Performing Meta Experiments Using the Sequential Parameter Optimization Toolbox SPOT

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

August 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. The goal of classical tuning is the determination of one good algorithm parameter setting for one specific problem instance. Using SPOT's meta mode, good parameter settings of one algorithm for several problem instances can be determined. This article exemplifies how meta experiments can be performed using the SPOT framework.

1 Introduction

This article describes the experimental setup which is necessary to perform 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

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 $(\vec{x_p}, y_p)$

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.

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

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

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

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

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 (1+1)-ES performs optimally, if the step-sizes σ is modified according to the 1/5th 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;

[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)

3. it terminates, if the function value is smaller than 1e-6;
4. the sphere function is used as the objective function;
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). To enable SPOT's meta mode, a META file has to be specified.

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.

   (iv) *Meta* files (META) specify parameters from the problem design. The algorithm is separately tuned for each setting of the meta parameters. For example, consider tuning a (1+1)-ES on the sphere function for varying problem dimensions, say $n = 10, 20, \ldots, 100$. The problem dimension n is a meta parameter in this case.

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.

   (iii) During the meta mode, the algorithm is tuned on several problem instances. The best configuration from each problem instance is written to the *final best solution file* (FBS).

3.2 Implementation of SPOT's Meta Mode

The implementation of the meta mode is straightforward. For each problem instance, a new sub directory is generated. SPOT is run in this sub directory and the best algorithm configuration from the tuning process is copied to the FBS file in the working directory.

3.3 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.

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

The CONF file to be used in the META mode is very similar to the CONF file from the regular mode. For each problem instance, SPOT generates a subdirectory. Therefore, the path to the algorithm has to be specified, i.e., the variable `alg.path` must contain the absolute path to the algorithm. In our example, `alg.path` is set to
`/home/bartz/workspace/SvnSpot/trunk/ExperimentsBartz.d/Java0Meta.d`.

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

```
alg.path = "/home/bartz/workspace/SvnSpot/trunk/ExperimentsBartz.d/Java0Meta.d"
alg.func = "spotInterfacingTemplateMeta"
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.

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

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

3.4 The Region of Interest

A *region of interest* (ROI) file specifies algorithm parameters and associated lower and upper bounds for the algorithm parameters. There is no difference between the ROI file used in SPOT's regular and SPOT's META mode.

3.4.1 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.

[4] http://advm1.gm.fh-koeln.de/ bartz/SpotMeta.d/java0Meta.conf

```
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[5].

3.5 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 `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.

3.5.1 Modifications of the SPOT Interface to the (1+1)-ES

The meta mode requires a small modification of the interface to the algorithm, because each SPOT run of the meta mode is performed in a new sub directory. Therefore, the call string of the algorithm has to be modified. Replace

```
callString <- paste("java -jar simpleOnePlusOneES.jar"
  ,seed, steps, target, f, n, xp0, sigma0, a, g, px, py, sep = " ")
```

with

```
callString <- paste("java -jar
   /home/bartz/workspace/SvnSpot/trunk/ExperimentsBartz.d/Java0Meta.d/simpleOnePlusOneES.jar"
   , seed, steps, target, f, n, xp0, sigma0, a, g, px, py, sep = " ")
```

Note, you have to replace

```
/home/bartz/workspace/SvnSpot/trunk/ExperimentsBartz.d/Java0Meta.d/simpleOnePlusOneES.jar
```

with the path to your local `simpleOnePlusOneES.jar` file.

3.5.2 Modifications of the APD file for the meta mode

In general, it is not necessary to modify the APD file for the META mode. However, here we are facing a special situation, because we are modifying the problem dimension n. These variations of the n values require an update of the specification of the starting point xp0 of the (1+1)-ES. A ten dimensional problem requires a ten dimensional starting point, whereas a hundred dimensional problem requires a hundred dimensional starting point. Therefore, a few lines of code were added to the interfacing procedure of the (1+1)-ES:

```
if(length(xp0) != n){
                elem <- xp0[[1]]
                xp0 <- paste('"',"[", elem, sep="")
```

[5] http://advm1.gm.fh-koeln.de/ bartz/SpotMeta.d/java0Meta.roi

```
        if(n>1){
                for (kk in 2:n){
                        xp0 <- paste(xp0, elem, sep=",")
                }
        }
        xp0 <- paste(xp0, "]",'"', sep="")
}
```

The setting of the starting point xp0 in the APD file is modified accordingly. Now, we use `xp0 = 1.0` instead of
`xp0 = "[1.0,0.0,1.0,0.0,1.0,0.0,1.0,0.0,1.0,0.0]"` which gives more flexibility. This new setting can be used for any dimension. The modified APD file is shown in Sect. 3.5.3.

3.5.3 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
seed = 123

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

SPOT provides an (1+1)-ESAPD template, which can be downloaded from the workshop's web site[6].

3.6 The Meta File

The configuration of the meta file is simple and similar to the APD file. Each line defines settings for one variable. These settings are defined as a list. Since we are modifying the problem dimension n, the META file looks as follows:

```
n = list(1:50)
```

Note, the user can add more lines to the META file. Each combination of these parameter settings are used. SPOT provides an (1+1)-ESMETA template, which can be downloaded from the workshop's web site[7].

[6] http://advm1.gm.fh-koeln.de/ bartz/SpotMeta.d/java0Meta.apd
[7] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/java0Meta.meta

4 Running SPOT's Meta Mode

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

4.1 Meta Runs with 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.

- SPOT configuration (CONF), e.g., java0Meta.conf[8]
- Region of interest (ROI), e.g., java0Meta.roi[9]
- Algorithm and problem parameters (APD) java0Meta.apd[10]
- The interface to the algorithm. spotInterfacingTemplateMeta.R[11]
- The algorithm, i.e., the `jar` file. simpleOnePlusOneES.jar[12]

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

```
> library(SPOT)
> spot("java0Meta.conf", "meta")
```

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 meta process terminates with the following output:

```
[1] "java0Meta_n50.des"
       y           SIGMANULL           VARA            VARG
 Min.   :22.71   Min.   :0.1199   Min.   :1.007   Min.   : 7.00
 1st Qu.:31.13   1st Qu.:2.7788   1st Qu.:1.056   1st Qu.:58.00
 Median :34.49   Median :3.2686   Median :1.088   Median :72.00
 Mean   :35.46   Mean   :3.2226   Mean   :1.124   Mean   :67.73
 3rd Qu.:39.07   3rd Qu.:3.8006   3rd Qu.:1.102   3rd Qu.:83.00
 Max.   :50.00   Max.   :4.6745   Max.   :1.939   Max.   :94.00

 Best solution found with 103 evaluations:
          Y SIGMANULL     VARA VARG COUNT CONFIG
23 30.76017  2.509786 1.062909   83     5     23
```

[8] http://advm1.gm.fh-koeln.de/ bartz/SpotMeta.d/java0Meta.conf
[9] http://advm1.gm.fh-koeln.de/ bartz/SpotMeta.d/java0Meta.roi
[10] http://advm1.gm.fh-koeln.de/ bartz/SpotMeta.d/java0Meta.apd
[11] http://advm1.gm.fh-koeln.de/ bartz/SpotMeta.d/spotInterfacingTemplateMeta.R
[12] http://advm1.gm.fh-koeln.de/ bartz/simpleOnePlusOneES.jar

A final best solution file (FBS), which contains important information from the meta run, has been written to the working directory. It can be downloaded as java0.res[13] from the workshop's web page.

```
Y SIGMANULL VARA VARG COUNT CONFIG n
1.66994929364791e-11 1.77243773721587 1.47313903401513 50 5 23 1
1.76974481621422e-08 4.13757762718033 1.59132047591917 58 5 17 2
0.000249949023120775 0.767797299823724 1.24111634415295 36 5 28 3
0.00381202963815855 1.87226565002762 1.57047875361983 45 5 19 4
0.0083062486475069 0.725049436075427 1.2164403182324 47 5 29 5
0.0204929156602843 1.60901337389555 1.0696734005278 20 5 15 6
0.0414125716487922 2.6420768151477 1.21203334785718 25 5 20 7
0.170030497403991 3.11775652743802 1.12311313991807 66 5 28 8
0.303833850226578 0.647755945593305 1.1481811835384 35 5 27 9
0.228793829315659 0.770315962409414 1.06917319222819 39 5 26 10
0.36032566622193 1.79691847858382 1.18159458260797 27 5 28 11
0.529091703473727 0.15483357580388 1.04816489817109 21 5 26 12
1.59017957171204 1.374371843467 1.22557020798698 100 5 21 13
1.53346841235075 0.88388465179829 1.08124839728046 82 5 30 14
1.66758926424082 3.36431918830862 1.13585389079526 73 5 16 15
2.80938651706311 3.22419239962334 1.07190223618876 89 5 27 16
1.81535398571898 0.408259528433532 1.06461427548435 30 5 21 17
3.75908073911434 1.62718580127852 1.06491679834202 78 5 26 18
2.30691487221778 0.174113577309996 1.00983339973167 70 5 11 19
4.36070376993907 3.79613268197458 1.1593797170762 34 5 15 20
3.98349143775901 0.322219615078066 1.09240018011723 86 5 22 21
4.59500657124994 0.594469441467989 1.05669404837769 51 5 23 22
5.73073754875688 1.44470423722174 1.05566112561524 67 5 27 23
5.52095265382445 0.162522591060773 1.00735978167411 34 5 17 24
6.42185007423266 0.193069812484458 1.06613574731536 69 5 25 25
8.2245623079002 3.69819622882549 1.10985683534481 97 5 19 26
8.3078065049788 0.161955996157322 1.05821049421187 37 5 19 27
11.4679870926236 3.82484304474983 1.08829998440947 7 5 19 28
12.4512753858261 3.05403130636755 1.10779903379455 44 5 16 29
9.64608770371497 0.284192788536847 1.04912395175919 41 5 17 30
9.73772022960643 0.10114390837783 1.00906914806087 62 5 24 31
10.5597011557683 1.72445825159233 1.07217559588514 87 5 14 32
10.3672357078329 0.134599235072359 1.01938627839554 93 5 19 33
18.1913106357228 1.58889361474207 1.03560492046829 51 5 29 34
14.1955885102874 0.568881144746486 1.01537999345176 94 5 19 35
14.7899472156359 0.111073942558374 1.06076744625717 97 5 15 36
18.8861950215593 2.07688707274152 1.09084986476228 50 5 10 37
23.6324271290547 0.698208594874013 1.68827389127109 67 5 5 38
17.1204625181177 0.262166819186881 1.02370456977002 51 5 21 39
22.2447438564846 3.6468800928615 1.06743559596874 9 5 24 40
20.3085146925552 0.817548313462362 1.03124808251765 6 5 16 41
23.2342172915281 3.80829033931 1.0696718550818 4 5 29 42
29.0779101086268 1.61163491582554 1.37209779803921 27 5 26 43
17.7119480987105 0.142815414902847 1.00902041078079 95 5 20 44
25.9085994525956 4.08213876903672 1.07963561266102 37 5 19 45
25.5711905535851 0.338241372768674 1.13696653940249 18 5 23 46
28.1548588619646 0.751029239553493 1.09044733802509 54 5 27 47
33.2487176101654 4.07324539682129 1.19218263512291 87 5 21 48
31.1223365561554 3.4563801769861 1.05835333938897 92 5 30 49
30.7601655678372 2.50978620407097 1.06290862832591 83 5 23 50
```

[13] http://advm1.gm.fh-koeln.de/ bartz/SpotMeta.d/java0Meta.fbs

4.2 The Report Task for the Meta Mode

If SPOT's termination criterion is fulfilled, a meta report can be generated. A default meta report function is included in the SPOT package: `spotReportMetaDefault`.

```
spot("javaOMeta.conf","rep")
```

It generates the output shown in Fig. 1.

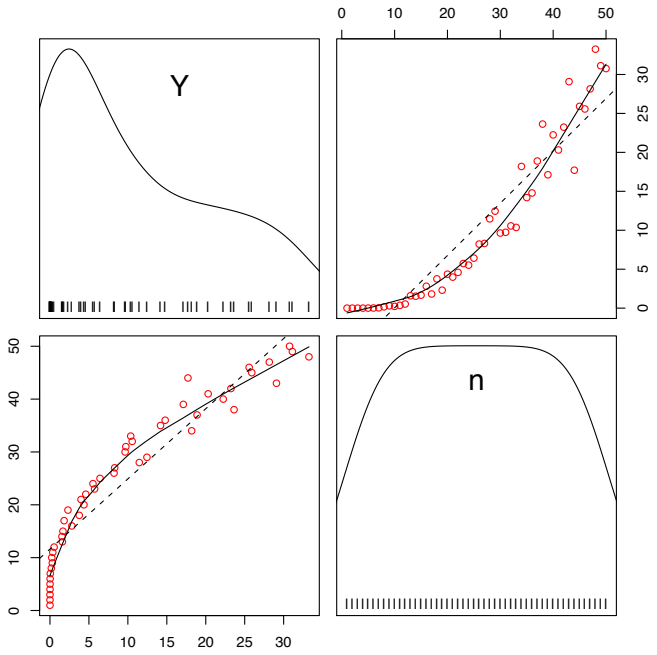

Figure 1: Default meta report output

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

5 All Inclusive

The file allInclusive.zip[14] contains all files used in this workshop, so you do not have to download the files separately.

[14] http://advm1.gm.fh-koeln.de/ bartz/SpotPerforming.d/allInclusive.zip

6 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.

4. *Meta* file (META) to specify meta variables.

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.

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.

3. (FBS) files contain information from the meta runs.

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.