Using parallel computing in **GA** package

Luca Scrucca
Università degli Studi di Perugia, Italy
luca@stat.unipg.it

August 2013

1 Introduction

By default searches performed using the **GA** package occour sequentially. In some cases, particularly when the evaluation of the fitness function is time consuming, parallelization of the search algorithm may be able to speedup computing time. Starting with version 2.0, the **GA** package provides facilities for implementing parallelization of genetic algorithms.

Parallel computing with **GA** requires the following packages to be installed: **parallel** (available in base R), **doParallel**, **foreach**, and **iterators**.

2 Usage

To use parallel computing with the **GA** package is simple as manipulating the optional argument `parallel` in the `ga()` function call.

The argument `parallel` can be a logical argument specifying if parallel computing should be used (`TRUE`) or not (`FALSE`, default) for evaluating the fitness function. This argument could also be used to specify the number of cores to employ; by default, this is taken from `detectCores()` function in **parallel** package.

Two types of parallel functionality are implemented depending on system OS: on Windows only **snow** type functionality is available, while on POSIX operating systems, such as Unix, GNU/Linux, and Mac OSX, both **snow** and **multicore** (default) functionalities are available. In the latter case a string can be used to specify which parallelization method should be used.

3 Example

Consider the following simple example where we artificially introduced a pause statement to simulate an expensive fitness function.

```
> Rastrigin <- function(x1, x2)
  {
    Sys.sleep(0.1)
    20 + x1^2 + x2^2 - 10*(cos(2*pi*x1) + cos(2*pi*x2))
  }
> system.time(GA1 <- ga(type = "real-valued",
                        fitness =  function(x) -Rastrigin(x[1], x[2]),
                        min = c(-5.12, -5.12), max = c(5.12, 5.12),
                        popSize = 50, maxiter = 100, monitor = FALSE,
                        seed = 12345))
    user  system elapsed
  1.077   0.203 415.489
```

```
> system.time(GA2 <- ga(type = "real-valued",
                       fitness =  function(x) -Rastrigin(x[1], x[2]),
                       min = c(-5.12, -5.12), max = c(5.12, 5.12),
                       popSize = 50, maxiter = 100, monitor = FALSE,
                       seed = 12345, parallel = TRUE))
   user  system elapsed
  6.155   5.230 124.345
```

```
> system.time(GA3 <- ga(type = "real-valued",
                       fitness =  function(x) -Rastrigin(x[1], x[2]),
                       min = c(-5.12, -5.12), max = c(5.12, 5.12),
                       popSize = 50, maxiter = 100, monitor = FALSE,
                       seed = 12345, parallel = 2))
  user  system elapsed
  6.170   5.916 222.454
```

```
> system.time(GA4 <- ga(type = "real-valued",
                       fitness =  function(x) -Rastrigin(x[1], x[2]),
                       min = c(-5.12, -5.12), max = c(5.12, 5.12),
                       popSize = 50, maxiter = 100, monitor = FALSE,
                       seed = 12345, parallel = "snow"))
   user  system elapsed
  5.412   0.418 143.913
```

The following table summarizes the results and show the improvement achieved by using parallelization in GAs:

Num. cores	System time	Gain
1	415.489	1.00
2 (multicore)	222.454	0.54
4 (multicore)	124.345	0.30
4 (snow)	143.913	0.35

The system times reported above refer to a MacBook Pro, Intel Core i5 at 2.3 GHz, with 4 cores and 4 GB RAM, running OSX 10.8.3.