Experimental Analysis of Optimization Heuristics Using R

Marco Chiarandini

March 9, 2009

1 Setting up the Race

This section describes how to set up an automatic sequential testing procedure for the comparison, configuration or tuning of algorithms. The description assumes 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.

It should be kept well in mind that the function `race` only implements the race in an *unreplicated design*.

It is instead possible to perform 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 best way to proceed in order to set up the race is to retrieve the wrapper file from the race and to copy it into the working directory.

```
> file.path(system.file(package = "race"), "examples", "example-wrapper.R")
```

```
[1] "/examples/example-wrapper.R"
```

The 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 example available at `http://www.imada.sdu.dk/~marco/Teaching/Fall2008/DM812/Lab/wrapper-race.R`.

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` serves 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:

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

```
> 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, `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 smooth 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 `http://www.imada.sdu.dk/~marco/Teaching/Fall2008/DM812/Lab/plot.race.R`.

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

It might be needed to edit the function and change it for layout adjustments.

References

[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.