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.

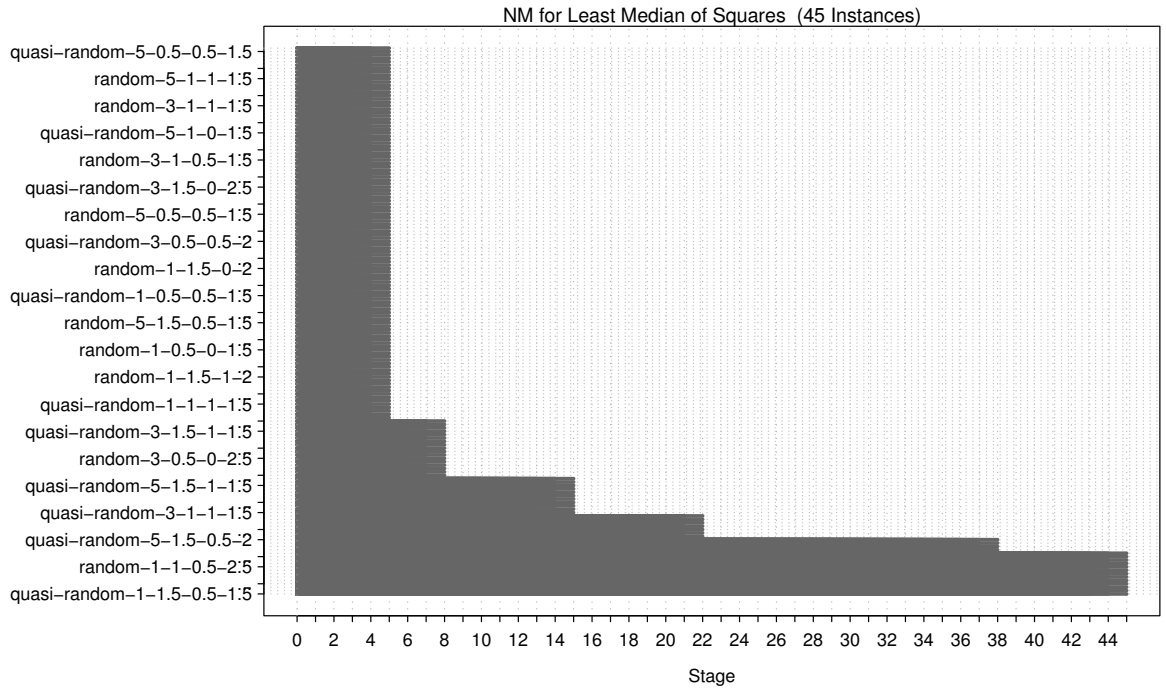

Figure 3.1: Graphical view of the race described in the text.