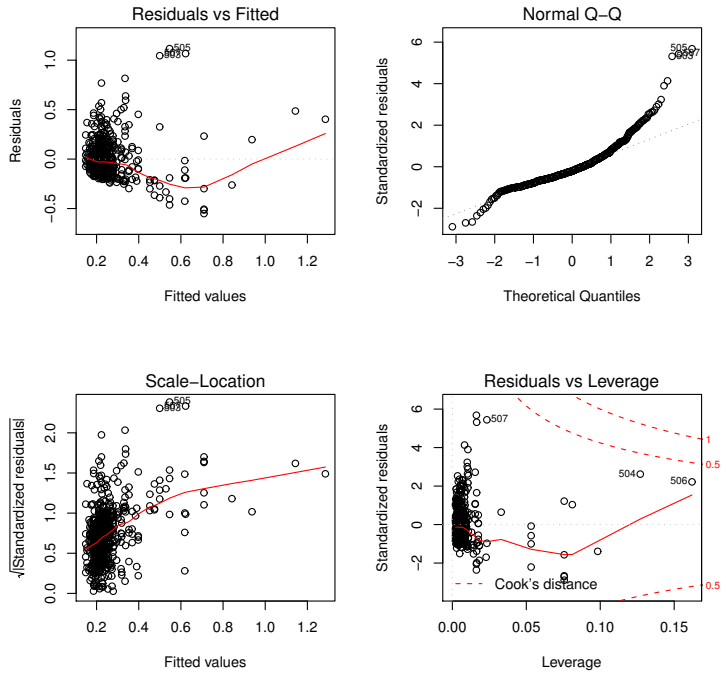

Figure 5: Diagnostic plot from the `report` task

Fig. 6 shows effect plots related to this linear model. Note, since the new report plugin is an R function, it has to be removed from R's workspace, if modifications to the report plugin should have an effect. This can be accomplished by R's `rm()` command. So, the following commands have to be executed if the report plugin `myReport.R` is modified several times.

```
> rm(myReport)
> spot("java0.conf", rep)
```

Integrating New Report Plugins (Part 2) The report plugin `myReport2.R`, which combines results from the previous paragraph can be downloaded from the workshop's web site[27].

[27] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/myReport2.R

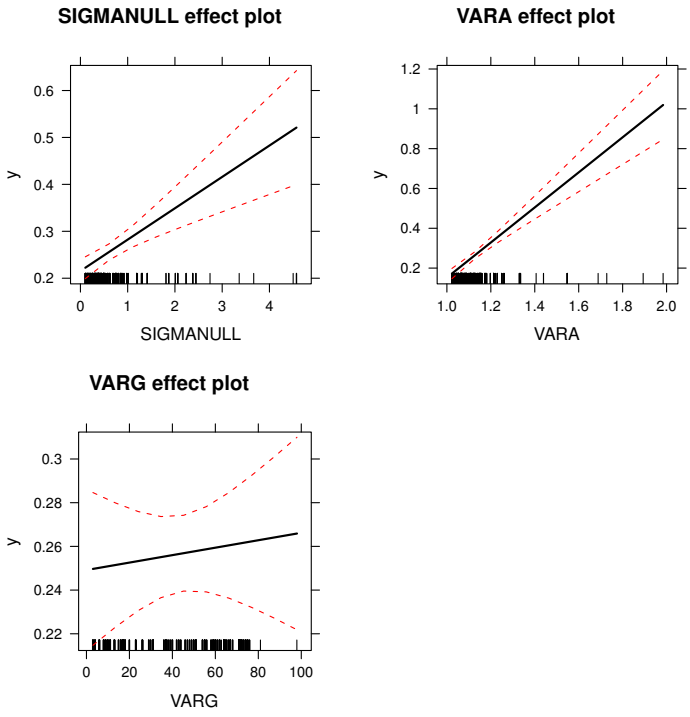

Figure 6: Effect plots from the `report` task `myReport`

```
myReport2 <- function(spotConfig) {
spotWriteLines(spotConfig,2,"  Entering spotReportDefault");
rawB <- spotGetRawDataMatrixB(spotConfig);
print(summary(rawB));
mergedData <- spotPrepareData(spotConfig)
mergedB <- spotGetMergedDataMatrixB(mergedData, spotConfig)
my.lm <- lm(y ~ ., data= rawB)
print(summary(my.lm))
par(mfrow=c(2,2))
plot(my.lm)
x11()
library(effects)
x11()
plot(allEffects(my.lm), ask=FALSE)
###
x11()
C1 = spotWriteBest(mergedData, spotConfig)
C1 = C1[C1$COUNT == max(C1$COUNT), ]
xNames <- setdiff(names(rawB), c(spotConfig$alg.resultColumn,
```

```
"y"))
B <- NULL
nsens = 20
for (i in 1:nsens) {
B <- rbind(B, data.frame(C1[1, xNames]))
}
fit <- randomForest(rawB[, xNames], rawB$y, ntree = 100)
rwb <- cbind(spotConfig$alg.roi, t(B[1, ]))
names(rwb)[length(rwb)] <- "BEST"
Y <- spotReportSensY(B, fit, spotConfig$alg.roi, nsens)
X = seq(-1, 1, length.out = nsens)
matplot(X, Y, type = "l", lwd = rep(2, ncol(Y)), col = 1:ncol(Y),
xlab = "normalized ROI")
XP = (rwb$BEST - rwb$low)/(rwb$high - rwb$low) * 2 - 1
XP = rbind(XP, XP)
YP = min(Y)
YP = rbind(YP, YP)
matpoints(XP, YP, pch = rep(21, ncol(Y)), bg = 1:ncol(Y),
cex = 2)
legend("topleft", legend = names(Y), lwd = rep(2, ncol(Y)),
lty = 1:ncol(Y), col = 1:ncol(Y), text.col = 1:ncol(Y))
cat(sprintf("\n Sensitivity plot for this ROI:\n"))
print(rwb)
cat(sprintf("\n Best solution found with %d evaluations:\n",
nrow(rawB)))
print(C1[1, ])
spotWriteLines(spotConfig, 2, "\n Leaving spotReportSens")
}
```

Figs. 7 – 10 show some results from the report plugin `myReport2.R`.

6 Templates for Your Project

The following files can be downloaded to set up your own SPOT project ("myProject").

- myProject.conf[28].

  ```
  ### myProject.conf: template for user defined configuration files
  ### Basically, you have to enter the name of the R interface to your algorithm in line
  alg.path="."
  alg.func = "startMyAlgorithm"
  alg.seed = 1235
  spot.seed = 125
  ```

[28] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/myProject.conf

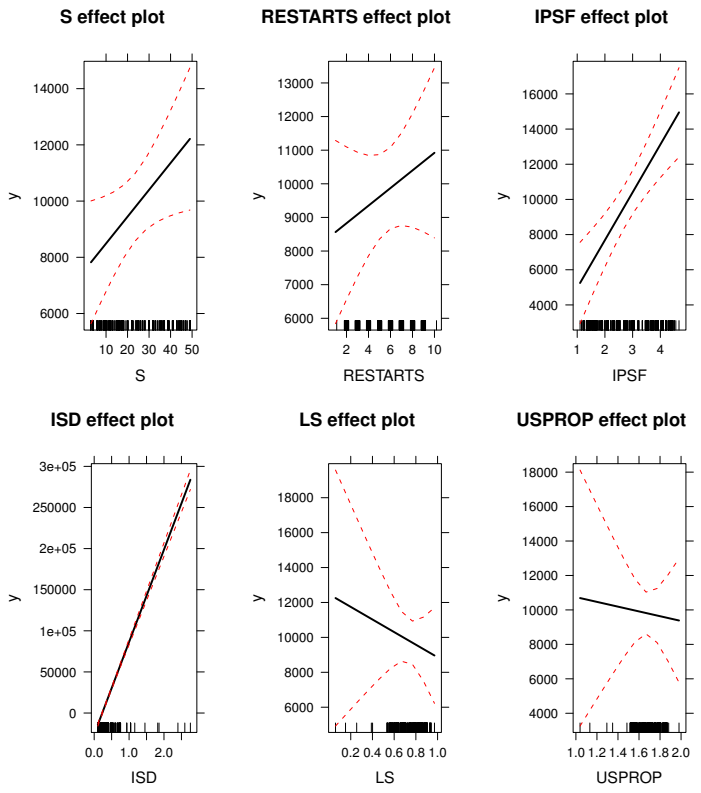

Figure 7: Effect plots from the `report` task `myReport2`

```
auto.loop.steps = 50;
init.design.func = "spotCreateDesignLhd";
init.design.size = 10;
init.design.repeats  = 1;
seq.design.maxRepeats = 5;
seq.design.size = 250
seq.predictionModel.func = "spotPredictRandomForest"
io.verbosity=3
```

- myProject.roi[29].

```
### myProject.roi: roi template for user defined region of interest files
### Variables which should be optimized are listed here.
### Each line describes one variable, its ranges, and type
### Space (" ") is used as a separator
name low high type
```

[29] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/myProject.roi

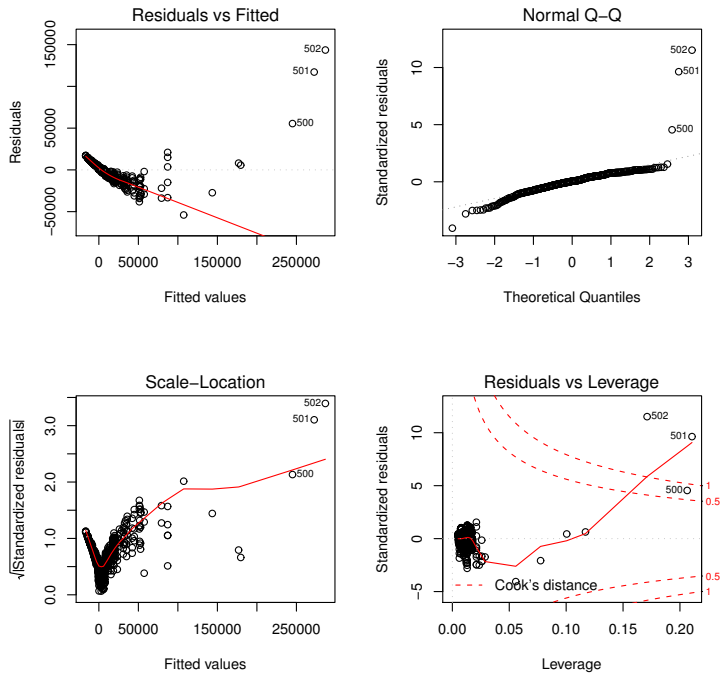

Figure 8: Analysis of the linear model from the `report` task `myReport2`

```
VARA 0 1 INT
VARB 0 100 FLOAT
```

- myProject.apd[30].

```
### myProject.apd: template for user defined apf files
### Define values from the problem design, e.g., dim = 10
var1 = value1
var2 = value2
var3 = value2
```

7 All Inclusive

The file allInclusive.zip[31] contains all files used in this workshop, so you do not have to download the files separately.

[30] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/myProject.apd
[31] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/allInclusive.zip

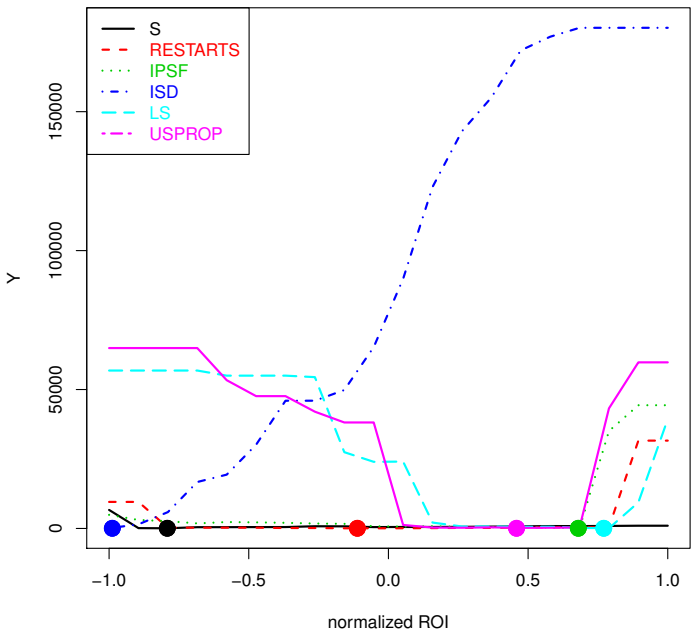

Figure 9: Analysis of the sensitivity analysis from the `report` task `myReport2`

8 Summary

SPOT requires the specification of the following files:

1. *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."

2. *Algorithm design* (APD) files are used to specify parameters used by the algorithm, e.g., problem dimension, objective function, starting point, or initial seed.

3. *Configuration* files (CONF) specify SPOT specific parameters, such as the prediction model or the initial design size.

SPOT will generate the following files:

1. *Design* files (DES) specify algorithm designs. They are generated automatically by SPOT and will be read by the optimization algorithms.

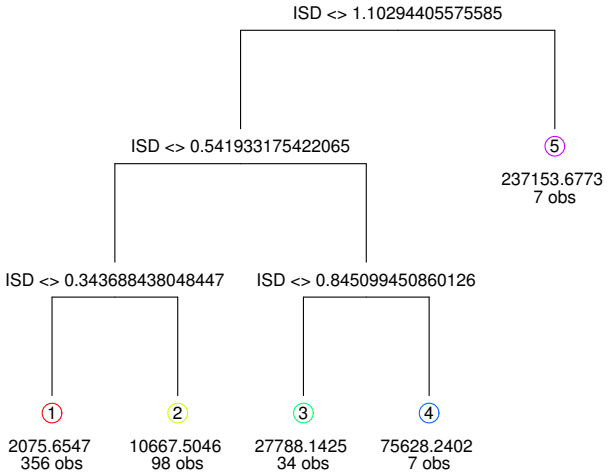

Figure 10: Regression tree from the `report` task `myReport2`

2. 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.

References

Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation—The New Experimentalism*. Natural Computing Series. Springer, Berlin, Heidelberg, New York.

Beyer, H.-G. (2001). *The Theory of Evolution Strategies*. Springer, Berlin, Heidelberg, New York.

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.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology. Wiley, New York NY.