Using Julia with ResistanceGA

Bill Peterman

7/25/2020

Description

One of the greatest limitations of using `ResistancaGA` for landscape resistance optimization is computation time. This time increases as the number of samples in space increases and/or the dimensions of the raster surface increase. Recently, CIRCUITSCAPE was re-written in the Julia computing language. This implementation can be 4–10x faster than the original CIRCUITSCAPE, which is implemented in Python. This short document details the steps needed to use CIRCUITSCAPE in Julia when optimizing with `ResistanceGA`

Download and Installation

- First, you need to download Julia

- Following installation of Julia, follow directions for installing the Circuitscape package. In short, open the Julia command-line console, then type/paste the following:

```
using Pkg
Pkg.add("Circuitscape")
Pkg.test("Circuitscape")
```

- Next, install the latest version of `ResistanceGA`

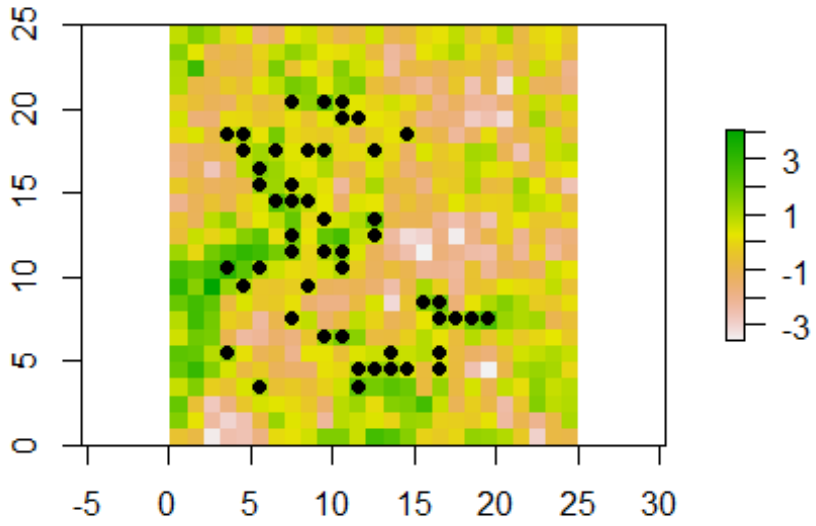
```
devtools::install_github("wpeterman/ResistanceGA")
```

Optimizing with Julia

For the most part, optimizing with Julia is very similar to using `Circuitscape` of `gdistance`. The development package has some example data sets that we can use for demonstration purposes.

```
library(ResistanceGA)

## Loading required package: raster

## Loading required package: sp

## Warning: replacing previous import 'lme4::getData' by 'raster::getData'
when
## loading 'ResistanceGA'

## Registered S3 method overwritten by 'spatstat':
##   method     from
##   print.boxx cli
```

```
## Sample locations
sp.dat <- sample_pops$sample_cont

## Continuous landscape surface
cont.rast <- raster_orig$cont_orig

## Genetic distance measured between sample locations (chord distance)
gen.dist <- Dc_list$Dc_cont

plot(cont.rast)
plot(sp.dat, add = T, pch = 19)
```



Next prepare the GA and Julia inputs. You need to specify the full path to the Julia software on your computer, and connect Julia to R. The first time you run Julia within an R session, it will be quite slow as a lot is being prepared in the background.

```
## Specify path the to `bin` directory
## This path may vary depending upon Julia version or operating system
JULIA_HOME <- "C:/Users/peterman.73/AppData/Local/Programs/Julia 1.5.0/bin/"
JuliaCall::julia_setup(JULIA_HOME)
```

If you get this error:

```
Error: Error happens in Julia.
InitError: could not load library "libdSFMT"
The specified module could not be found.
```

re-run the `jl.prep` function. This should only occur once, if at all.

Next run the preparation functions.

```
GA.inputs <- GA.prep(ASCII.dir = cont.rast,
                     Results.dir = "C:/Rga_examples/",
                     parallel = 4)
jl.inputs <- jl.prep(n.Pops = length(sp.dat),
                     response = lower(gen.dist),
                     CS_Point.File = sp.dat,
                     JULIA_HOME = JULIA_HOME)
```

If everything is working correctly, you should see the following message in the console:

```
[1] "Test: Run Circuitscape from Julia"

Test Passed
```

The default method for solving the resistance surface is to use `cholmod = TRUE`. As described on the Circuitscape Julia page, this is where the speed gains are achieved. However, this approach can be VERY memory intensive, so may not be practical if dealing with large rasters. As an alternative when using large rasters, you may want to switch to the *experimental* single precision method. This requires setting `cholmod = FALSE` and `precision = TRUE`.

Finally, optimize your resistance surface(s).

```
jl.optim <- SS_optim(jl.inputs = jl.inputs,
                     GA.inputs = GA.inputs)
```

The use of `SS_optim`, `MS_optim`, and `all_comb` are the same as before. You just need to provide the `jl.inputs` object created from the `jl.prep` function to optimize with Julia.

Good luck, and please let me know if you encounter issues!