Writing Interfaces for 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 existing optimization algorithms can be integrated into the SPOT framework

1 Introduction

This article illustrates how existing algorithms can be integrated into 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 algorithms written in JAVA can be integrated into the SPOT framework. Two JAVA implementations of evolution strategies are discussed.

- A simple (1+1)-ES in Sec. 2.1
- Hansen's CMA-ES in Sec. 2.2

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
```

2.1.3 The SPOT Interface for the (1+1)-ES

SPOT provides an interface template, which can be downloaded from the workshop's web site[3]. The user has to modify the following pieces of code.

1. Add user defined parameters. These parameters will be extracted from a data frame and passed to the algorithm.
2. Define a call string. This string is executed to start the algorithm.
3. Extract the result from the algorithm.

The corresponding R code is shown here:

[3] http://advm1.gm.fh-koeln.de/ bartz/SpotInterfacing.d/spotInterfacingTemplate.R

```
spotAlgStartOnePlusOneEsJava <- function(io.apdFileName
, io.desFileName
, io.resFileName){
## read default problem design
source(io.apdFileName,local=TRUE)
## read doe/dace etc settings:
print(io.desFileName)
des <- read.table(io.desFileName
, sep=" "
, header = TRUE
);
config<-nrow(des);
attach(des)
for (k in 1:config){
if(des$REPEATS[k]>=1){
for (i in 1:des$REPEATS[k]){
### 1. Add user defined parameters here:
if (exists("SIGMANULL")){
sigma0 <- des$SIGMANULL[k]
}
if (exists("VARA")){
a <- des$VARA[k]
}
if (exists("VARG")){
g <- round(des$VARG[k])
}
## End of user defined section.
conf <- k
if (exists("CONFIG")){
conf <- des$CONFIG[k]
}
spotStep<-NA
if (exists("STEP")){
spotStep <- des$STEP[k]
}
seed <- des$SEED[k]+i
### 2. The call string has to be modified by the user:
callString <- paste("java -jar simpleOnePlusOneES.jar",
seed, steps, target, f, n, xp0, sigma0, a, g, px, py, sep = " ")
### End of user defined call string.
y <-system(callString, intern= TRUE)
res <- NULL
res <- list(Y=y,
### 3. User specific parameter values have to be added to the list:
SIGMANULL=sigma0,
VARA=a,
```

```
VARG=g,
### End of user specific parameter values.
Function=f,
MAXITER=steps,
DIM=n,
TARGET=target,
SEED=seed,
CONFIG=conf
)
if (exists("STEP")){
res=c(res,STEP=spotStep)
}
res <-data.frame(res)
colNames = TRUE
if (file.exists(io.resFileName)){
colNames = FALSE
}
## quote = false is required for JAVA
write.table(res
, file = io.resFileName
, row.names = FALSE
, col.names = colNames
, sep = " "
, append = !colNames
, quote = FALSE
);
colNames = FALSE
} # end for i
} # end if(des$REPEATS[k]>=1)
} #end for k
detach(des)
}
```

This interface can be used to call the (1+1)-ES from SPOT.

SPOT generates designs (parameter settings for the (1+1)-ES. These designs are based on information provided by the user. How these information can be specified is described in the tutorial *Performing experiments in the SPOT environment*, which can be downloaded from the workshop's web site[4].

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.

[4] http://advm1.gm.fh-koeln.de/ bartz/geccoworkshop.html

2.2.2 The JAVA Implementation of the CMA-ES

The following implementation uses version

```
Last change: $Date: 2010-03-17 11:43:16 +0100 (Wed, 17 Mar 2010)
```

of Hansen's CMA-ES, see `http://www.lri.fr/~hansen/javadoc/index.html`. The function `warnings()` was modified in the following manner:

```
private void warning(String s) {
        // commented for SPOT: println(" CMA-ES warning: " + s);
    }
```

This modification has no negative impact on CMA-ES's performance or behaviour. No further modifications were performed.

We generated a jar file, cmaEs.jar[5], which implements the CMA-ES. It uses the following command-line arguments.

```
Example usage:
---
java -jar cmaEs.jar 1 22 100000 50 0 0.0 9 1 2.0 0.5 0.3  null null true

Usage:
---
usage: cmaEs seed d evals f frot arot s restrts ipsf tx isd ls us
  where
    seed    random seed ( e.g. 12345 )
    d       search space dimension ( e.g. 22 )
    evals   maximum number of fitness function evaluations ( e.g. 100000 )
    f       objective function index ( e.g. 50, 10 is Sphere )
    frot    objective function rotation (e.g. 0)
    arot    objective function axis rotation ( e.g. 0.0 )
    s       population size ( e.g. 9 )
    restarts number of restarts ( e.g. 1 )
    ipsf    increase population size factor ( e.g. 2.0 )
    tx      typical X ( e.g. 0.5 )
    isd     initial standard deviations ( e.g. 0.3 )
    ls      lower standard deviations (step sizes)
            ( null               = defaults are chosen by CMA-ES,
              c(ls_0,...,ls_n)   = the vector [ls_0,...,ls_n],
              [ls_0,...,ls_n]    = the vector [ls_0,...,ls_n],
              ls_0,...,ls_n      = the vector [ls_0,...,ls_n] )
    us      upper standard deviations (step sizes)
            ( null               = defaults are chosen by CMA-ES,
              c(us_0,...,us_n)   = the vector [us_0,...,us_n],
              [us_0,...,us_n]    = the vector [us_0,...,us_n],
```

[5] http://advm1.gm.fh-koeln.de/ bartz/cmaEs.jar

Table 3: JAVACMA-ES: Parameters as reported by the algorithm

Name	Parameter
s	population size (e.g. 9)
restarts	number of restarts (e.g. 1)
ipsf	increase population size factor (e.g. 2.0)
isd	initial standard deviations (e.g. 0.3)
ls	lower standard deviations (step sizes)
us	upper standard deviations (step sizes)
seed	random seed (e.g. 12345)
d	search space dimension (e.g. 22)
evals	maximum number of fitness function evaluations (e.g. 100000)
f	objective function index (e.g. 50, 10 is Sphere)
frot	objective function rotation (e.g. 0)
arot	objective function axis rotation (e.g. 0.0)
tx	typical X (e.g. 0.5)

```
            us_0,...,us_n      = the vector [us_0,...,us_n] )
   verbose print status messages during runs ( e.g. true )
```

The JAVACMA-ES algorithm uses the parameters from Tab. 3. The first parameters are related to the algorithm design and will be specified in SPOT's ROI file, whereas the latter belong to the problem design and will be specified in SPOT's APD file.

Based on these information about the algorithm design, we can modify SPOT's generic interface to existing algorithms, which can be downloaded from the workshop's web site[6]. The modified version looks like follows. It can be downloaded from the workshop's web site[7]. Note, we replaced the variable name `ls` with `varLs`, because `ls()` is a basic R function and it is a bad idea to use existing function names as variable names.

```
spotAlgStartCmaEsJava <- function(io.apdFileName
, io.desFileName
, io.resFileName){
## read default problem design
source(io.apdFileName,local=TRUE)
## read doe/dace etc settings:
print(io.desFileName)
des <- read.table(io.desFileName
, sep=" "
, header = TRUE
);
config<-nrow(des);
```

[6] http://advm1.gm.fh-koeln.de/ bartz/SpotInterfacing.d/spotInterfacingTemplate.R
[7] http://advm1.gm.fh-koeln.de/ bartz/SpotInterfacing.d/spotAlgStartCmaEsJava.R

```
attach(des)
for (k in 1:config){
if(des$REPEATS[k]>=1){
for (i in 1:des$REPEATS[k]){
### 1. Add user defined parameters here:
if (exists("S")){
s <- round(des$S[k])
}
if (exists("RESTARTS")){
restarts <- round(des$RESTARTS[k])
}
if (exists("IPSF")){
ipsf <- des$IPSF[k]
}
if (exists("ISD")){
isd <- des$ISD[k]
}
if (exists("LS")){
varLs <- des$LS[k]
}
if (exists("USPROP")){
usprop <- des$USPROP[k]
us <- usprop * (1+varLs)
}
## End of user defined section.
conf <- k
if (exists("CONFIG")){
conf <- des$CONFIG[k]
}
spotStep<-NA
if (exists("STEP")){
spotStep <- des$STEP[k]
}
seed <- des$SEED[k]+i
### 2. The call string has to be modified by the user:
callString <- paste("java -jar cmaEs.jar",
seed, d, evals, f, frot, arot, s, restarts, ipsf, tx, isd, varLs, us, "null", "null", "false
sep = " ")
### End of user defined call string.
y <-system(callString, intern= TRUE)
res <- NULL
res <- list(Y=y,
### 3. User specific parameter values have to be added to the list:
S=s,
RESTARTS=restarts,
IPSF = ipsf,
```

```
ISD = isd,
LS = varLs,
USPROP = usprop,
### End of user specific parameter values.
Function=f,
EVALS=evals,
D=d,
SEED=seed,
CONFIG=conf
)
if (exists("STEP")){
res=c(res,STEP=spotStep)
}
res <-data.frame(res)
colNames = TRUE
if (file.exists(io.resFileName)){
colNames = FALSE
}
## quote = false is required for JAVA
write.table(res
, file = io.resFileName
, row.names = FALSE
, col.names = colNames
, sep = " "
, append = !colNames
, quote = FALSE
);
colNames = FALSE
} # end for i
} # end if(des$REPEATS[k]>=1)
} #end for k
detach(des)
}
```

This interface can be used to call the CMA-ES from SPOT.

SPOT generates new design points (parameter settings for the CMA-ES) 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

How these information can be specified is described in the tutorial *Performing experiments in the SPOT environment*, which can be downloaded from the

workshop's web site[8].

3 Summary

SPOT includes a template which can be used as an interface to existing algorithms. It can be downloaded from the workshop's web site[9].

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.

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.

[8] http://advm1.gm.fh-koeln.de/ bartz/geccoworkshop.html
[9] http://advm1.gm.fh-koeln.de/ bartz/SpotInterfacing.d/spotInterfacingTemplate.R