In my daily work I often have to transform a long table to a wide matrix so accommodate some function. At some stage in my life I came across the reshape2 package, and I have been with that philosophy ever since – I find it makes data wrangling easy and straight forward. I particularly like the tidyverse philosophy where data should be in a long table, where one row is an observation, and a column a parameter. It just makes sense.

However, I quite often have to transform the data into another format, a wide matrix especially for functions of the vegan package, and one day I wondering how to do that in the fastest way.

The code to create the test sets and benchmark the functions is in section ‘*Settings and script’* at the end of this document.

I created several data sets that mimic the data I usually work with in terms of size and values. The data sets have 2 to 10 groups, where each group can have up to 50000, 100000, 150000, or 200000 samples. The methods xtabs() from base R, dcast() from data.table, dMcast() from Matrix.utils, and spread() from tidyr were benchmarked using microbenchmark() from the package microbenchmark. Each method was evaluated 10 times on the same data set, which was repeated for 10 randomly generated data sets.

After the 10 x 10 repetitions of casting from long to wide it is clear the spread() is the worst. This is clear when we focus on the size (figure 1).

Figure 1. Runtime for 100 repetitions of data sets of different size and complexity.

And the complexity (figure 2).

Figure 2. Runtime for 100 repetitions of data sets of different complexity and size.

**Close up on the top three methods**

Casting from a long table to a wide matrix is clearly slowest with spread(), where as the remaining look somewhat similar. A direct comparison of the methods show a similarity in their performance, with dMcast() from the package Matrix.utils being better — especially with the large and more complex tables (figure 3).

Figure 3. Direct comparison of set size.

I am aware, that it might be to much to assume linearity, between the computation times at different set sizes, but I do believe it captures the point – dMcast() and dcast() are similar, with advantage to dMcast() for large data sets with large number of groups. It does, however, look like dcast() scales better with the complexity (figure 4).

Figure 4. Direct comparison of number groups.

**Settings and script**

**Session info**

## ─ Session info ──────────────────────────────────────────────────────────

## setting value

## version R version 3.5.2 (2018-12-20)

## os Ubuntu 18.04.1 LTS

## system x86\_64, linux-gnu

## ui X11

## language en\_GB:en\_US

## collate en\_DE.UTF-8

## ctype en\_DE.UTF-8

## tz Europe/Berlin

## date 2019-02-03

##

## ─ Packages ──────────────────────────────────────────────────────────────

## package \* version date lib source

## assertthat 0.2.0 2017-04-11 [1] CRAN (R 3.5.2)

## bindr 0.1.1 2018-03-13 [1] CRAN (R 3.5.2)

## bindrcpp \* 0.2.2 2018-03-29 [1] CRAN (R 3.5.2)

## cli 1.0.1 2018-09-25 [1] CRAN (R 3.5.2)

## colorspace 1.4-0 2019-01-13 [1] CRAN (R 3.5.2)

## crayon 1.3.4 2017-09-16 [1] CRAN (R 3.5.2)

## digest 0.6.18 2018-10-10 [1] CRAN (R 3.5.2)

## dplyr \* 0.7.8 2018-11-10 [1] CRAN (R 3.5.2)

## evaluate 0.12 2018-10-09 [1] CRAN (R 3.5.2)

## ggplot2 \* 3.1.0 2018-10-25 [1] CRAN (R 3.5.2)

## glue 1.3.0 2018-07-17 [1] CRAN (R 3.5.2)

## gtable 0.2.0 2016-02-26 [1] CRAN (R 3.5.2)

## highr 0.7 2018-06-09 [1] CRAN (R 3.5.2)

## htmltools 0.3.6 2017-04-28 [1] CRAN (R 3.5.2)

## knitr 1.21 2018-12-10 [1] CRAN (R 3.5.2)

## labeling 0.3 2014-08-23 [1] CRAN (R 3.5.2)

## lazyeval 0.2.1 2017-10-29 [1] CRAN (R 3.5.2)

## magrittr 1.5 2014-11-22 [1] CRAN (R 3.5.2)

## munsell 0.5.0 2018-06-12 [1] CRAN (R 3.5.2)

## packrat 0.5.0 2018-11-14 [1] CRAN (R 3.5.2)

## pillar 1.3.1 2018-12-15 [1] CRAN (R 3.5.2)

## pkgconfig 2.0.2 2018-08-16 [1] CRAN (R 3.5.2)

## plyr 1.8.4 2016-06-08 [1] CRAN (R 3.5.2)

## purrr 0.2.5 2018-05-29 [1] CRAN (R 3.5.2)

## R6 2.3.0 2018-10-04 [1] CRAN (R 3.5.2)

## Rcpp 1.0.0 2018-11-07 [1] CRAN (R 3.5.2)

## rlang 0.3.1 2019-01-08 [1] CRAN (R 3.5.2)

## rmarkdown 1.11 2018-12-08 [1] CRAN (R 3.5.2)

## scales 1.0.0 2018-08-09 [1] CRAN (R 3.5.2)

## sessioninfo 1.1.1 2018-11-05 [1] CRAN (R 3.5.2)

## stringi 1.2.4 2018-07-20 [1] CRAN (R 3.5.2)

## stringr \* 1.3.1 2018-05-10 [1] CRAN (R 3.5.2)

## tibble 2.0.1 2019-01-12 [1] CRAN (R 3.5.2)

## tidyselect 0.2.5 2018-10-11 [1] CRAN (R 3.5.2)

## viridisLite 0.3.0 2018-02-01 [1] CRAN (R 3.5.2)

## withr 2.1.2 2018-03-15 [1] CRAN (R 3.5.2)

## xfun 0.4 2018-10-23 [1] CRAN (R 3.5.2)

## yaml 2.2.0 2018-07-25 [1] CRAN (R 3.5.2)

]czo1OTpcIjxjb2RlIHN0eWxlPVwiZGlzcGxheTogbm9uZTtcIj48Y29kZSBzdHlsZT1cImRpc3BsYXk6IG5vbmU7XCI+ClwiO3tbJiomXX0=[

]czoxOlwiClwiO3tbJiomXX0=[

**Settings**

]czo2MDpcIjxjb2RlIHN0eWxlPVwiZGlzcGxheTogbm9uZTtcIj4KPGNvZGUgc3R5bGU9XCJkaXNwbGF5OiBub25lO1wiPgpcIjt7WyYqJl19[

# settings.yml

set\_size: [50000, 100000, 150000, 200000]

num\_groups: [2, 3, 4, 5, 6, 7, 8, 9, 10]

benchmark\_repetitions: 10

num\_test\_sets: 10

max\_value: 500

word\_length: 10

]czo2MDpcIjxjb2RlIHN0eWxlPVwiZGlzcGxheTogbm9uZTtcIj4KPGNvZGUgc3R5bGU9XCJkaXNwbGF5OiBub25lO1wiPgpcIjt7WyYqJl19[

**Data creation and benchmarking scripts**

]czo2MDpcIjxjb2RlIHN0eWxlPVwiZGlzcGxheTogbm9uZTtcIj4KPGNvZGUgc3R5bGU9XCJkaXNwbGF5OiBub25lO1wiPgpcIjt7WyYqJl19[

]czoxOlwiClwiO3tbJiomXX0=[

]czoxOlwiClwiO3tbJiomXX0=[

# main.R

# Global variables ----------------------------------------------

# Set this to FALSE if you want to run the complete analysis

running\_test <- TRUE

vars <- yaml::read\_yaml("./settings.yml")

set\_size <- vars$set\_size

num\_groups <- vars$num\_groups

benchmark\_repetitions <- vars$benchmark\_repetitions

num\_test\_sets <- vars$num\_test\_sets

max\_value <- vars$max\_value

word\_length <- vars$word\_length

# Test variables ------------------------------------------------

if(running\_test){

set\_size <- seq.int(0L, 60L, 30L)

num\_groups <- c(2L:3L)

benchmark\_repetitions <- 2L

num\_test\_sets <- 2L

}

# Libraries -----------------------------------------------------

suppressPackageStartupMessages(library(foreach))

suppressPackageStartupMessages(library(doParallel))

# Setup parallel ------------------------------------------------

num\_cores <- detectCores() - 1

these\_cores <- makeCluster(num\_cores, type = "PSOCK")

registerDoParallel(these\_cores)

# Functions -----------------------------------------------------

run\_benchmark <- function(as){

source("test\_cast.R")

num\_groups <- as["num\_groups"]

set\_size <- as["set\_size"]

num\_test\_sets <- as["num\_test\_sets"]

word\_length <- as["word\_length"]

max\_value <- as["max\_value"]

test\_data <- prepare\_test\_data(set\_size, num\_groups, word\_length, max\_value)

perform\_benchmark(test\_data, benchmark\_repetitions)

}

# Setup and run tests -------------------------------------------

set\_size <- set\_size[set\_size > 0]

analysis\_comb <- expand.grid(num\_groups, set\_size)

analysis\_settings <- vector("list", length = nrow(analysis\_comb))

for(i in seq\_len(nrow(analysis\_comb))){

analysis\_settings[[i]] <- c(num\_groups =analysis\_comb[i, "Var1"],

set\_size = analysis\_comb[i, "Var2"],

word\_length = word\_length,

max\_value = max\_value,

benchmark\_repetitions = benchmark\_repetitions)

}

for(as in analysis\_settings){

num\_groups <- as["num\_groups"]

set\_size <- as["set\_size"]

report\_str <- paste("ng:", num\_groups,

"- setsize:", set\_size, "\n")

cat(report\_str)

rds\_file\_name <- paste0("./output/benchmark\_setsize-", set\_size,

"\_ng-", num\_groups, ".rds")

bm\_res <- foreach(seq\_len(num\_test\_sets), .combine = "rbind") %dopar% {

run\_benchmark(as)

}

bm\_res <- dplyr::mutate(bm\_res, `Number groups` = num\_groups,

`Set size` = set\_size)

saveRDS(bm\_res, rds\_file\_name)

report\_str

}

]czo2MDpcIjxjb2RlIHN0eWxlPVwiZGlzcGxheTogbm9uZTtcIj4KPGNvZGUgc3R5bGU9XCJkaXNwbGF5OiBub25lO1wiPgpcIjt7WyYqJl19[

]czoxOlwiClwiO3tbJiomXX0=[

# test\_cast.R

setup <- function(){

library(data.table)

library(tidyr)

library(dplyr)

library(Matrix.utils)

library(tibble)

}

prepare\_test\_data <- function(set\_size, num\_groups, word\_length, sample\_int\_n){

calc\_subset\_size <- function(){

subset\_size <- 0

while(subset\_size == 0 | subset\_size > set\_size){

subset\_size <- abs(round(set\_size - set\_size/(3 + rnorm(1))))

}

subset\_size

}

words <- stringi::stri\_rand\_strings(set\_size, word\_length)

subset\_sizes <- replicate(num\_groups, calc\_subset\_size())

purrr::map\_df(subset\_sizes, function(subset\_size){

tibble::tibble(Variable = sample(words, subset\_size),

Value = sample.int(sample\_int\_n, subset\_size, replace = TRUE),

Group = stringi::stri\_rand\_strings(1, word\_length))

})

}

test\_tidyr <- function(test\_df){

test\_df %>%

spread(Variable, Value, fill = 0L) %>%

tibble::column\_to\_rownames(var = "Group") %>%

as.matrix.data.frame()

}

test\_xtabs <- function(test\_df){

xtabs(Value ~ Group + Variable, data = test\_df)

}

test\_dMcast <- function(test\_df){

class(test\_df) <- "data.frame"

dMcast(test\_df, Group ~ Variable, value.var = "Value")

}

test\_dcast <- function(test\_df){

test\_df\_dt <- data.table(test\_df)

dcast(test\_df\_dt, Group ~ Variable, value.var = "Value", fill = 0) %>%

tibble::column\_to\_rownames(var = "Group") %>%

as.matrix.data.frame()

}

perform\_benchmark <- function(test\_df, benchmark\_repetitions){

suppressPackageStartupMessages(setup())

bm\_res <- microbenchmark::microbenchmark(

Spread = test\_tidyr(test\_df = test\_df),

Xtabs = test\_xtabs(test\_df = test\_df),

dMcast = test\_dMcast(test\_df = test\_df),

dcast = test\_dcast(test\_df = test\_df),

times = benchmark\_repetitions

)

class(bm\_res) <- "data.frame"

bm\_res %>%

mutate(time = microbenchmark:::convert\_to\_unit(time, "ms")) %>%

rename(Method = expr, `Time (ms)` = time)

}