Using the `doRNG` package

doRNG package – Version 1.2.2

Renaud Gaujoux

March 29, 2012

Research reproducibility is an issue of concern, in particular in bioinformatics [3, 7, 4]. Some analyses require multiple independent runs to be performed, or are amenable to a split-and-reduce scheme. For example, some optimisation algorithms are run multiple times from different random starting points, and the result that achieves the least approximation error is selected. The *foreach* package[1] [1] provides a very convenient way to perform parallel computations, with different parallel environments such as MPI or Redis, using a transparent loop-like syntax:

```
# load and register parallel backend for multicore computations
library(doParallel)

## Loading required package: foreach
## Loading required package: iterators
## Loading required package: codetools
## Loading required package: parallel

cl <- makeCluster(2)
registerDoParallel(cl)

# perform 5 tasks in parallel
x <- foreach(i = 1:5) %dopar% {
    i + runif(1)
}
unlist(x)

## [1] 1.471 2.295 3.888 4.229 5.366
```

For each parallel environment a *backend* is implemented as a specialised `%dopar%` operator, which performs the setup and pre/post-processing specifically required by the environment (e.g. export of variable to each worker). The `foreach` function and the `%dopar%` operator handle the generic parameter dispatch when the task are split between worker processes, as well as the reduce step – when the results are returned to the master worker.

When stochastic computations are involved, special random number generators must be used to ensure that the separate computations are indeed statistically independent – unless otherwise wanted – and that the loop is reproducible. Standard `%dopar%` loops are not reproducible:

```
# with standard %dopar%: foreach loops are not reproducible
set.seed(123)
res <- foreach(i=1:5) %dopar% { runif(3) }
set.seed(123)
res2 <- foreach(i=1:5) %dopar% { runif(3) }
identical(res, res2)
```

[1] http://cran.r-project.org/package=foreach

```
## [1] FALSE
```

A random number generator commonly used to achieve reproducibility is the combined multiple-recursive generator from L'Ecuyer [5]. This generator can generate independent random streams, from a 6-length numeric seed. The idea is then to generate a sequence of random stream of the same length as the number of iteration (i.e. tasks) and use a different stream when computing each one of them.

The *doRNG* package[2] [2] provides convenient ways to implement reproducible parallel `foreach` loops, independently of the parallel backend used to perform the computation. We illustrate its use, showing how non-reproducible loops can be made reproducible, even when tasks are not scheduled in the same way in two separate set of runs, e.g. when the workers do not get to compute the same number of tasks or the number of workers is different.

1 The %dorng% operator

The *doRNG* package defines a new operator, `%dorng%`, to be used with foreach loops, instead of the standard %dopar%. Lops that use this operator are *de facto* reproducible.

```
# load the doRNG package
library(doRNG)

## Loading required package: methods


# using %dorng%: loops _are_ reproducible
set.seed(123)
res <- foreach(i=1:5) %dorng% { runif(3) }
set.seed(123)
res2 <- foreach(i=1:5) %dorng% { runif(3) }
identical(res, res2)

## [1] TRUE
```

The actual random seed used for the first loop iteration is stored as attribute `'RNG'` in the result:

```
attr(res, "RNG")

## [1]         407   642048078    81368183 -2093158836   506506973  1421492218 -1906381517
```

The following iterations are seeded recursively calling the `nextRNGStream` function from the *parallel* package[3] [6], hence creating a sequence of seeds for statistically independent RNG streams.

The initial seed can also be passed via an option to the `%dorng%` operator, and be a single numeric (as for `set.seed`), or a 6 or 7-length numeric which is used as the initial seed for L'Ecuyer's RNG or value for `.Random.seed`[4] respectively:

```
# use a single numeric as a seed
res3 <- foreach(i = 1:5, .options.RNG = 123) %dorng% {
    runif(3)
}
identical(res3, res)
```

[2] http://cran.r-project.org/package=doRNG

[3] http://cran.r-project.org/package=parallel

[4] Note that the RNG type is always required to be the `"L'Ecuyer-CMRG"`.

```
## [1] TRUE


# use complete .Random.seed
res4 <- foreach(i = 1:5, .options.RNG = attr(res, "RNG")) %dorng%
    {
        runif(3)
    }
identical(res4, res)

## [1] TRUE


# use a 6-length numeric
s <- foreach(i = 1:5, .options.RNG = 1:6) %dorng% {
    runif(3)
}
attr(s, "RNG")

## [1] 407   1   2   3   4   5   6
```

An important feature of `%dorng%` loops is that their result is independent of the underlying parallel physical settings. Two separate runs seeded with the same value will always produce the same results. Whether they use the same number of worker processes, parallel backend or task scheduling does not influence the final result. This also applies to computations performed sequentially with the the `doSEQ` backend. The following code illustrates this using 2 or 3 workers.

```
res_2workers <- foreach(i = 1:5, .combine = rbind, .options.RNG = 123) %dorng%
    {
        c(pid = Sys.getpid(), val = runif(1))
    }

# stop previous cluster (that uses 2 workers)
stopCluster(cl)

# create cluster with 3 workers
cl <- makeCluster(3)
registerDoParallel(cl)
res_3workers <- foreach(i = 1:5, .combine = rbind, .options.RNG = 123) %dorng%
    {
        c(pid = Sys.getpid(), val = runif(1))
    }

# task schedule is different
pid <- rbind(res1 = res_2workers[, 1], res2 = res_3workers[,
    1])
storage.mode(pid) <- "integer"
pid

##      result.1 result.2 result.3 result.4 result.5
## res1    19358    19367    19358    19358    19358
## res2    19380    19389    19398    19380    19389

# results are identical
identical(res_2workers[, 2], res_3workers[, 2])

## [1] TRUE
```

2 Seamless convertion of %dopar% into reproducibile loops

The *doRNG* package also provides a way to convert `%dopar%` loops into reproducible loops without changing their actual definition. It is useful to quickly ensure the reproducibility of existing code or functions whose definition is not accessible (e.g. from other packages). This is achieved by registering the `doRNG` backend:

```
registerDoRNG(123)
res_dopar <- foreach(i = 1:5) %dopar% {
    runif(3)
}
identical(res_dopar, res)

## [1] TRUE

attr(res_dopar, "RNG")

## [1]         407   642048078    81368183 -2093158836   506506973  1421492218 -1906381517
```

3 Reproducibility of multiple loops

Sequences of multiple loops are reproducible, whether using the `%dorng%` operator or the registered `doRNG` backend:

```
set.seed(456)
s1 <- foreach(i=1:5) %dorng% { runif(3) }
s2 <- foreach(i=1:5) %dorng% { runif(3) }
# the two loops do not use the same streams: different results
identical(s1, s2)

## [1] FALSE


# but the sequence of loops is reproducible as a whole
set.seed(456)
r1 <- foreach(i=1:5) %dorng% { runif(3) }
r2 <- foreach(i=1:5) %dorng% { runif(3) }
identical(r1, s1) && identical(r2, s2)

## [1] TRUE


# one can equivalently register the doRNG backend and use %dopar%
registerDoRNG(456)
r1 <- foreach(i=1:5) %dopar% { runif(3) }
r2 <- foreach(i=1:5) %dopar% { runif(3) }
identical(r1, s1) && identical(r2, s2)

## [1] TRUE
```

Cleanup

```
stopCluster(cl)
```

Session information

```
R version 2.14.2 (2012-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_ZA.UTF-8       LC_NUMERIC=C               LC_TIME=en_ZA.UTF-8
 [4] LC_COLLATE=en_ZA.UTF-8     LC_MONETARY=en_ZA.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=C                 LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_ZA.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] methods   parallel  stats     graphics  grDevices utils     datasets  base

other attached packages:
[1] doRNG_1.2.2      doParallel_1.0.0 foreach_1.3.5    codetools_0.2-8  iterators_1.0.5
[6] knitr_0.4

loaded via a namespace (and not attached):
 [1] compiler_2.14.2 digest_0.5.2    evaluate_0.4.1  formatR_0.3-4   highlight_0.3.1
 [6] parser_0.0-14   plyr_1.7.1      Rcpp_0.9.10     stringr_0.6     tools_2.14.2
```

References

[1] Revolution Analytics. *foreach: Foreach looping construct for R*, 2012. R package version 1.3.5.

[2] Renaud Gaujoux. *doRNG: Generic Reproducible Parallel Backend for foreach Loops*, 2010. R package version 1.2.2.

[3] Torsten Hothorn and Friedrich Leisch. Case studies in reproducibility. *Briefings in bioinformatics*, January 2011.

[4] John P A Ioannidis, David B Allison, Catherine A Ball, Issa Coulibaly, Xiangqin Cui, Aedín C Culhane, Mario Falchi, Cesare Furlanello, Laurence Game, Giuseppe Jurman, Jon Mangion, Tapan Mehta, Michael Nitzberg, Grier P Page, Enrico Petretto, and Vera Van Noort. The reproducibility of lists of differentially expressed genes in microarray studies. *Nature Genetics*, 41(2):149–155, 2008.

[5] Pierre L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47(1), 1999.

[6] R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.

[7] Victoria C Stodden. The Digitization of Science: Reproducibility and Interdisciplinary Knowledge Transfer, 2011.