Experimental Analysis of Optimization Heuristics Using R

Marco Chiarandini

October 5, 2009

Abstract

This document introduces the reader to the analysis of results of computational experiments on heuristic algorithms for optimization by means of R, the free software environment for statistical computing and graphics. In the first chapter relevant R commands are listed. In the second chapter, a basic analysis is developed on two example studies. In chapter three, the setup and use of the racing method for configuration and tuning of the algorithms is described. In chapter four, elements from survival analysis and extreme value theory are used to model the behaviour of the algorithms. In chapter five, advanced analysis from experimental design are reviewed.

This is a very preliminary version and suggestions and corrections are very welcome.

Contents

Chapter 1

Introduction

1.1 R Download and Installation

- R web page (`http://www.r-project.org/`), select Download - CRAN → Select a mirror → Select in the first frame, named 'Download and Install R', your operating system.

- Windows: → select 'base' → click on 'R-?.?.?-win32.exe'.

- Under Linux Ubuntu:

  ```
  sudo apt-get install r-base
  ```

- Alternatively, go to `http://cran.r-project.org/doc/manuals/R-admin.html` and follow description there.

- A package is a collection of functions and programs that can be used within R. To install new packages type from R command line:

  ```
  > install.packages("lattice")
  ```

- Package `Rcmdr` provides a graphical interface to R.

- R documentation: see `http://www.sbtc.ltd.uk/freenotes.html`: 'Getting Started in R' by Saghir Bashir, and the references under the R link `http://cran.r-project.org/manuals.html`.

- Emacs users can find a mode for R at `http://ess.r-project.org/`.

1.2 Basic commands

See help page for details on all commands listed here.

1.2.1 System commands

- `R` starts R from command line.

- `library(Rcmdr)` loads the `Rcmdr` package and starts graphical interface.

- `q()` quits your R session.

- `options(width=120)` determines the position of the line break in R output.

- `?plot` a question mark followed by the name of the function opens the help page relative to that function.

- `help.start()` opens a browser with documentation.

- `example(plot)` calls one or more examples implemented for the function.

- `demo(package.name)` calls a demonstration for the functionality of the defined package.

- `ls()` provides a list of objects in the current R workspace.

- `sessionInfo()` prints version information about R and attached or loaded packages (see also `Sys.getlocale()`)

1.2.2 Data and operations

- `read.table()` and `write.table()` read and write from text file.

- `load()` and `save()` load and save R objects.

- `^`, `%%`, `%/%` operators for power, modulus and integer part of the division, respectively.

- `c()` function used to collect things together into a vector, example: `x <- c(1,2,3)`

- `vector()`, `matrix()`, `array()`, `data.frame()`, `list()` data structures. Tests or coercions can be done with `is.data.frame()`, `as.data.frame()`, respectively

- `integer() double()` data types. Can be queried or coerced with `is.integer()` and `as.integer()`

- `str()` compactly displays the structure of an arbitrary R object

- `mean(). median()`, `sum()`, `var()`, `summary()`, `interquartile() range()` compute sample statistics.

- `factor()` offers an alternative way of storing character data. For example a *factor* can have four elements and two *levels*:

```
> algorithms <- c("greedy", "grasp", "greedy", "grasp")
> algorithms

[1] "greedy" "grasp"  "greedy" "grasp"
```

```
> algorithms <- factor(algorithms)
> algorithms

[1] greedy grasp  greedy grasp
Levels: grasp greedy
```

- Generate sequences of integers by:

```
> 1:12

 [1]  1  2  3  4  5  6  7  8  9 10 11 12

> seq(1, 21, by = 2)

 [1]  1  3  5  7  9 11 13 15 17 19 21

> rep(3, 12)

 [1] 3 3 3 3 3 3 3 3 3 3 3 3
```

- expand.grid() creates a data frame from all combinations of factors

```
> (table <- expand.grid(algorithm = algorithms, instance = c("A",
+     "B")))

  algorithm instance
1    greedy        A
2     grasp        A
3    greedy        A
4     grasp        A
5    greedy        B
6     grasp        B
7    greedy        B
8     grasp        B
```

- paste(), substr() strsplit() work with strings. The first concatenates, the second returns substrings within two positions, the third splits strings. Example:

```
> colors <- c("red", "yellow", "green")
> paste(colors, "flowers")

[1] "red flowers"    "yellow flowers" "green flowers"

> paste("several ", colors, "s", sep = "")

[1] "several reds"    "several yellows" "several greens"

> paste("I like", colors, collapse = ", ")

[1] "I like red, I like yellow, I like green"

> substr(colors, 1, 2)

[1] "re" "ye" "gr"

> unlist(strsplit("a.b.c", "\\."))

[1] "a" "b" "c"
```

4

1.2.3 Graphics

- `plot()`, `lines()`, `points()`, `curve()`, `hist()`, `barplot()`, `boxplot()`, graphics functions from the base installation

- `par()` lists and changes graphic setting

- `colors()` the colors available in R

- Graphics are device independent. Type `?device` to see on which device they can be printed and how.

- `dev.copy(dev=pdf,file='Rplot.pdf')` copies the graphic in a pdf file. Remember to close the pipeline with `dev.off()`

- `lattice` and `ggplot` two packages for multivariate conditional plots. Try `demo()` on them for a demonstration of their facilities. The package `lattice` is thoroughly explained in reference [6].

1.2.4 Advanced Commands

- `stack`, `unstack`, `reshape` and `merge` are useful functions for rearragging data.

```
> table$res <- runif(8, 0, 1)
> reshape(table, timevar = "algorithm", idvar = "instance", direction = "wide")

  instance res.greedy res.grasp
1        A  0.6363240 0.9138498
5        B  0.9843632 0.9745345

> tab <- data.frame(instance = c("A", "B"), opt = c(1, 2))
> merge(table, tab, by.x = "instance", by.y = "instance")

  instance algorithm       res opt
1        A    greedy 0.63632395   1
2        A     grasp 0.91384982   1
3        A    greedy 0.82857233   1
4        A     grasp 0.02554118   1
5        B    greedy 0.98436320   2
6        B     grasp 0.97453453   2
7        B    greedy 0.63009546   2
8        B     grasp 0.31191750   2
```

- `which()` returns the index of an element in a vector.

  Example: `which(!(colnames(table) %in% c("algorithm","instace")))`

- `which.min()` returns the index of the minnmal element

- `missing()` tests whether a value was specified as an argument to a function.

- `stopifnot()` stops if one of a list of conditions is not true reporting an error with the first violated condition.

- `debug()` debugs a function. `Q` to exit debug mode.

- package `xtable` provides a function to convert an R object into a LaTeX or an HTML table.

```
> library(xtable)
> xtable(table)

% latex table generated in R 2.9.1 by xtable 1.5-5 package
% Mon Aug 31 19:41:09 2009
\begin{table}[ht]
\begin{center}
\begin{tabular}{rllr}
  \hline
 & algorithm & instance & res \\
  \hline
1 & greedy & A & 0.64 \\
  2 & grasp & A & 0.91 \\
  3 & greedy & A & 0.83 \\
  4 & grasp & A & 0.03 \\
  5 & greedy & B & 0.98 \\
  6 & grasp & B & 0.97 \\
  7 & greedy & B & 0.63 \\
  8 & grasp & B & 0.31 \\
   \hline
\end{tabular}
\end{center}
\end{table}
```

- `Sweave` is a tool that allows to embed the R code in latex documents[7, 8]. It implements the literate programming concept.

1.3 Memory issues

- `.Machine` and `.Platform` variables holding information on the numerical characteristics of the machine and information on the platform on which R is running. Consult these for information such as the largest double or integer and the machine's precision. See also `?"Memory-limits"`

- `object.size()` provides an estimate of the memory that is being used to store an R object.

- `gc(verbose=TRUE)` causes a garbage collection to take place and prints memory usage statistics

Chapter 2

Comparing Distributions of Experimental Results

In this chapter, we discuss basic ways to visualize results for the analysis of computational experiments in optimization. We have chosen two running examples. The first considers the combinatorial optimization problem of finding a good approximation to the chromatic number of a graph and evaluates different heuristics for it. The second, is a continuous optimization problem drawn from the field of statistics and considers both the implementation of the solvers and their assessment.

2.1   Example 1: Discrete Optimization

In this example, we wish to assess different heuristics for the graph coloring problem, ie, finding a good approximation of the chromatic number of a graph.

2.1.1   Univariate Analysis

We have collected results for different heuristics run for the same amount of time on a class of instances randomly generated. By construction we know that each instance has a coloring that uses a certain number of colors. This number is encoded in the name of the instance. We load the data as an R object and produce a first exploratory plot to investigate the distribution of data. We may choose one of the three ways to represent distributions depicted in Figure 2.1. All summary statistics of the distributions, such as location measures (mean, median, max) or dispersion measures (variance, standard deviation, interquartile) necessarily omit information. Inspection of the full distribution of data is therefore a good practice to understand which of these summary statistics better represent the data at hand.

Boxplots are advantages because they allow to fit easily several distributions on the same plot.

The data are stored in a text file organized in columns. The import in a data frame is straightforward. By means of `str` we gain a view on the data. There are three algorithms and ten instances. For each instance and algorithm ten results are collected. We rename the columns according to the information they provide and produce the boxplots, shown in Figure 2.2, upper left plot.
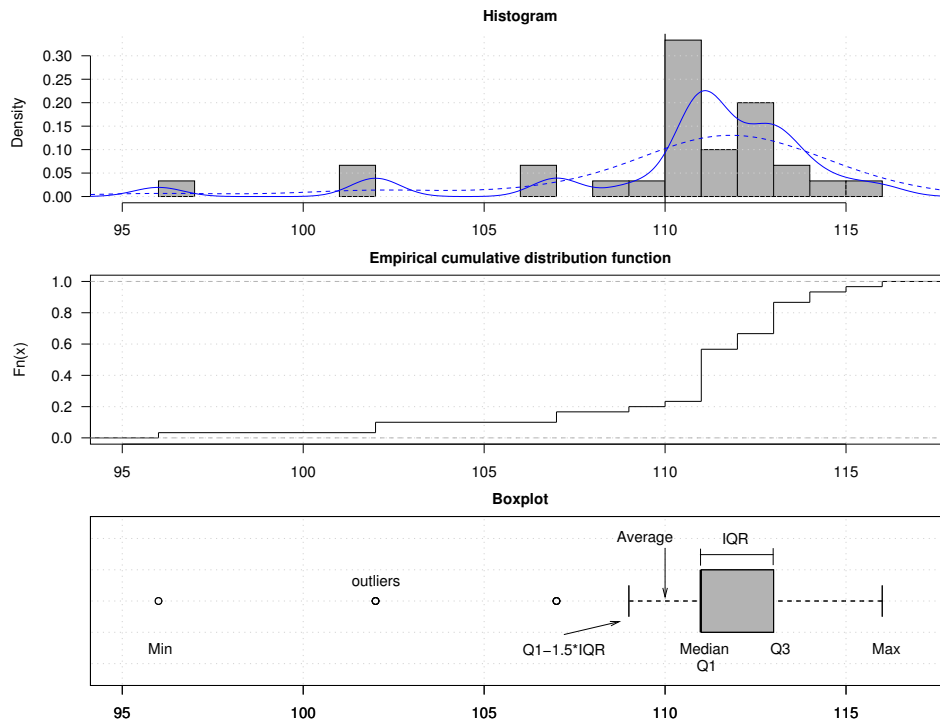
Figure 2.1: Histogram, density function, empirical distribution function and boxplots are different ways to look at the distribution of empirical data.

```
> G <- read.table("Data/TS-class-G.txt")
> names(G) <- c("alg", "inst", "trial", "sol", "time", "best")
> G[1:5, ]
  alg             inst trial sol time best
1 TS1 G-1000-0.5-30-1.1     1  59 9.90   30
2 TS1 G-1000-0.5-30-1.1     2  64 9.74   30
3 TS1 G-1000-0.5-30-1.1     3  64 9.91   30
4 TS1 G-1000-0.5-30-1.1     4  68 9.95   30
5 TS1 G-1000-0.5-30-1.1     5  63 9.91   30

> str(G)

'data.frame':       300 obs. of  6 variables:
 $ alg  : Factor w/ 3 levels "TS1","TS2","TS3": 1 1 1 1 1 1 1 1 1 1 ...
 $ inst : Factor w/ 10 levels "G-1000-0.5-30-1.1",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ trial: int  1 2 3 4 5 6 7 8 9 10 ...
 $ sol  : int  59 64 64 68 63 63 65 65 71 62 ...
 $ time : num  9.9 9.74 9.91 9.95 9.91 ...
 $ best : int  30 30 30 30 30 30 30 30 30 30 ...

> par(mfrow = c(2, 2), las = 1, font.main = 1, mar = c(2, 3, 3,
    1))
> boxplot(sol ~ alg, data = G, horizontal = TRUE, main = "Original data")
```

This way of aggregating data tends to confound effects because instances may have different scales. For example an instance can have optimal solution 20 another 100 and if the algorithms are good they may find solutions close to these values, thus aggregating data would mix
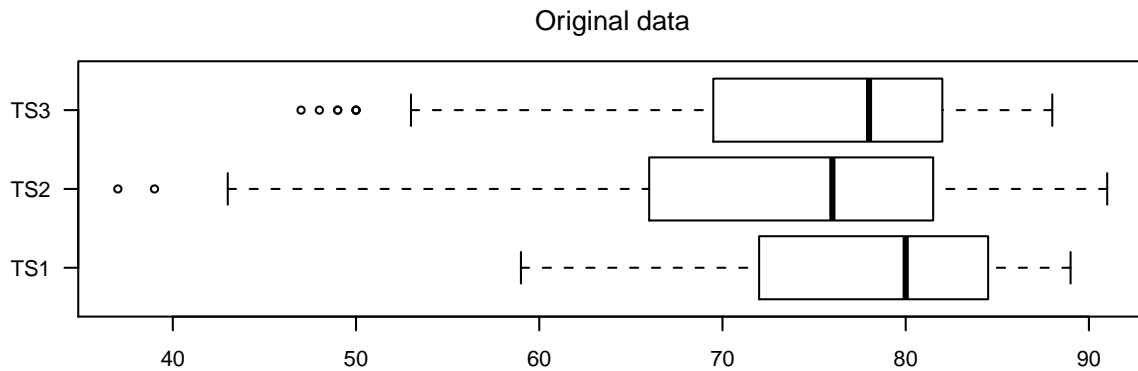
Figure 2.2: Aggregate analysis with original data

number that have little to do together. Let's have a closer look at these data by observing the distributions separately on each instance. For this we need the multi-panel functionality of the package `lattice`.

```
> library(lattice)
> print(bwplot(alg ~ sol | inst, data = G, layout = c(5, 2)))
```

It is clear from the resulting plot in Figure 2.3 that there are differences among the algorithms. Hence an aggregate analysis must make use of some data transformation in order to uniform instance scales.

We review four possible transformations, standard error, relative error, invariant error and ranks. The boxplots obtained by these transformations are reported in Figure 2.4

Standard error   This is the classical transformation used in statistics indicating the distance from the mean value in terms of standard deviation. In R the function `scale` takes care of the transformation.

```
> T1 <- split(G$sol, list(G$inst))
> T2 <- lapply(T1, scale, center = TRUE, scale = TRUE)
> T3 <- unsplit(T2, list(G$inst))
> T4 <- split(T3, list(G$alg))
> T5 <- stack(T4)
> boxplot(values ~ ind, data = T5, horizontal = TRUE, main = expression(paste("Standard error: ",
    frac(x - bar(x), sqrt(sigma)))))
> library(latticeExtra)
> print(ecdfplot(~T5$values, groups = T5$ind, main = expression(paste("Standard error: ",
    frac(x - bar(x), sqrt(sigma))))))
```

More concisely, the addition of a column with the standard error could have been achieved as follows:

```
> G$scale <- 0
> split(G$scale, G$inst) <- lapply(split(G$sol, G$inst), scale,
    center = TRUE, scale = TRUE)
```
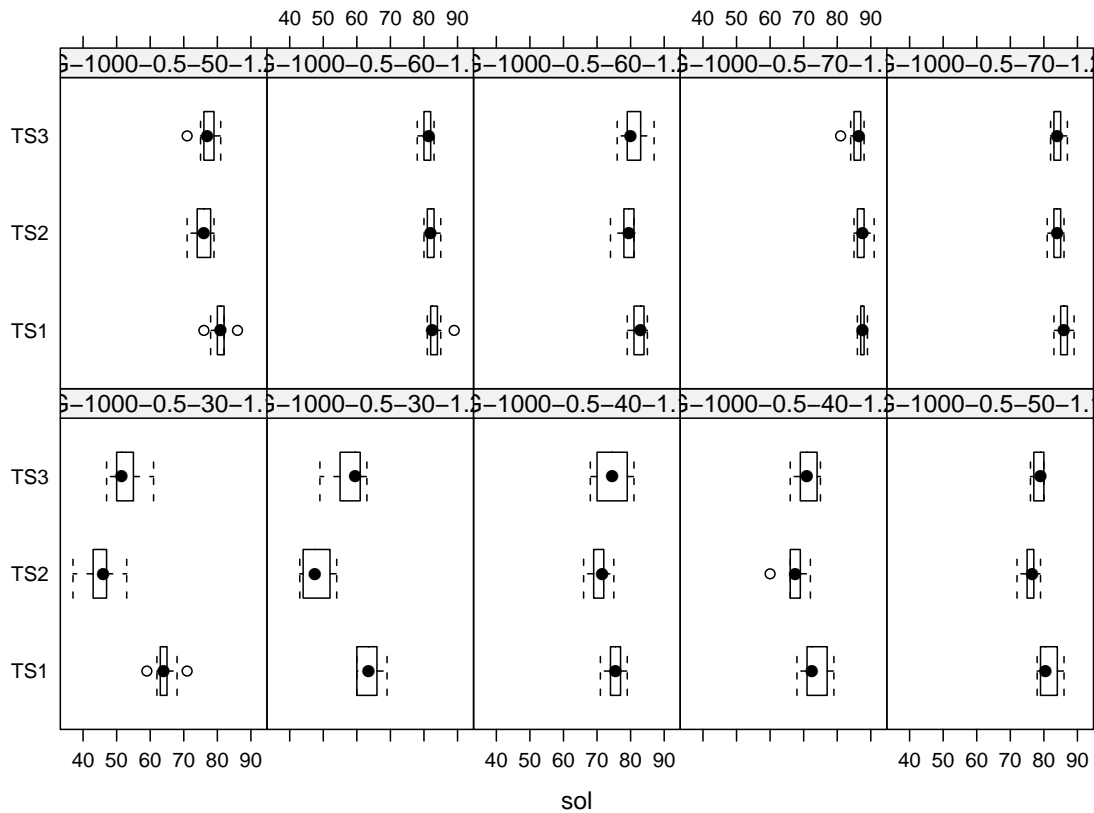
Figure 2.3: Boxplots of solution found by the algorithms on instance basis

Relative error   Where lower bounds, best solutions or optimal solutions are known it is possible to use a measure more typically found in optimization, i.e., the relative error or optimality gap.

```
> G$err2 <- (G$sol - G$best)/G$best
> boxplot(err2 ~ alg, data = G, horizontal = TRUE, main = expression(paste("Relative error: ",
      frac(x - x^(best), x^(best)))))
> print(ecdfplot(~G$err2, groups = G$alg, main = expression(paste("Relative error: ",
      frac(x - x^(best), x^(best))))))
```

Invariant error   The relative error is not invariant with repsct to linear transformation of the input data. It can be proved that substituting in the denominator x^{best} with $x^{worst} - x^{best}$, where x^{worst} is the worst possible result on the instance, the resulting mearue is invariant. Often x^{worst} is unknown and it might be hard to determine. Hence, we use as surrogate of x^{worst} the median solution returned by the simplest algorithm for the graph coloring, that is, the ROS heuristic.

```
> H <- read.table("Data/ROS-class-G.txt")
> names(H) <- c("alg", "inst", "trial", "sol", "time")
```

10

```
> X <- aggregate(H$sol, list(H$inst), median)
> G$ref <- sapply(G$inst, function(x) X[X$Group.1 == x, ]$x)
> G$err3 <- (G$sol - G$best)/(G$ref - G$best)
> boxplot(err3 ~ alg, data = G, horizontal = TRUE, main = expression(paste("Invariant error: ",
      frac(x - x^(best), x^(worst) - x^(best)))))
> print(ecdfplot(~G$err3, groups = G$alg, main = expression(paste("Invariant error: ",
      frac(x - x^(best), x^(worst) - x^(best))))))
```

Ranks The last transformation that we consider consists in ranking the data within each instance.

Let k be the number of candidate solvers and $X_{i1r}, X_{i2r}, \ldots, X_{ikr}$ the results they obtain on an instance i in the run r. Results over the r runs can be arranged as follows

<center>Solvers</center>

Instance	1	2	...	k
1	$X_{1,1,1}, \ldots, X_{1,1,r}$	$X_{1,2,1}, \ldots, X_{1,1,r}$	...	$X_{1,k,1}, \ldots, X_{1,1,r}$
2	$X_{2,1,1}, \ldots, X_{2,1,r}$	$X_{2,2,1}, \ldots, X_{2,1,r}$	...	$X_{2,k,1}, \ldots, X_{2,1,r}$
3	$X_{3,1,1}, \ldots, X_{3,1,r}$	$X_{3,2,1}, \ldots, X_{3,1,r}$	...	$X_{3,k,1}, \ldots, X_{3,1,r}$
...	...	...	...	...
10	$X_{10,1,1}, \ldots, X_{3,1,r}$	$X_{10,2,1}, \ldots, X_{10,1,r}$	...	$X_{10,k,1}, \ldots, X_{10,1,r}$

The results are transformed in ranks within each instance i assigning to each $X_{i,j,r}$ a value $R(X_{i,j,r})$ from 1 to rk. We can compute the sum of the ranks R_j and the average rank $\overline{R}_j$ for a solver j by

$$R_j = \sum_{i=1}^{10} \sum_{l=1}^{r} R(X_{i,j,r}) \qquad \overline{R}_j = \frac{R_j}{rk}$$

.

```
> G$rank <- 0
> split(G$rank, G$inst) <- lapply(split(G$sol, G$inst), rank)
> boxplot(rank ~ alg, data = G, horizontal = TRUE, main = "Ranks")
> print(ecdfplot(~rank, groups = alg, data = G, main = "Ranks"))
```

Numerical results and tables Once we observed the distributions we may choose the statistics worth to be reported in a table with numerical details. Since distributions are not perfectly symmetric we decide to use the median as summary measure of location and the interquartile, that is, the difference between the first and the third quartile[1]. Since we are also interested in the best results we also report the minimum of the results found. Then we use `xtable` to produce the table in LaTeX.

```
> df1 <- aggregate(G$sol, list(inst = G$inst, alg = G$alg), median)
> df2 <- aggregate(G$sol, list(inst = G$inst, alg = G$alg), IQR)
> df3 <- aggregate(G$sol, list(inst = G$inst, alg = G$alg), min)
> df <- cbind(df3, iqr = df1$x, min = df2$x)
```

---

[1]The p-quantile corresponds to the value x of a distribution such that $\Pr[X < x] = p$. The first quartile is the 25%-quantile, i.e., the value x such that $\Pr[X < x] = 0.25$.
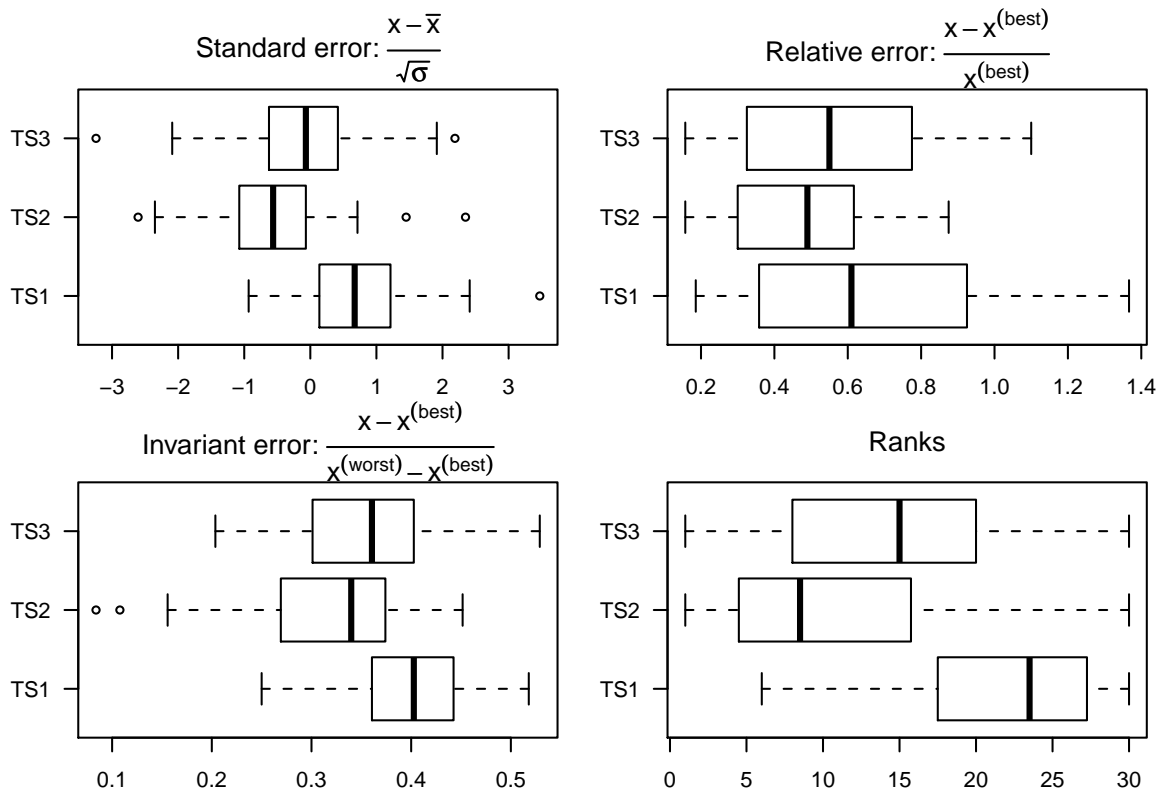
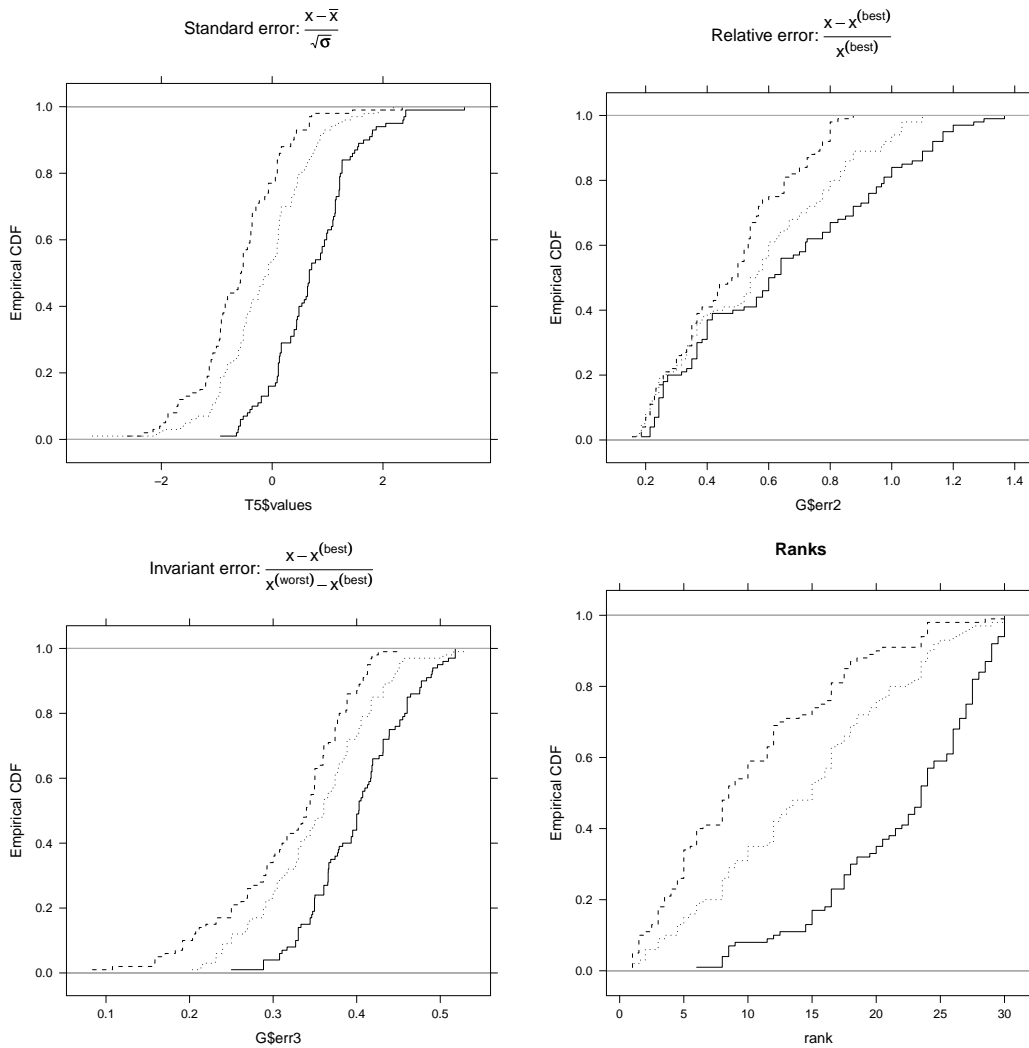Figure 2.4: Aggregate analysis with different performance measures

Figure 2.5: Analysis by means of empirical cumulative distribution functions for different performance measures

inst	x.TS1	iqr.TS1	min.TS1	x.TS2	iqr.TS2	min.TS2	x.TS3	iqr.TS3	min.TS3
G-1000-0.5-30-1.1	59	64.00	2.00	37	46.00	3.25	47	51.50	4.75
G-1000-0.5-30-1.2	60	63.50	5.50	43	47.50	7.00	49	59.50	6.00
G-1000-0.5-40-1.1	71	75.50	2.75	66	71.50	2.75	68	74.50	7.50
G-1000-0.5-40-1.2	68	72.50	5.25	60	67.50	3.00	66	71.00	4.00
G-1000-0.5-50-1.1	78	80.50	4.50	72	76.50	1.75	76	79.00	2.75
G-1000-0.5-50-1.2	76	81.00	2.00	71	76.00	3.50	71	77.00	2.75
G-1000-0.5-60-1.1	81	82.50	2.00	80	82.00	1.75	78	81.50	1.75
G-1000-0.5-60-1.2	79	83.00	2.75	74	79.50	3.00	76	80.00	3.50
G-1000-0.5-70-1.1	86	87.50	1.00	85	87.50	2.00	81	86.50	2.00
G-1000-0.5-70-1.2	83	86.00	2.00	81	84.00	1.75	82	84.00	2.00

```
> tab <- reshape(df, timevar = "alg", idvar = "inst", direction = "wide")
> library(xtable)
> print(xtable(tab), include.rownames = FALSE, floating = FALSE)

% latex table generated in R 2.9.1 by xtable 1.5-5 package
% Mon Aug 31 20:13:40 2009
\begin{tabular}{lrrrrrrrrr}
  \hline
inst & x.TS1 & iqr.TS1 & min.TS1 & x.TS2 & iqr.TS2 & min.TS2 & x.TS3 & iqr.TS3 & min.TS3 \\
  \hline
G-1000-0.5-30-1.1 &  59 & 64.00 & 2.00 &  37 & 46.00 & 3.25 &  47 & 51.50 & 4.75 \\
  G-1000-0.5-30-1.2 &  60 & 63.50 & 5.50 &  43 & 47.50 & 7.00 &  49 & 59.50 & 6.00 \\
  G-1000-0.5-40-1.1 &  71 & 75.50 & 2.75 &  66 & 71.50 & 2.75 &  68 & 74.50 & 7.50 \\
  G-1000-0.5-40-1.2 &  68 & 72.50 & 5.25 &  60 & 67.50 & 3.00 &  66 & 71.00 & 4.00 \\
  G-1000-0.5-50-1.1 &  78 & 80.50 & 4.50 &  72 & 76.50 & 1.75 &  76 & 79.00 & 2.75 \\
  G-1000-0.5-50-1.2 &  76 & 81.00 & 2.00 &  71 & 76.00 & 3.50 &  71 & 77.00 & 2.75 \\
  G-1000-0.5-60-1.1 &  81 & 82.50 & 2.00 &  80 & 82.00 & 1.75 &  78 & 81.50 & 1.75 \\
  G-1000-0.5-60-1.2 &  79 & 83.00 & 2.75 &  74 & 79.50 & 3.00 &  76 & 80.00 & 3.50 \\
  G-1000-0.5-70-1.1 &  86 & 87.50 & 1.00 &  85 & 87.50 & 2.00 &  81 & 86.50 & 2.00 \\
  G-1000-0.5-70-1.2 &  83 & 86.00 & 2.00 &  81 & 84.00 & 1.75 &  82 & 84.00 & 2.00 \\
   \hline
\end{tabular}
```

2.1.2 Bivariate Analysis

We now turn our attention to heurstics with natural termination criterion, leading to performance measures that depend on both time and solution quality.

For three such heuristics we collected one single run on 30 different instances with same or similar characteristics. As we saw, an aggregate analysis calls for some transformation of data. We choose the standard error transofmation in this case, although better transformations could be found.

We use the lattice `xyplot` to produce a scatter plot that gives us a comprehensive visualization of the data. The analysis could be produced also using the standard `plot` and `text` functions.

Figure 2.6 left shows the clouds of points associated with the three algorithms. It is evident that the algorithm RLF has much higher variability than the other two. If we have many mor algorithms to compare, we may wish to repsent data more syntetically by plotting only the median values. Moreover, logarithmic transformations are useful to emphasize differences.
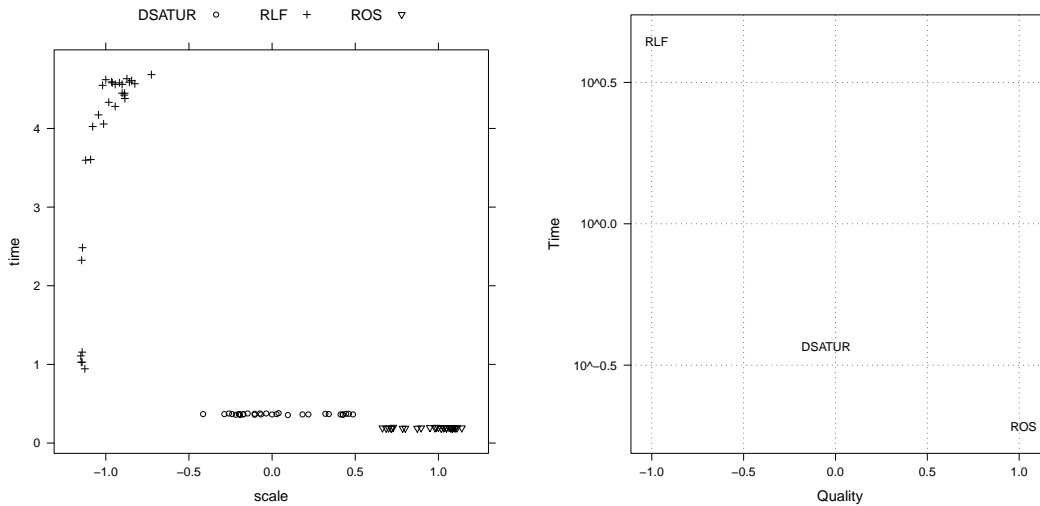
14

Figure 2.6: Bivariate analysis of time and quality. On the right the full data with scale transformation for the quality. On the left the median values and log transformation for the time

The resulting plot is given in Figure 2.6, right. The conclusion from this analysis is that the three algorithms are non domaniated in Pareto sense. Hence the decision on which to use will depend on the application.

```
> G <- read.table("/home/marco/Work08/RTutorial/Data/ch-class-G.txt")
> names(G) <- c("algo", "inst", "sol", "time")
> str(G)

'data.frame':       90 obs. of  4 variables:
 $ algo: Factor w/ 3 levels "DSATUR","RLF",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ inst: Factor w/ 30 levels "G-1000-0.5-20-0.1",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ sol : int  119 117 117 116 119 121 122 122 121 118 ...
 $ time: num  0.188 0.184 0.188 0.192 0.188 ...

> G$scale <- 0
> split(G$scale, G$inst) <- lapply(split(G$sol, G$inst), scale,
      center = TRUE, scale = TRUE)
> print(xyplot(time ~ scale, data = G, groups = algo, auto.key = list(columns = 3)))
> A <- aggregate(G$scale, list(algo = G$algo), median)
> B <- aggregate(G$time, list(algo = G$algo), median)
> G1 <- merge(A, B, by = "algo")
> names(G1) <- c("algo", "sol", "time")
> print(xyplot(time ~ sol, data = G1, groups = algo, scales = list(relation = "free",
      y = list(rot = 0, log = TRUE)), panel = function(x, y, subscripts,
      groups) {
      panel.grid(h = -1, v = -1, lty = 3, col = "grey30")
      ltext(x = x, y = y, label = groups[subscripts], cex = 0.8,
          fontfamily = "Helvetica")
 }, ylab = "Time", xlab = "Quality"))
```

2.2 Example 2: Continuous Optimization

Let's consider a basic linear regression model:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon \qquad i = 1, \ldots, n$$

where X_i is the value of the predictor variable in the ith trial, Y_i is the value of the response variable in the ith trial, β_0 and β_1 are parameters and ϵ_i are mutually independent random errors with $E[\epsilon_i] = 0$ and $\sigma[\epsilon_i] = 0$. We can rewrite in matrix notation:

$$\boldsymbol{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} \qquad \boldsymbol{X} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix} \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \qquad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

The $\mathbf{X}$ is often referred to as the *design matrix*. Hence,

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

We simulate these data in R. The function `rexp(N,l)` generates `N` following an exponential distribution with mean `l`.

```
> set.seed(1)
> N <- 100
> X <- array(dim = c(N, 2))
> X[, 1] <- 1
> X[, 2] <- seq(0.01, 1, by = 0.1)
> Y <- X %*% matrix(c(0, 1)) + matrix(rexp(N, 1))
```

The usual way to estimate the values of the parameters $\boldsymbol{\beta}$ is by means of the least square method that minimizes:

$$\min \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_i)^2$$

The estimators β_0 and β_1 can be found by analytical procedure and in R using the method `lm`.

```
> l <- lm.fit(X, Y)
> plot(X[, 2], Y)
> abline(l)
```

Here, we want instead to use the *least median of squares method*, that is,

$$z(\boldsymbol{\beta}) = \min \left\{ \, \mathrm{median}[(Y_i - \beta_0 - \beta_1 X_i)^2] \right\}$$

The least median method should be more robust against outliers in the model, however the estimation of the parameters β_0 and β_1 requires minimizing a non-differentiable, non-linear, multi-modal function for which an analytical procedure is not known. We can inspect this visually using R methods for 3D plots.

First, we declare a function that implements the function we want to minimize, i.e., $\mathrm{median}[(\boldsymbol{Y} - \boldsymbol{\beta}\boldsymbol{X})^2]$. R allows to declare functions in the following way:
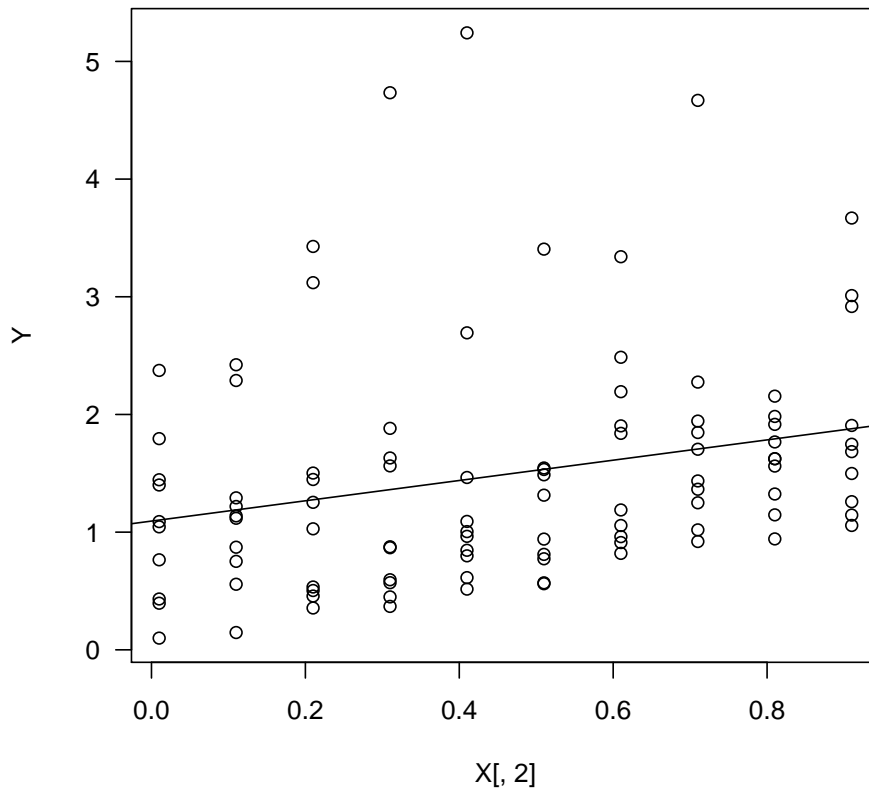
Figure 2.7: Scatter plot and linear regression line superimposed.

```
lmedian <- function(beta, Response, Design) {
  counter <<- counter + 1
  median((Response - Design %*% beta)^2)
}
```

The operator `<-` is used for assignments. Assignments within functions are local. The operator `<<-` is a global assignment, the value of counter will remain modified also outside the function.

Then, we use the method `wireframe` from the package `lattice` to produce a 3D plot. In order to do this we must first determine plot points and then evaluate the function on these points

```
gr <- expand.grid(beta.0=seq(-1,1,0.08),
                  beta.1=seq(-1,5,0.1))
gr$z <- apply(gr,1,function(x) lmedian(x,Y,X))
trellis.par.set("axis.line",list(col=NA,lty=1,lwd=1))
print(
      wireframe(z ~ beta.0 * beta.1, data = gr,
                scales = list(arrows = FALSE),
```
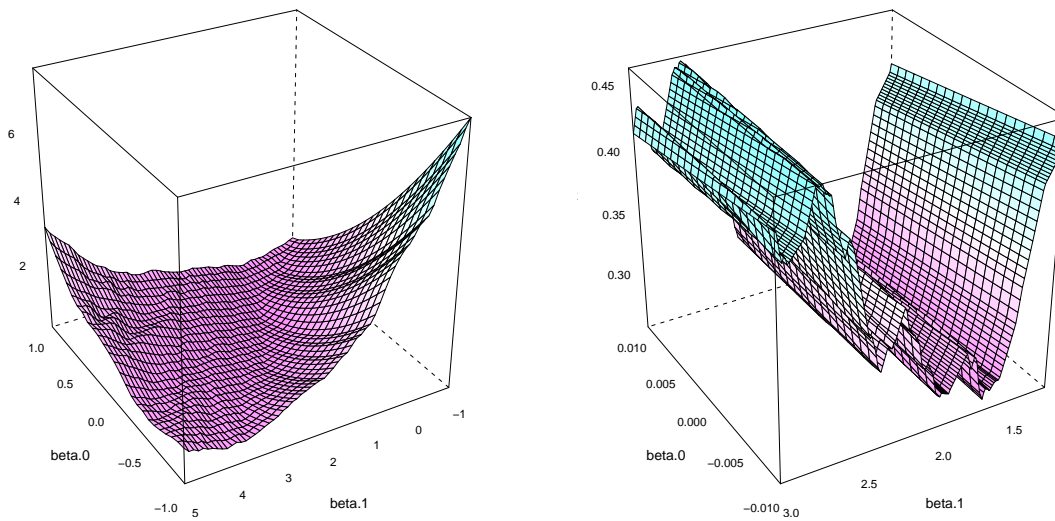
Figure 2.8: Surface of the function to minimize on the two parameters `beta.0` and `beta.1`.

```
            drape = TRUE, colorkey = FALSE,
            aspect=c(1,1),
            screen = list(z = 120, x = -60),zoom=1)
    )
```

The outcome is shown in Figure 2.8, left. At first sight this might seem a well-shaped convex function. However a closer look, shown in Figure 2.8, right, unveils its real nature:

```
gr <- expand.grid(beta.0=seq(-0.01,0.01,0.001),
                  beta.1=seq(1.2,3,0.01))

gr$z <- apply(gr[,1:2],1,function(x) lmedian(x,Y,X))
trellis.par.set("axis.line",list(col=NA,lty=1,lwd=1))
print(
      wireframe(z ~ beta.0 * beta.1, data = gr,
                scales = list(arrows = FALSE),
                drape = TRUE, colorkey = FALSE,
                aspect=c(1,1),
                screen = list(z = 120, x = -60),zoom=1)
    )
```

The presence of local optima becomes evident and together with it the difficulty of solving this optimization problem.

We must therefore resort to numerical methods and heuristics. The R method `optim` offers an implementation of the Nelder-Mead [9] method and Simulated Annealing for continuous optimization. The Nelder-Mead is the default in the function `optim` provided by R. It requires a starting value for the parameters, the function to optimize and its parameters.

```
> beta.init <- matrix(c(0,0))
> O <- optim(c(beta=beta.init), lmedian, Response=Y, Design=X, control=list(trace=6))
```

```
Nelder-Mead direct search function minimizer
function value for initial parameters = 1.730979
  Scaled convergence tolerance is 2.57936e-08
Stepsize computed as 0.100000
BUILD               3 1.730979 1.477877
EXTENSION           5 1.599763 1.184003
EXTENSION           7 1.477877 0.856270
EXTENSION           9 1.184003 0.381746
REFLECTION         11 0.856270 0.306620
.....
REFLECTION        115 0.247260 0.247260
HI-REDUCTION      117 0.247260 0.247260
Exiting from Nelder Mead minimizer
    119 function evaluations used
> O
$par
  beta1   beta2
0.85100 0.45652

$value
[1] 0.24726

$counts
function gradient
     119       NA

$convergence
[1] 0

$message
NULL

> plot(X[,2],Y)
> lines( X[,2], O$par[1]+O$par[2]*X[,2], lty=2, col="blue")
```

Repeating the optimization from different starting points we obtain a value of 0.24581 after 107 function evaluations starting from $\boldsymbol{\beta}_0 = [0,2]^T$ and a value of 0.39371 after 81 function evaluations starting from $\boldsymbol{\beta}_0 = [0,3]^T$.

Optimization methods and algorithms In the previous section we saw that the initial solution has a strong impact on the final result of the Nelder-Mead algorithm. In this section we want to study the effect of random restart on this algorithm. Each descent starts from a solution randomly chosen in the square $[-2,3] \times [0,100]$. We will compare the algorithm without restart and two versions with 100 random restarts. In the first version start points are chosen uniformly at random. In the second version a quasi Monte Carlo method is used. Quasi-Monte Carlo methods are based on low-discrepancy sequences. We use the algorithm and the Fortran implementation by Niederreiter [5]. The difference between the two sampling methods is illustrated in Figure 2.10.

```
> no.restart <- function(seed = 1) {
    set.seed(seed)
    pc <- matrix(c(runif(1, -2, 3), runif(1, 0, 100)))
    O <- optim(c(beta = pc), lmedian, Response = Y, Design = X,
        control = list(trace = 0))
```
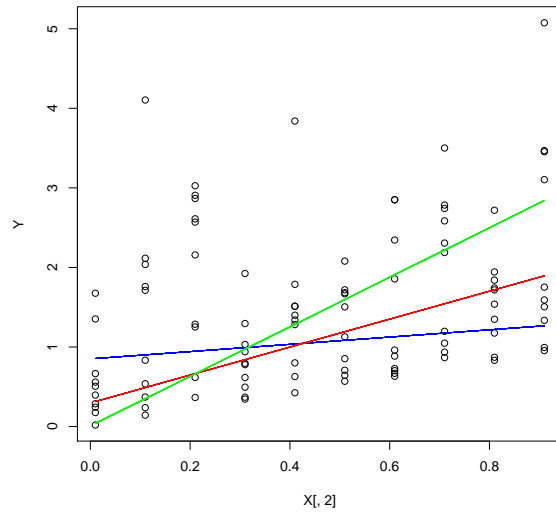
Figure 2.9: Least median of squares regression. Colors refer to different starting points: blue line refers to $\boldsymbol{\beta}_0 = [0,0]^T$ with $z(\boldsymbol{\beta}) = 0.24726$, red to $\boldsymbol{\beta}_0 = [0,2]^T$ with $z(\boldsymbol{\beta}) = 0.24581$ and green to $\boldsymbol{\beta}_0 = [0,3]^T$ with $z(\boldsymbol{\beta}) = 0.39371$. Finally the black line is the supposed optimum found by one of the random restart strategies illustrated in the next section.
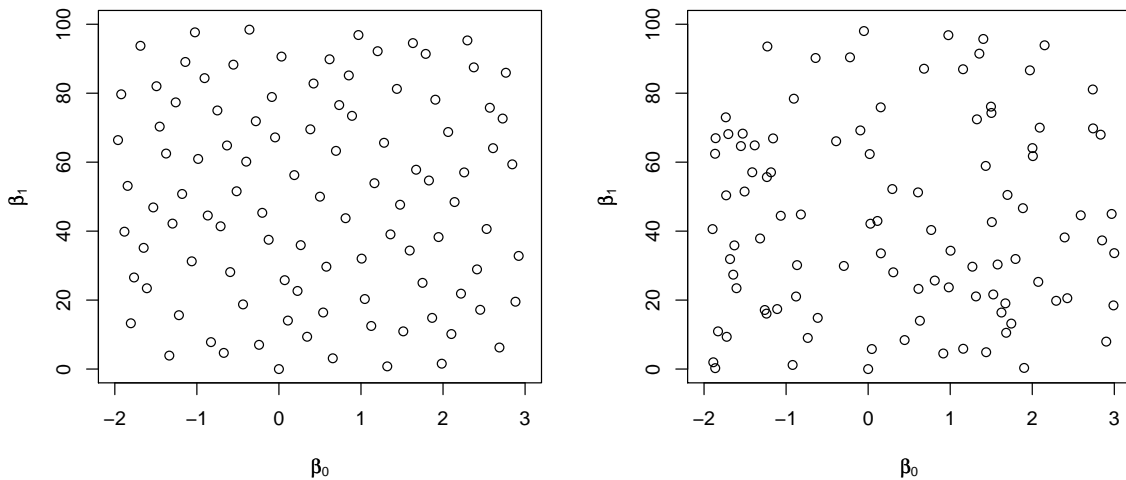

Figure 2.10: Comparison of a sample of 100 points in the square $[-2,3] \times [0,100]$ by quasi-Monte Carlo method (left) and uniform distribution (right). (For the use of mathematical symbols in R plots see ?plotmath.)

```
      return(O$value)
 }
> restart.uniform <- function(N, seed = 1) {
     set.seed(seed)
     values <- array(dim = c(N))
     for (i in 1:N) {
         pc <- matrix(c(runif(1, -2, 3), runif(1, 0, 100)))
         O <- optim(c(beta = pc), lmedian, Response = Y, Design = X,
             control = list(trace = 0))
         values[i] <- O$value
     }
     return(min(values))
 }
> restart.qmc <- function(N, seed = 1) {
     mylib <- file.path(".", "LowDiscrepancy", paste("niederreiter_prb",
         .Platform$dynlib.ext, sep = ""))
     dyn.load(mylib)
     values <- array(dim = c(N))
     s <- seed
     f <- vector(mode = "double", length = 2)
     for (i in 1:N) {
         qmc <- .Fortran("generate", as.integer(2), as.integer(2),
             s = as.integer(s), f = as.double(f))
         s <- qmc$s
         pc <- matrix(c(-2 + qmc$f[1] * (3 - (-2)), 0 + qmc$f[2] *
             (100 - 0)))
         O <- optim(c(beta = pc), lmedian, Response = Y, Design = X,
             control = list(trace = 0))
         values[i] <- O$value
     }
     return(min(values))
 }
```

2.2.1   Comparison on one instance

Univariate analysis   We compare these three algorithms on the basis of the solution cost.
We first collect the data:

```
D <- data.frame(no.restart=sapply(1:30,function(x) no.restart(x)),
               uniform.restart=sapply(1:30,function(x) uniform.restart(10,x)),
               qmc.restart=sapply(1:30,function(x) qmc.restart(10,x)))
> D
   no.restart uniform.restart qmc.restart
1     1.7857          1.7857      1.7852
2     1.7857          1.7855      1.7852
3     1.8247          1.7852      1.7852
.....
29    1.7846          1.7846      1.7852
30    1.8069          1.7857      1.7852
> str(D)
'data.frame':       30 obs. of  3 variables:
```
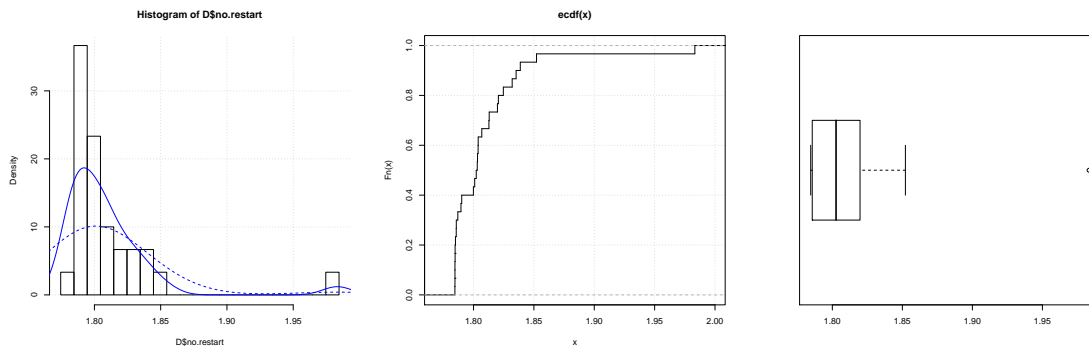
Figure 2.11: Three different ways to look at the distribution of sampled data.

```
$ no.restart      : num   1.79 1.79 1.82 1.98 1.80 ...
$ uniform.restart: num   1.79 1.79 1.79 1.78 1.79 ...
$ qmc.restart     : num   1.79 1.79 1.79 1.79 1.79 ...
```

It is good practice to inspect the data and to check that there are no strange results. We can do this drawing some plots. The following lines show how to produce histograms, empirical cumulative distribution functions and boxplots (see Figure 2.1 and Figure 2.11).

```
> hist(D$no.restart,
        breaks=seq(-0.01+min(D$no.restart),max(D$no.restart+0.01),0.01),
        probability=TRUE,panel.first=grid())
> lines(density(D$no.restart),col="blue")
> lines(density(D$no.restart,adjust=3),lty=2,col="blue")
> plot.ecdf(D$no.restart,panel.first=grid(),do.points=FALSE,verticals=TRUE)
> boxplot(D$no.restart,horizontal=TRUE,names=c("no.restart"),las=1)
```

Repeating the inspection for the other two algorithms we see that the `qmc.restart` procedure returns always the same value. Indeed the algorithm of Niederreiter [5] is deterministic if the same number of samples is required.

Boxplots can be used for the comparison of the three algorithms. The result of the following line is reported in Figure 2.12.

```
> boxplot(D)
```

It clearly arises that `qmc.restart` and `uniform.restart` outperform `no.restart` while in the comparison between `qmc.restart` and `uniform.restart` it is more difficult to distinguish a winner. It is also relevant to have a closer insight at some statistics of the numerical data:

```
> summary(D)
  no.restart    uniform.restart   qmc.restart
Min.   :1.78    Min.    :1.78     Min.    :1.79
1st Qu.:1.79    1st Qu.:1.78      1st Qu.:1.79
Median :1.80    Median :1.78      Median :1.79
Mean   :1.81    Mean    :1.79     Mean    :1.79
3rd Qu.:1.82    3rd Qu.:1.79      3rd Qu.:1.79
Max.   :1.98    Max.    :1.79     Max.    :1.79
```

The conclusion is that `qmc.restart` and `uniform.restart` perform about the same on this specific instance.
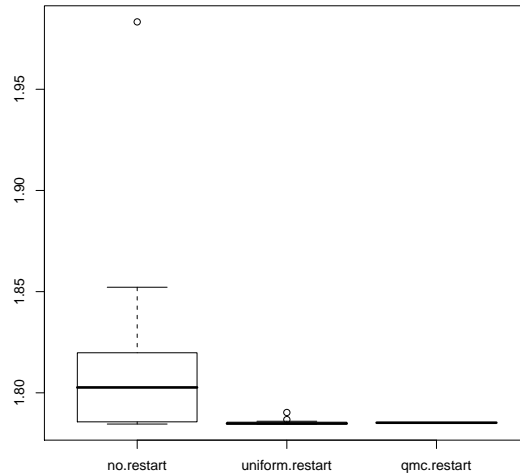
Figure 2.12: Boxplot comparing the three algorithms.

Bivariate analysis In the previous analysis we have not considered the computation time. In the following we take that also into account.

```
D2 <- data.frame()
for ( i in 1:30 )
  {
    T <- system.time(S <- no.restart(i))
    D2 <- rbind(D2,data.frame(algorithm="no.restart",trial=i,quality=S,time=T[[1]]))
    T <- system.time(S <- uniform.restart(10,i))
    D2 <- rbind(D2,data.frame(algorithm="uniform.restart",trial=i,quality=S,time=T[[1]]))
    T <- system.time(S <- qmc.restart(10,i))
    D2 <- rbind(D2,data.frame(algorithm="qmc.restart",trial=i,quality=S,time=T[[1]]))
  }

> D2
        algorithm trial quality  time
1       no.restart     1  1.7857 0.024
2  uniform.restart     1  1.7857 0.292
3      qmc.restart     1  1.7852 0.328
.....
88      no.restart    30  1.8069 0.028
89 uniform.restart    30  1.7857 0.352
90     qmc.restart    30  1.7852 0.312
> str(D2)
'data.frame':       90 obs. of  4 variables:
 $ algorithm: Factor w/ 3 levels "no.restart","uniform.restart",..: 1 2 3 1 2 3 1 2 3 1 ...
 $ trial    : int  1 1 1 2 2 2 3 3 3 4 ...
 $ quality  : num  1.79 1.79 1.79 1.79 1.79 ...
 $ time     : num  0.024 0.292 0.328 0.028 0.344 ...
```

This time we collected the data in the data frame structure in a long format, contrary to D that was in wide format. It is however easy to go from one form to the other by means of
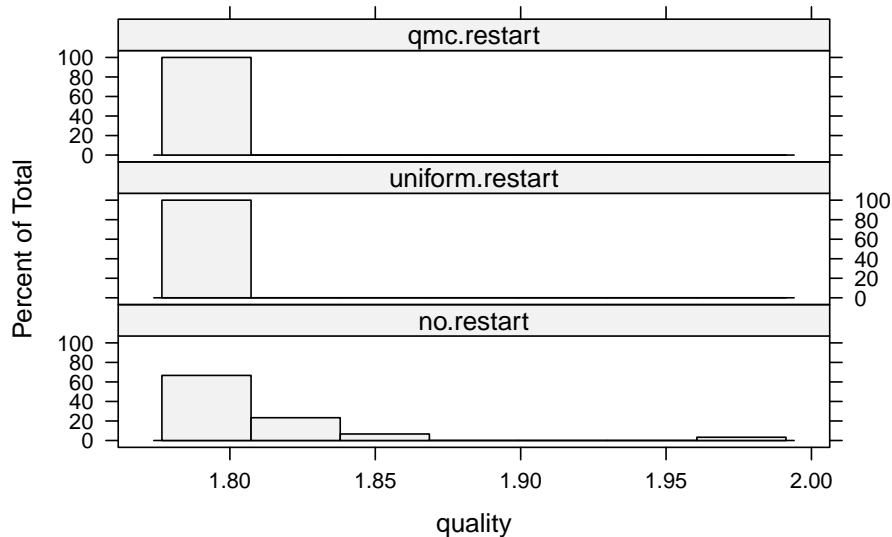
23

Figure 2.13: A conditional histogram. The different panels represent differnt algorithms.

the functions `stack`, `unstack` and `reshape`, for example,

```
> unstack(D2[,c(3,1)])

   no.restart uniform.restart qmc.restart
1      1.7857          1.7857      1.7852
2      1.7857          1.7855      1.7852
.....
29     1.7846          1.7846      1.7852
30     1.8069          1.7857      1.7852
```

The second format is however more convenient for multivariate analysis, in which the responses for different variables are reported in different columns. In our case the two variables are time and quality. Moreover, this format allows us to use the methods from the package `lattice` which is specific for multivariate data visualization. For example,

```
> library(lattice)
> print(histogram(~quality | factor(algorithm), data = D2, layout = c(1,
    3)))
```

produces the output in Figure 2.13.

Back to our comparison, we can plot the results on a time-quality plot. We may do this by plotting all the data or by summarizing them by means of the median. See Figure 2.14.

```
> print(xyplot(quality ~ time, groups = algorithm, data = D2))


> A <- aggregate(D2$quality, list(algorithm = D2$algorithm), median)
> B <- aggregate(D2$time, list(algorithm = D2$algorithm), median)
> D2s <- merge(A, B, by = "algorithm")
> names(D2s) <- c("algorithm", "quality", "time")
> D2s
```
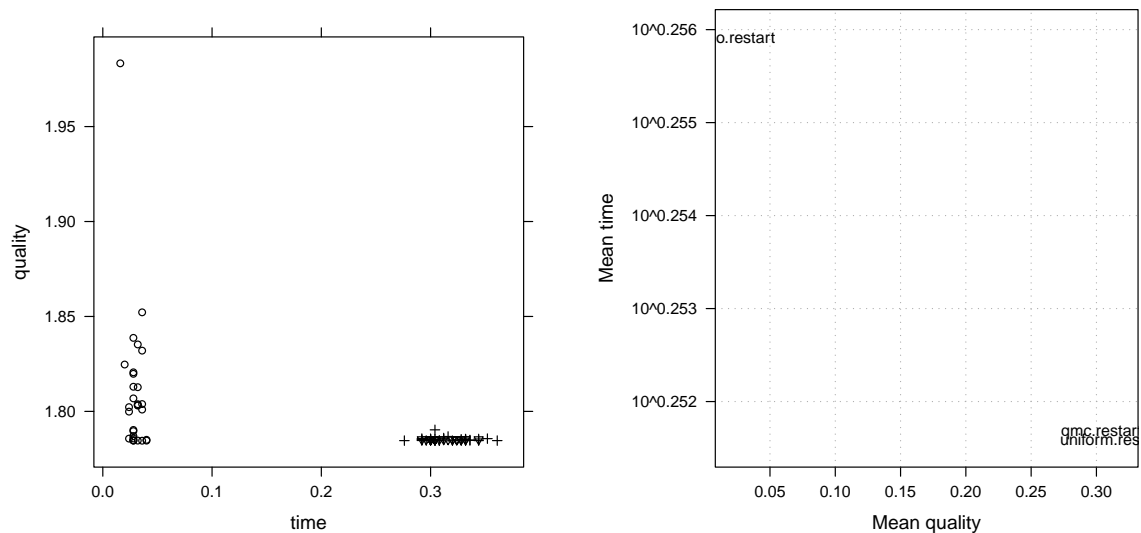
Figure 2.14: Time-quality plot of the results of 30 runs of the three algorithms. On the left, all data are plotted and algorithms differ by the sign of the points. On the right, the median results are used as coordinate for the algorithm's label.

```
        algorithm quality  time
1       no.restart    1.80 0.028
2      qmc.restart    1.79 0.304
3 uniform.restart     1.78 0.312

> print(xyplot(quality ~ time, data = D2s, groups = algorithm,
      scales = list(relation = "free", y = list(rot = 0, log = FALSE)),
      panel = function(x, y, subscripts, groups) {
          panel.grid(h = -1, v = -1, lty = 3)
          ltext(x = x, y = y, label = groups[subscripts], cex = 0.8,
              fontfamily = "Helvetica")
      }, ylab = "Mean time", xlab = "Mean quality"))
```

From Figure 2.14, right, we may conclude that since the quality of the solutions returned is slightly better for the `uniform.restart` and the computation time is about the same, `uniform.restart` seems a better algorithm in this experiment.

2.2.2 Comparison on a set of instances

So far we only evaluated our three algorithms on a single instance. This might be misleading if we want to take a conclusion on which algorithm is the best for instances of a certain type because the instance used for comparison might not be a good representative of the whole population. In this section we will focus on the comparison over a set of instances sampled from the distribution of instances of a certain type.

Let's first generate the instances and store them in a list

```
generate.instance <- function(seed)
  {
```

25

```
    set.seed(seed)
    N <- 1000
    X <- array(dim=c(N,2))
    X[,1] <- 1
    X[,2] <- 10*seq(0.01,1,by=0.1)#rnorm(N,5,10)#seq(0.01,1,by=0.1)
    Y <- X %*% matrix(c(0,1)) + matrix(rnorm(N,0,2))
    return(list(X=X,Y=Y))
}


instances <- list()
for (i in 1:30)
  {
    instances[[i]] <- generate.instance(i)
  }
```

and then let's collect the results

```
D3 <- data.frame()
for ( i in 1:30 )
  {
    X <<- instances[[i]]$X
    Y <<- instances[[i]]$Y
    T <- system.time(S <- no.restart(i))
    D3 <- rbind(D3,data.frame(algorithm="no.restart",instance=i,quality=S,time=T[[1]]))
    T <- system.time(S <- uniform.restart(10,i))
    D3 <- rbind(D3,data.frame(algorithm="uniform.restart",instance=i,quality=S,time=T[[1]]))
    T <- system.time(S <- qmc.restart(10,i))
    D3 <- rbind(D3,data.frame(algorithm="qmc.restart",instance=i,quality=S,time=T[[1]]))
  }
```

This time we present the analysis of the two variates time and quality using the conditional plots of `lattice`. We first need to reshape the data as follows

```
> str(D3)

'data.frame':        90 obs. of  4 variables:
 $ algorithm: Factor w/ 3 levels "no.restart","uniform.restart",..: 1 2 3 1 2 3 1 2 3 1 ...
 $ instance : int  1 1 1 2 2 2 3 3 3 4 ...
 $ quality  : num  1.79 1.79 1.79 1.9 1.83 ...
 $ time     : num  0.032 0.312 0.332 0.044 0.364 ...

> D3r <- reshape(D3, idvar = "id", timevar = "response", varying = list(c("time",
     "quality")), direction = "long", times = c("time", "quality"),
     v.names = "values")
> str(D3r)

'data.frame':        180 obs. of  5 variables:
 $ algorithm: Factor w/ 3 levels "no.restart","uniform.restart",..: 1 2 3 1 2 3 1 2 3 1 ...
 $ instance : int  1 1 1 2 2 2 3 3 3 4 ...
 $ response : chr  "time" "time" "time" "time" ...
 $ values   : num  0.032 0.312 0.332 0.044 0.364 ...
 $ id       : int  1 2 3 4 5 6 7 8 9 10 ...
 - attr(*, "reshapeLong")=List of 4
  ..$ varying:List of 1
  .. ..$ : chr  "time" "quality"
  ..$ v.names: chr "values"
```
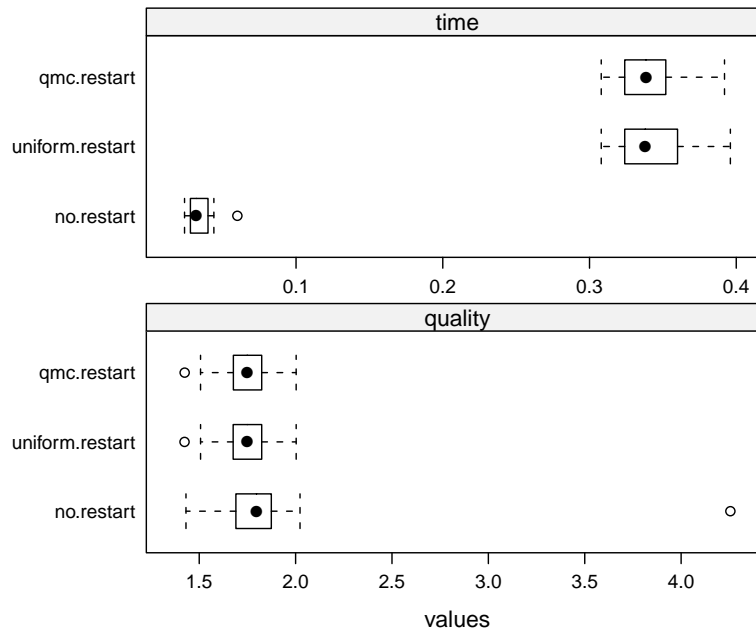
Figure 2.15: Boxplots of time and quality responses of the three algorithms over 30 instances of the least median of squares regression problem. Times are expressed in seconds and refer to a Intel(R) Core(TM)2 CPU at 1.86GHz.

```
..$ idvar  : chr "id"
..$ timevar: chr "response"
```

and then produce the plot of Figure 2.15 with the command

```
> print(bwplot(algorithm ~ values | response, data = D3r, layout = c(1,
      2), scales = "free"))
```

Looking at the quality response we see that apparently there are no significant differences. This contrasts with the conclusion of Figure 2.12 and should make us suspicious. Indeed, if no transformation of data is applied the different scale of the instances is likely to hide differences among the performance of the algorithms. We redo the analysis by using rank transformation of the results within each instance (hence results are mapped in the interval $[1, 3]$ because we have three algorithms and one run per instance). The result is illustrated in Figure 2.16.

```
> D3$rank <- unsplit(tapply(D3$quality, D3$instance, rank, ties.method = "average"),
      D3$instance)
> F <- aggregate(D3$rank, list(D3$algorithm), mean)
> l <- F[order(F$x), 1]
> print(bwplot(factor(algorithm, levels = l) ~ rank, data = D3))
```

The conclusion we should draw from Figure 2.16 is opposite to what we had before. When we consider a set of instances the algorithm qmc.restart shows best performance.
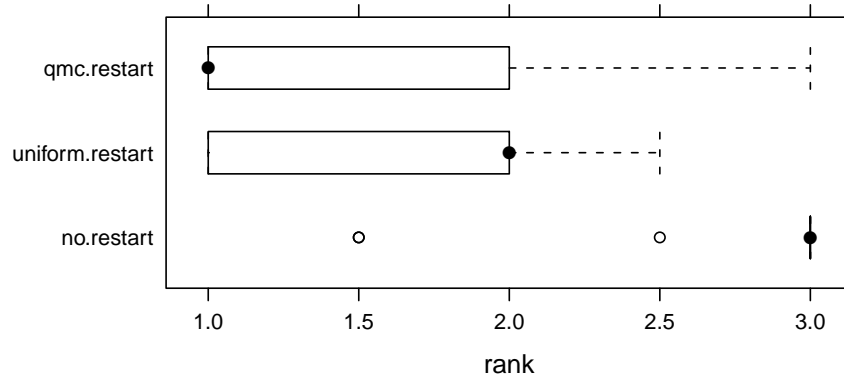
27

Figure 2.16: The comparison on a set of instances after rank transformation.

It might be also worth reporting a table with numerical results. For LaTeX this can be easily achieved with the package `xtable`:

```
> D3t <- unstack(D3[, c(3, 1)])
> library(xtable, lib.loc = "/home/marco/.R/library")
> xtable(D3t[1:5, ])

% latex table generated in R 2.9.1 by xtable 1.5-5 package
% Mon Aug 31 20:13:47 2009
\begin{table}[ht]
\begin{center}
\begin{tabular}{rrrr}
  \hline
 & no.restart & uniform.restart & qmc.restart \\
  \hline
1 & 1.79 & 1.79 & 1.79 \\
  2 & 1.90 & 1.83 & 1.83 \\
  3 & 1.81 & 1.78 & 1.78 \\
  4 & 1.67 & 1.67 & 1.67 \\
  5 & 4.26 & 1.72 & 1.72 \\
   \hline
\end{tabular}
\end{center}
\end{table}
```

	no.restart	uniform.restart	qmc.restart
1	1.79	1.79	1.79
2	1.90	1.83	1.83
3	1.81	1.78	1.78
4	1.67	1.67	1.67
5	4.26	1.72	1.72

Chapter 3

The Racing Method

The racing method is an automatic sequential testing procedure for the comparison, configuration or tuning of algorithms. This chapter describes the use of the method in R. For the theory behind the method, the reader is referred to [1, 2, 4].

3.1 Set up

We assume that the R package `race`[1] developed by M.Birattari [1, 2, 3] is installed (see R documentation on how to install packages.

```
> install.packages("race")
```

The package `race` consists of a library and a *wrapper file*. The library defines the engine of the race, that is, the function `race`. The wrapper file defines all details concerning the specific experiment that must be undertaken. Only this latter file must be edited to specify the details of the experiment.

It is worth emphasizing that the function `race` only implements the race in an *unreplicated design*!

It is possible to execute the race in a distributed computing environment. In this case the machine where the experiments are launched is identified as the master and the other machines as the slaves. Running the race in a distributed environment requires to have installed `pvm` and `Rpvm` in the master and the slave machines.

The first step in setting up the race is retrieving the wrapper file from the library of the R installation:

```
> file.path(system.file(package = "race"), "examples", "example-wrapper.R")
[1] "/home/marco/lib/R/library/race/examples/example-wrapper.R"
```

The library example implements the tuning of a neural network algorithm using a cross validation methodology that divides data in training data and testing data. It might be confusing for the typical cases of optimization. Hence, alternatively, one can use the examples available at www.imada.sdu.dk/ marco/Teaching/Files/.

---

[1]http://cran.r-project.org/web/packages/race/

The wrapper file contains the following functions that have to be adapted to the specific case:

– `race.init` the initialization function

– `race.wrapper` the interface between `race` and the external optimization program. It is the function called by the library to launch one single run of an algorithm on a single instance.

– `race.info` for reporting purposes

– `race.describe` for reporting purposes.

The two main functions to look at are `race.init` and `race.wrapper`. `race.init` is needed to define all data of the race. In particular the instances and the configurations to test. Both can be either read from an external file or encoded in R:

```
> instances <- scan(file = "u-1000-10-1000.txt", what = as.character(0),
+     skip = 0, quiet = TRUE)
> n <- length(instances)
> candidates <- as.data.frame(rbind(c(label = "300787", path = "300787/src",
+     command = "Driver -tt 30 -ch 2 -ls 3"), c(label = "100884",
+     path = "100884/", command = "dm811e/Forced")))
```

Alternatively, candidates can be generated by a full factorial design by crossing several factors. For example:

```
> candidates <- expand.grid(solver = c("CH", "LS"), alpha = c(0.5,
+     1.5), idle = c(100, 300))
> candidates[1:5, ]
  solver alpha idle
1     CH   0.5  100
2     LS   0.5  100
3     CH   1.5  100
4     LS   1.5  100
5     CH   0.5  300
```

The output of the `race.init` function is a list of data. in particular: `no.tasks` is the maximum number of stages in the race. Clearly, in an unreplicated design, this number corresponds to the number of instances. The number of subtasks should be always left to its default value which is 1. Finally, `smpl` is a vector of randomly shuffled integers `1:n` that serves as a mask for deciding the order of examination of the instances.

```
> return(list(name = class, no.candidates = nrow(candidates), no.tasks = n,
+     no.subtasks = 1, wd = wd, smpl = smpl, instances = instances,
+     candidates = candidates))
```

The `race.wrapper` function strongly depends on the way the external program has been implemented. The function must return one single value which is the result of the run of the algorithm `candidate` on the instance `smpl[task]`. The simplest way is to let the optimization program return one single value and redirect all the rest. For example, with a C program:

```
> command <- paste(data$candidates[candidate, ]$command, " -i ",
+     instance, " -t ", time, " -s ", data$smpl[task], " -o ",
+     paste(candidate, task, 1, sep = "-"), " 2>/dev/null", sep = "")
> s <- system(command, intern = TRUE, ignore.stderr = TRUE)
```

It might be wise in a debugging phase to print out the full launch command, for example, with `cat(command)`. Further, it is advisable, when running long experiments, to write the outcome of the run in a log file as soon as this result is retrieved.

When the wrapper file is ready it is advisable to run some tests. For example:

```
> D <- race.init()
> D
> race.wrapper(1, 1, D)
```

If the tests run fine then everything is ready to be launched. The following is an example of launch command:

```
> O <- race("wrapper-race.R", maxExp = 5000, stat.test = c("friedman"),
+     conf.level = 0.95, first.test = 5, interactive = TRUE, log.file = "race.log",
+     no.slaves = 0)
```

See the race documentation (`?race`) for an explanation of the parameters.

When the race is finished it is possible to plot a profile of what happened by means of the function `plot.race` available from www.imada.sdu.dk/ marco/Teaching/Files/plot.race.R.

```
> source("plot.race.R")
> plot.race(O, "wrapper-file.R")
```

It might be necessary to edit the function for layout adjustments.

3.2   An Example

```
> O <- race(wrapper.file="example-wrapper.R",
+           maxExp=3240, ## multiple of number of candidates
+           stat.test=c("friedman"),
+           conf.level=0.95,
+           first.test=5,
+           interactive=TRUE,
+           #log.file=paste(file,".log",sep=""),
+           no.slaves=0)

Racing methods for the selection of the best
Copyright (C) 2003 Mauro Birattari
This software comes with ABSOLUTELY NO WARRANTY


Race name.......................NM for Least Median of Squares
Number of candidates.....................................162
Number of available tasks.................................45
Max number of experiments..............................3240
Statistical test..................................Friedman test
```

```
Tasks seen before discarding..................................5
Initialization function......................................ok
Parallel Virtual Machine.....................................no

                        Markers:
                          x No test is performed.
                          - The test is performed and
                            some candidates are discarded.
                          = The test is performed but
                            no candidate is discarded.


+-+----------+----------+----------+----------+----------+
| |     Task|     Alive|      Best| Mean best| Exp so far|
+-+----------+----------+----------+----------+----------+
|x|        1|       162|        81| 2.869e-05|       162|
|x|        2|       162|       140| 2.761e-05|       324|
|x|        3|       162|        86| 2.607e-05|       486|
|x|        4|       162|       140| 2.887e-05|       648|
|-|        5|        52|       140| 3.109e-05|       810|
|=|        6|        52|        34| 3.892e-05|       862|
...
|=|       42|        13|        32|  4.76e-05|      1703|
|=|       43|        13|        32| 4.684e-05|      1716|
|=|       44|        13|        32| 4.616e-05|      1729|
|=|       45|        13|        32|  4.55e-05|      1742|
+-+----------+----------+----------+----------+----------+

Selected candidate:         32       mean value:   4.55e-05


Description of the selected candidate:
   initial.method max.reinforce alpha beta gamma                     label
32   quasi-random              1   1.5  0.5   1.5 quasi-random-1-1.5-0.5-1.5
```

The race finished after all instances available (45) have been used without a single winner. At the end 13 configurations were still alive, that is, not yet found significantly different. The race returns however a winner decided on the basis of the median rank of its results.

Figure 3.1 visualizes the process. The grey area represents the aggregate computation time effectively used by the race. This corresponds to 1742 runs of the algorithms with the best of them tested on 45 instances. Given the 162 initial algorithm configurations, experimenting all of them on the 45 instance would have required 3240 runs corresponding to the whole area covered by the graph. Alternatively, with the same computation time it would have been possible to experiment only on 20 instances.
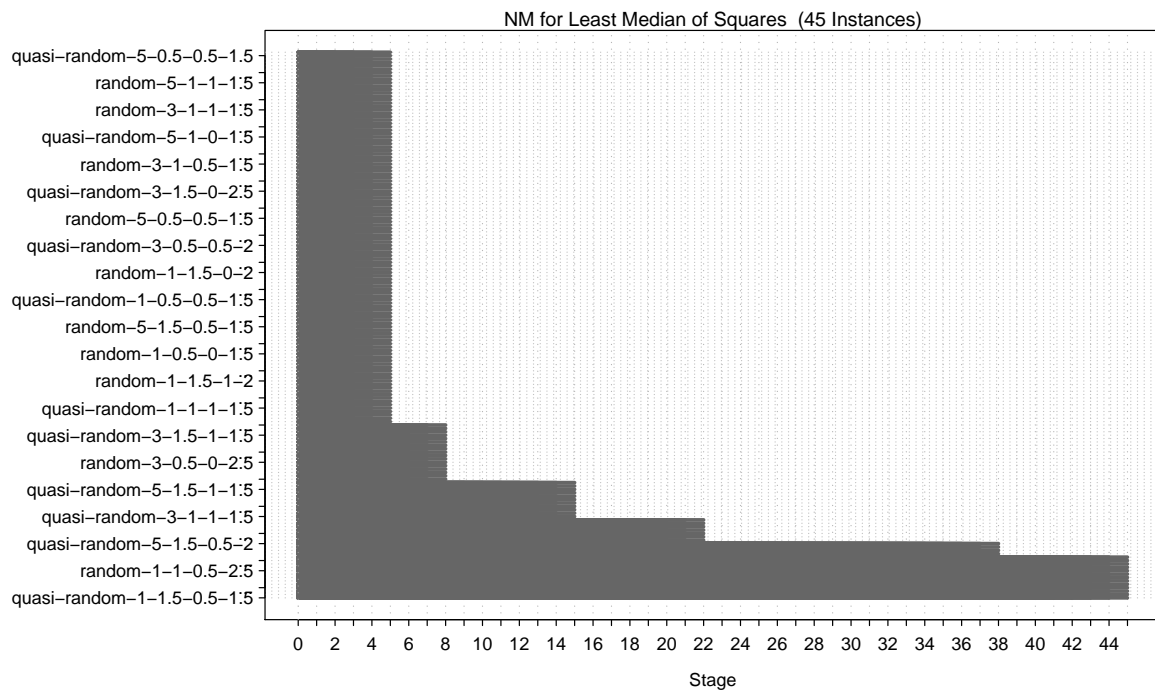
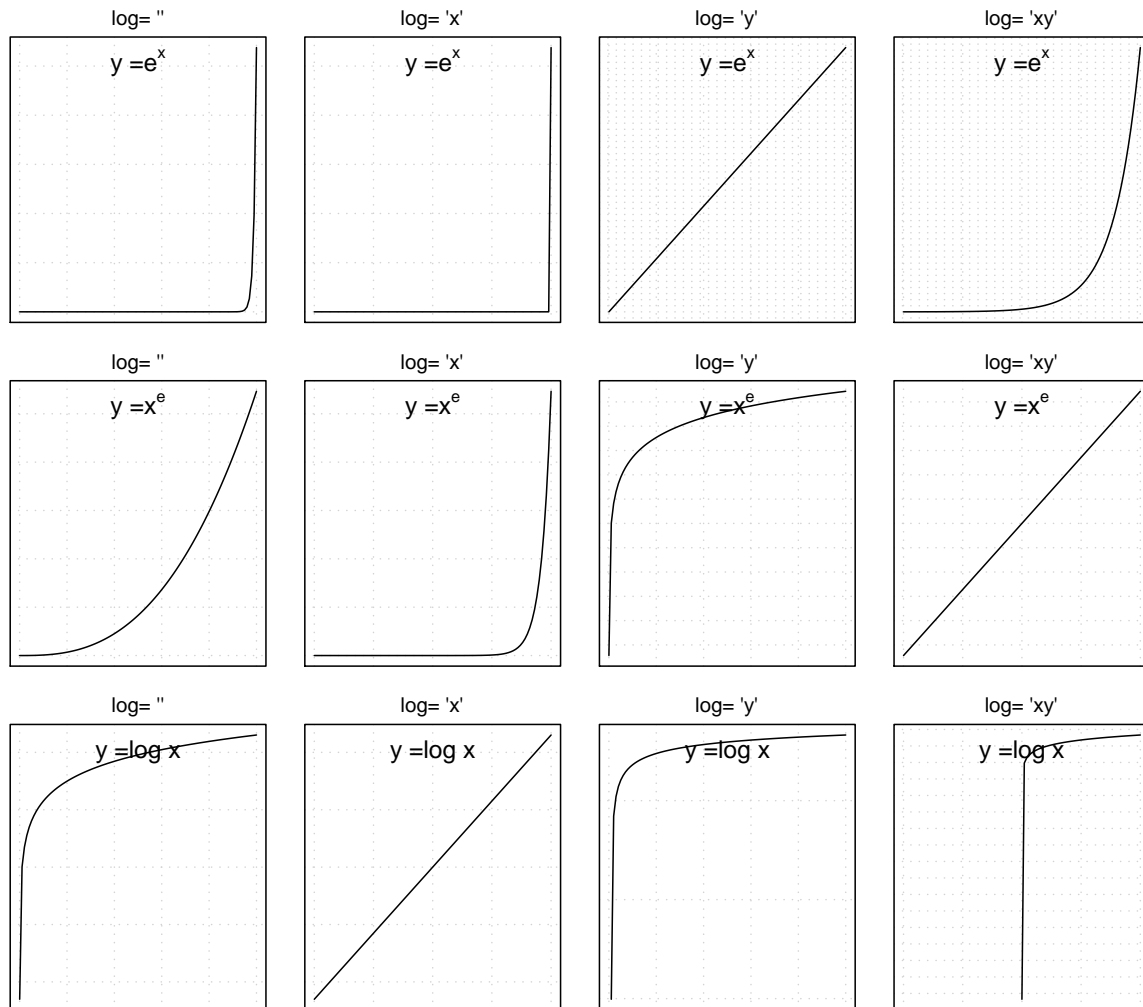Figure 3.1: Graphical view of the race described in the text.

Appendix

Figure 2: Log transformations and linear trends. Three different growth functions are depicted in the three rows, while in the columns different transformations are applied on the scales of the graph

Bibliography

[1] M. Birattari. The race package for R. racing methods for the selection of the best. Technical Report TR/IRIDIA/2003-37, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2003.

[2] M. Birattari. *The Problem of Tuning Metaheuristics, as seen from a machine learning perspective.* PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2004.

[3] Mauro Birattari. *race: Racing methods for the selection of the best*, 2005. R package version 0.1.56.

[4] Mauro Birattari. *Tuning Metaheuristics, A Machine Learning Perspective*, volume 197 of *Studies in Computational Intelligence.* Springer, 2009.

[5] P. Bratley, B. L. Fox, and H. Niederreiter. Algorithm-738 - programs to generate niederreiters low-discrepancy sequences. *ACM Transactions On Mathematical Software*, 20(4):494–495, December 1994.

[6] Sarkar Deepayan. *Lattice Multivariate Data Visualization with R.* Springer, New York, 2007. ISBN 978-0-387-75968-5.

[7] Friedrich Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. pages 575–580, 2002. ISBN 3-7908-1517-9.

[8] Friedrich Leisch. Sweave user manual. 2008.

[9] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965. An Errata has been published in The Computer Journal 1965 8(1):27.