datelife Functions Benchmark

Luna L. Sanchez Reyes

17 February 2019

I. Benchmarking `datelife` Functions to Get Source Data

You'll need `datelife` and `microbenchmark` packages.

```
install.packages("microbenchmark")
install.packages("datelife")
library(microbenchmark)
library(datelife)
```

Then, we generate a vector of seeds to use before each test to be able to reproduce the results afterwards:

```
set.seed(10)
seeds <- runif(100, 1, 1e9)
# set.seed only accepts numbers up to 9 integers-ish:
# set.seed(2140000000) # this works
# works with numbers <=2.14e+09
save(seeds, file="data/1_datasource/1_name_samples/seeds.RData")
```

1. Function to search input taxa across a chronogram database

1.a. Generating a meaningful set of inputs to profile

The `datelife` function that performs the chronogram searches is called `datelife_search` (previously called `get_filtered_results`) To benchmark this function, we used species names of birds (any species within the Aves class) as input. Running time of the function was tested with a different number of input taxa: 10, 100, 200, 300, 400, 500, 700, 1000, 1500, 2000, 3000, 5000, 7000, 8000, 9000, 10000 and up to all named bird species in Open Tree Taxonomy (OTT). To do this, first we obtained all named bird species from OTT with the `make_datelife_query` function:

```
install.packages("datelife")
library(datelife)
aves.spp <- make_datelife_query(input="Aves", sppfromtaxon=TRUE) # 12750 spp names
save("aves.spp", file="data/1_datasource/1_name_samples/aves.spp.RData")
```

```
names(aves.spp)
```

```
## [1] "phy"           "cleaned.names"
```

```
length(aves.spp$cleaned.names)
```

```
## [1] 12750
```

So, there are 12750 named bird species in the OTT. Then, we generated a character vector of randomly sampled bird names for each input size. We saved these independently to ensure reproducibility:

```
ninput <- c(10, 100, 200, 300, 400, 500, 700, 1000, 1500, 2000, 3000, 5000, 7000,
            8000, 9000, 10000)
for (i in ninput){
    set.seed(seeds[1])
    x <- sample(aves.spp$cleaned.names, i)
    xname <- paste0("spp",i)
    assign(xname, x)
    save(list=xname, file=paste0(xname,".RData"))
}
```

1.b. Profiling

This was my first time using `microbenchmark` to profile running time of functions, so I did a little test first, using 400 bird names drawn at random from `aves.spp$cleaned.names` vector, just to look at the structure of the output and all:

```
set.seed(seeds[1])
spp400.1 <- sample(aves.spp$cleaned.names, 400)
aves400.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp400.1),
                                                   times=100L)
save(aves400.1.gfr.runtime_2017.12.28, file =
       "data/1_datasource/2_tests/1_same_spp_names/aves400.1.gfr.runtime_2017.12.28.RData")

aves400.1.gfr.runtime_2017.12.28

## Unit: milliseconds
##                                 expr     min       lq     mean   median
##  GetFilteredResults(input = spp400) 477.973 488.9229 513.7463 516.8565
##        uq      max neval
##  524.9569 568.6478   100

names(aves400.1.gfr.runtime_2017.12.28)

## [1] "expr" "time"

class(aves400.1.gfr.runtime_2017.12.28)

## [1] "microbenchmark" "data.frame"

length(aves400.1.gfr.runtime_2017.12.28)

## [1] 2

first_test <- rbind(aves400.1.gfr.runtime_2017.12.28, aves400.1.gfr.runtime_2017.12.28)
# res <- rbind(thraupidae.ed.runtime_2017.12.28, thraupidae.ed.runtime_2017.12.28)
first_test

## Unit: milliseconds
##                                 expr     min       lq     mean   median
##  GetFilteredResults(input = spp400) 477.973 488.9229 513.7463 516.8565
##        uq      max neval
##  524.9569 568.6478   200

class(first_test)

## [1] "microbenchmark" "data.frame"

length(first_test)

## [1] 2
```

```
pdf(file = "plots/plot1_first_test.pdf", height = 3, width = 7)
microbenchmark:::autoplot.microbenchmark(first_test)
dev.off()
```

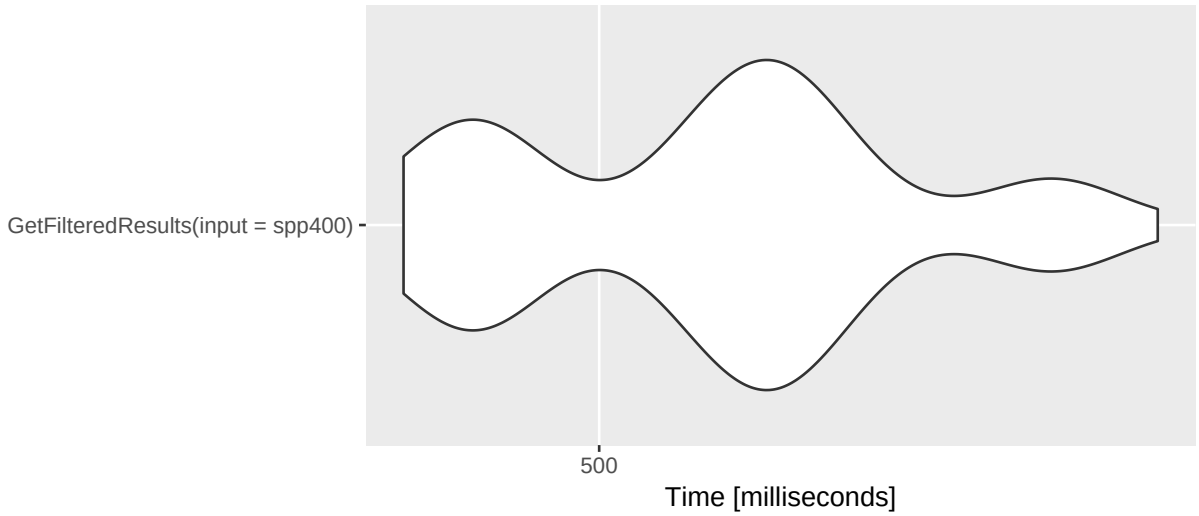

Figure 1: First try with `microbenchmark` function.

Confident of understanding the structure of a microbenchmark output, we started the formal benchmarking tests.

Up to 1k names, we ran `microbenchmark` on the same R console and saved everything at the end with a loop:

```
aves10.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp10),
                                                  times=100L)
aves100.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp100),
                                                   times=100L)
aves200.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp200),
                                                   times=100L)
aves300.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp300),
                                                   times=100L)
aves400.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp400),
                                                   times=100L)
aves500.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp500),
                                                   times=100L)
aves700.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp700),
                                                   times=100L)
aves1000.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp1000),
                                                    times=100L)
for(i in c(10,100,200,300,400,500,700,1000)){
    xname <- paste0("data/1_datasource/2_tests/1_same_spp_names/aves", i,
                    ".1.gfr.runtime_2017.12.28")
    save(list=xname, file=paste0(xname, ".RData"))
}
```

We ran each of the following in a different R console process and saved the results independently at the end of each run (they're named with teh date we started running them):

```
aves1500.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp1500),
                                                    times=100L)
```

```
save(aves1500.1.gfr.runtime_2017.12.28, file =
    "data/1_datasource/2_tests/1_same_spp_names/aves1500.1.gfr.runtime_2017.12.28.RData")
aves2000.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp2000),
                                                    times=100L)
save(aves2000.1.gfr.runtime_2017.12.28, file =
    "data/1_datasource/2_tests/1_same_spp_names/aves2000.1.gfr.runtime_2017.12.28.RData")
aves3000.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp3000),
                                                    times=100L)
save(aves3000.1.gfr.runtime_2017.12.28, file =
    "data/1_datasource/2_tests/1_same_spp_names/aves3000.1.gfr.runtime_2017.12.28.RData")
aves5000.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp5000),
                                                    times=100L)
save(aves5000.1.gfr.runtime_2017.12.28, file =
    "data/1_datasource/2_tests/1_same_spp_names/aves5000.1.gfr.runtime_2017.12.28.RData")
aves7000.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp7000),
                                                    times=100L)
save(aves7000.1.gfr.runtime_2017.12.28, file =
    "data/1_datasource/2_tests/1_same_spp_names/aves7000.1.gfr.runtime_2017.12.28.RData")
aves8000.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp8000),
                                                    times=100L)
save(aves8000.1.gfr.runtime_2017.12.28, file =
    "data/1_datasource/2_tests/1_same_spp_names/aves8000.1.gfr.runtime_2017.12.28.RData")
aves9000.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp9000),
                                                    times=100L)
save(aves9000.1.gfr.runtime_2017.12.28, file =
    "data/1_datasource/2_tests/1_same_spp_names/aves9000.1.gfr.runtime_2017.12.28.RData")
aves10000.1.gfr.runtime_2017.12.28 <- microbenchmark(get_datelife_result(input=spp10000),
                                                     times=100L)
save(aves10000.1.gfr.runtime_2017.12.28, file =
    "data/1_datasource/2_tests/1_same_spp_names/aves10000.1.gfr.runtime_2017.12.28.RData")
aves.all.gfr.runtime_2017.12.29 <- microbenchmark(get_datelife_result(
  input=aves.spp$cleaned.names), times=100L)
save(aves.all.gfr.runtime_2017.12.29, file =
       "data/1_datasource/2_tests/0_all_names/aves.all.gfr.runtime_2017.12.29.RData")
```

1.c. Plotting the profiles

To plot the results, we need to load all data sets into the same R console, `rbind` them and `autoplot` them:

```
res28 <- c()
for(i in ninput){
    xname <- paste0("aves",i,".1.gfr.runtime_2017.12.28")
    x <- paste0(xname, ".RData")
    load(x)
    res28 <- rbind(res28, get(xname))
}
res28 <- rbind(res, aves.all.gfr.runtime_2017.12.29)
pdf(file = "plots/plot2_res28.pdf", height = 3, width = 7)
microbenchmark:::autoplot.microbenchmark(res28)
dev.off()
```

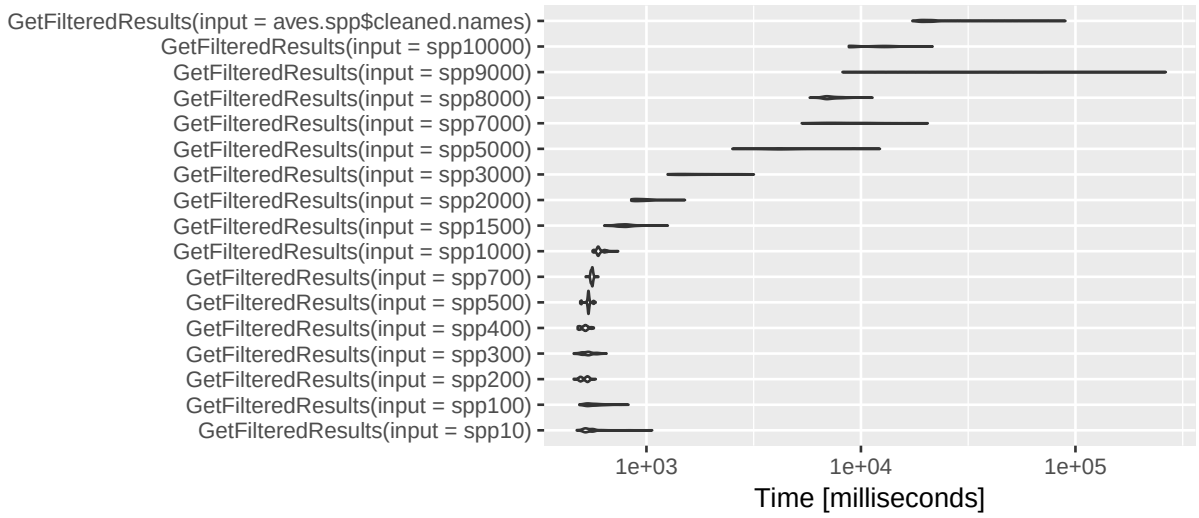

Figure 2: First profiling results. Profiling was run simultaneously for input size $> 1\ 000$ names.

Results look weirdly flat/overdispersed. This might be because we ran some of the tests simultaneously on the same computer. So we ran the whole thing again, one test after another (i.e., not running tests at the same time in the same computer):

```
ninput <- c(10, 100, 200, 300, 400, 500, 700, 1000, 1500, 2000, 3000, 5000, 7000,
            8000, 9000, 10000)
for(i in ninput){
    xname <- paste0("spp",i)
    load(paste0(xname,".RData"))
    x <- microbenchmark(get_datelife_result(input=get(xname), process_input=TRUE),
                        times=100L) # input must be processed :)
    # y <- levels(x$expr)
    # levels(x$expr)[levels(x$expr==y)] <- paste0(i, " names")
    levels(x$expr)[1] <- paste0(i, " names")
    xnameobj <- paste0("aves",i,".1.gfr.runtime_2017.12.29")
    assign(xnameobj, x)
    save(list=xnameobj, file=paste0("data/1_datasource/2_tests/1_same_spp_names/",
                                    xnameobj,".RData"))
    rm(list=xnameobj)
}
```

Then we load and `rbind` the outputs in a new console and use `autoplot` to visualize results:

```
res29 <- c()
for(i in ninput){
    xname <- paste0("aves",i,".1.gfr.runtime_2017.12.29")
    res29 <- rbind(res29, get(xname))
}

res29 <- rbind(res29, aves.all.gfr.runtime_2017.12.29)

pdf(file = "plots/plot2_res29.pdf", height = 3, width = 7)
microbenchmark:::autoplot.microbenchmark(res29)
dev.off()
```

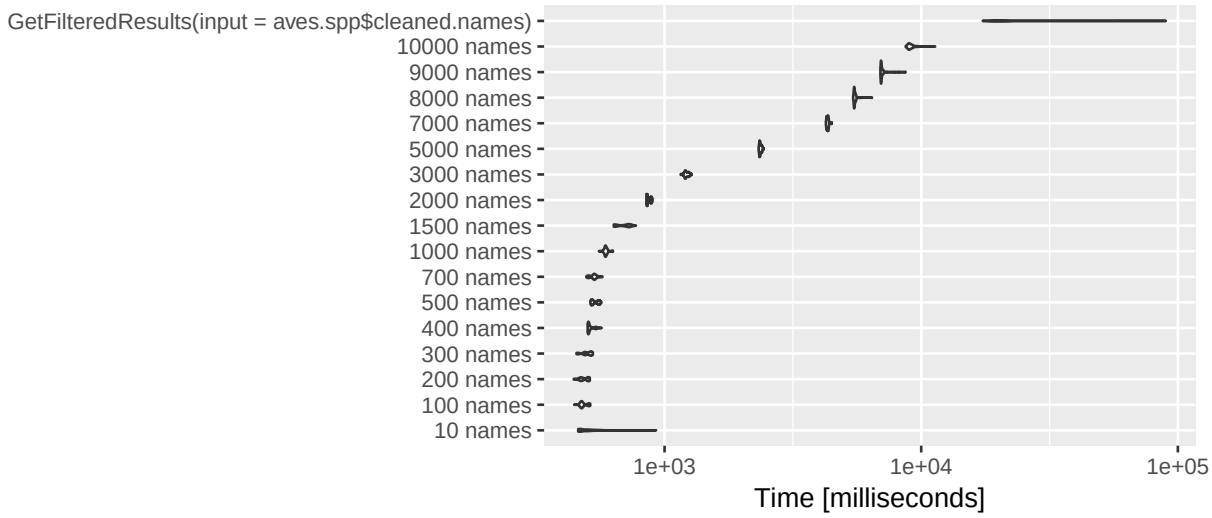

Figure 3: Second profiling results. Tests run consecutively on the same machine.

This is good, but now we want a prettier plot, so we're writing some code for that:

```
autoplot.gfr <- function (object, log = TRUE, y_min = 0.1, y_max = 1.05 * max(object$time), flip = TRUE
    object$Time <- microbenchmark:::convert_to_unit(object$time, "t")
    # changing the name of the element itself is the easiest way to make it appear as the axis label
    # object$'Query Length' <- object$expr #changing for a name with spaces won't work...
    plt <- ggplot2::ggplot(object, ggplot2::aes_string(x = "expr", y = "Time"))
    plt <- plt + coord_cartesian(ylim = c(y_min, y_max))
    plt <- plt + stat_ydensity()
    # plt <- plt + xlim(levels(object$expr)[length(levels(object$expr)):1])
    plt <- plt + scale_x_discrete(name = "")
    plt <- plt + theme(axis.text.x = element_text(angle= xlab_angle))
    plt <- plt + theme(axis.text.y = element_text(angle= ylab_angle))
    plt <- if (log) {
        # plt + scale_y_log10(name = sprintf("", attr(object$ntime, "unit")))
          # this does not work...
        # plt + scale_y_log10(name = sprintf("Time", attr(object$ntime, "unit")))
          # this does not work...
        # plt + scale_y_log10(name = "Seconds") # this does not work either...
        plt + scale_y_log10(breaks=c(1e+03, 1e+035, 1e+04, 1e+045, 1e+05),
                            labels=c("1e+03"="1s", "1e+035"="", "1e+04"="10s",
                                     "1e+045"="", "1e+05"="100s"), position="top")
    }
    else {
        plt + scale_y_continuous(name = sprintf("Time [%s]", attr(object$ntime, "unit")))
    }
  if(flip){
    plt <- plt + ggplot2::coord_flip() # these exchanges the axis
    # if I inactivate this and y_min is 0 I get the following Warning message:
    # Transformation introduced infinite values in continuous y-axis
    # that is because log(0) = Inf
  }
    plt
}
```

```
pdf(file = "plots/plot4_res29.pdf", height = 3, width = 7)
plt <- autoplot.gfr(res29)
plt <- plt + theme(plot.margin = margin(topmargin, 0.5, 0, 0, "inch"))
print(plt)
dev.off()
```

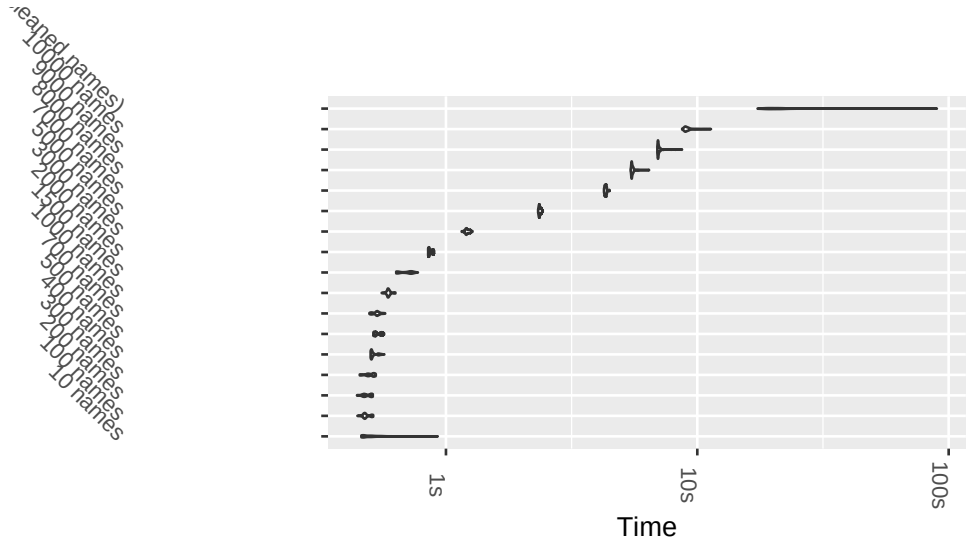

Figure 4: Second profiling results. Tests run consecutively on the same machine. Pretty plot.

Now with a non-logarithmic scale:

```
pdf(file = "plots/plot4_res29notlog.pdf", height = 3, width = 7)
plt <- autoplot.gfr(res29, log = FALSE)
plt <- plt + theme(plot.margin = margin(topmargin, 0.5, 0, 0, "inch"))
print(plt)
dev.off()
```

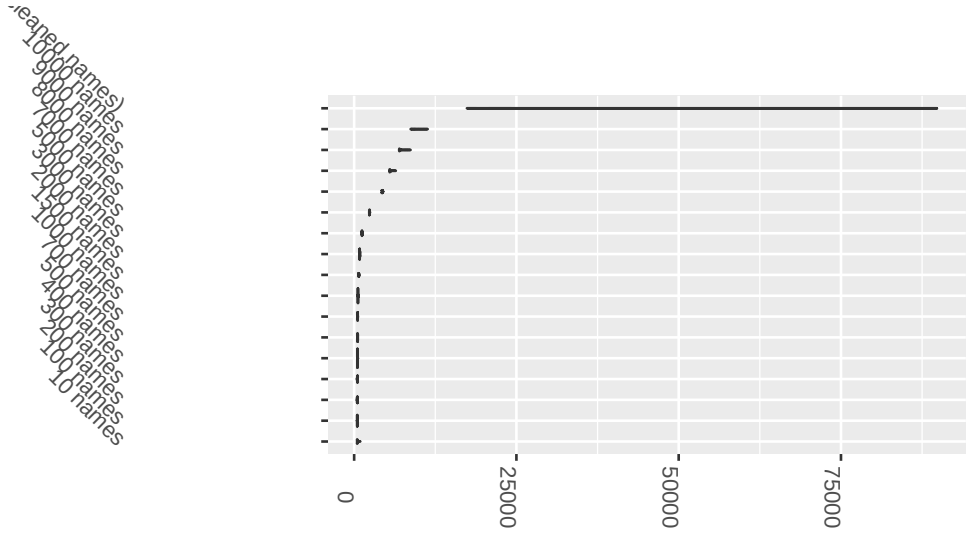

Figure 5: Second profiling results. Tests run consecutively on the same machine. Pretty plot not log.

Result graphs look good at this point, although a bit crowded on the y axis labels. So let's change the name

of `$expr` element from `aves.all.gfr.runtime_2017.12.29`, which is too long:

```
levels(res29$expr)
```

```
##  [1] "10 names"
##  [2] "100 names"
##  [3] "200 names"
##  [4] "300 names"
##  [5] "400 names"
##  [6] "500 names"
##  [7] "700 names"
##  [8] "1000 names"
##  [9] "1500 names"
## [10] "2000 names"
## [11] "3000 names"
## [12] "5000 names"
## [13] "7000 names"
## [14] "8000 names"
## [15] "9000 names"
## [16] "10000 names"
## [17] "GetFilteredResults(input = aves.spp$cleaned.names)"
```

```
levels(res29$expr)[17] <- "all names (12750)"
```

Also, now we wanna have the time on the y axis. So let's fix that:

```
pdf(file = "plots/plot5_res29.pdf", height = 3, width = 7)
plt <- autoplot.gfr(res29, flip = FALSE, xlab_angle = 90, ylab_angle = 0, y_min = 400, y_max = 1e+5)
plt <- plt + theme(plot.margin = margin(0.5, 0.5, 0, 0.5, "inch"))
print(plt)
dev.off()
```

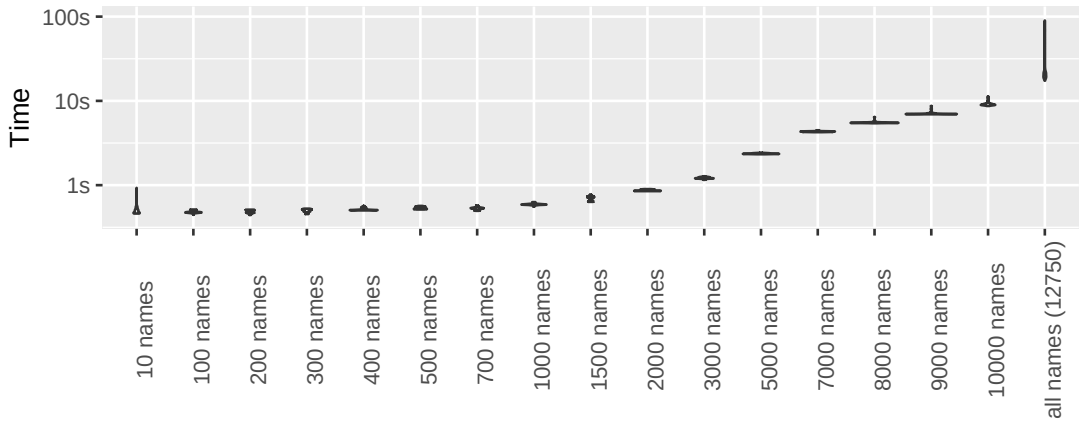

Figure 6: Second profiling results. Tests run consecutively on the same machine. Pretty plot with time in y axis and good x axis labels.

Now, that we have the hang of it, let's start some serious profiling. First, for each cohort input name, let's sample 100 different vectors of bird names:

```
ninput <- c(10, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000, 3000,
```

```
            4000, 5000, 6000, 7000, 8000, 9000, 10000)

for(i in ninput){
    x <- vector(mode="list")
    for(j in 1:100){
        x <- c(x, list(sample(aves.spp$cleaned.names, i)))
    }
    xname <- paste0("random_sample_",i, "_aves_spp")
    assign(xname, x)
    save(list=xname, file=paste0("data/1_datasource/1_name_samples/", xname,".RData"))
}
```

Now we noticed a slowdown on the first run, probably because cache is loading for the very first time. So we launched a first run that was not recorded, to make sure everything is loaded when we start the actual tests:

```
asd <- "((Zea mays,Oryza sativa),((Arabidopsis thaliana,(Glycine max,Medicago sativa)),
        Solanum lycopersicum)Pentapetalae);"
get_datelife_result(input=asd, process_input=TRUE)
```

Now we can run the tests consecutively again:

```
for(i in ninput){
    xname <- paste0("random_sample_",i, "_aves_spp")
    setwd("data/1_datasource/1_name_samples")
    load(file=paste0(xname,".RData"))
    y <- microbenchmark(get_datelife_result(input=get(xname)[[1]],
                                            process_input = TRUE),times=1L)
    # input should be processed? we are then testing two functions in here...
    levels(y$expr)[1] <- paste0(i, " names")
    for(j in 2:100){
        yy <- microbenchmark(get_datelife_result(input=get(xname)[[j]], process_input=TRUE),
                             times=1L)
        levels(yy$expr)[1] <- paste0(i, " names")
        y <- rbind(y, yy)
    }
    rm(list=xname)
    xnameobj <- paste0("gfr_runtime_2018.01.10_", i,"_aves_spp")
    assign(xnameobj, y)
    save(list=xnameobj, file=paste0("data/1_datasource/2_tests/2_random_spp_names/1_gfr",
                                    xnameobj,".RData"))
    rm(list=xnameobj)
}
aves.all.gfr.runtime_2018.01.12 <- microbenchmark(get_datelife_result(input =
                                  aves.spp$cleaned.names), times=100L)
levels(aves.all.gfr.runtime_2018.01.12$expr)[1] <- "12750"
save(aves.all.gfr.runtime_2018.01.12,
     file="data/1_datasource/2_tests/0_all_names/aves.all.gfr.runtime_2018.01.12.RData")
```

Now, load and `rbind` the stuff if you opened a new session:

```
res01 <- c()
for(i in ninput){
    xname <- paste0("gfr_runtime_2018.01.10_",i,"_aves_spp")
    x <- paste0("data/1_datasource/2_tests/2_random_spp_names/1_gfr", xname, ".RData")
    load(x)
    res01 <- rbind(res01, get(xname))
}
```

```
load(file="data/1_datasource/2_tests/0_all_names/aves.all.gfr.runtime_2018.01.12.RData")
```

```
res01 <- rbind(res01, aves.all.gfr.runtime_2018.01.12)
```

And plot again

We need to change xlabels. One way to do this is to change the levels of the `expr` element:

```
levels(res01$expr) <- as.character(c(ninput, "all Aves (12750)"))
```

Then set the time limits:

```
y_min <- 200
y_max <- 1e+5
res01$Time <- microbenchmark:::convert_to_unit(res$time, "t")
```

```
## Error in find_prefix(x * 1e-09, minexp = -9, maxexp = 0, mu = FALSE): object 'res' not found
```

```
#changing the name of the element itself is the easiest way to make it appear as axis label
# object$'Query Length' <- object$expr
# note that if you try to use spaces here it won't work...
plt <- ggplot2::ggplot(res01, ggplot2::aes_string(x = "expr", y = "Time"))
plt <- plt + ggplot2::coord_cartesian(ylim = c(y_min, y_max))
plt <- plt + ggplot2::stat_ydensity()
plt <- plt + ggplot2::scale_x_discrete(name = "Query Length (log10)",
       labels=c("10 names" = "1",
                "100 names" = expression(10^2),
                "200 names" = expression(2*"x"*10^2),
                "300 names" = expression(3*"x"*10^2),
                "400 names" = expression(4*"x"*10^2),
                "500 names" = expression(5*"x"*10^2),
                "700 names" = expression(7*"x"*10^2),
                "1000 names" = expression(10^3),
                "1500 names" = expression(1.5*"x"*10^3),
                "2000 names" = expression(2*"x"*10^3),
                "3000 names" = expression(3*"x"*10^3),
                "4000 names" = expression(4*"x"*10^3),
                "5000 names" = expression(5*"x"*10^3),
                "6000 names" = expression(6*"x"*10^3),
                "7000 names" = expression(7*"x"*10^3),
                "8000 names" = expression(8*"x"*10^3),
                "9000 names" = expression(9*"x"*10^3),
                "10000 names" = expression(10^4),
                "12750" = expression(1.275*"x"*10^4)
       ))
plt <- plt + ggplot2::theme(axis.text.x = ggplot2::element_text(angle=45, hjust=1))
plt <- plt + ggplot2::theme(axis.text.y = ggplot2::element_text(angle=0))
plt <- plt + ggplot2::scale_y_log10(name="Time (seconds)", breaks=c(1e+03, 1e+04, 3e+04, 1e+05),
        labels=c("1e+03"="1 s", "1e+04"="10 s", "3e+04"="30 s", "1e+05"="100 s"),
            position="left",
            sec.axis = ggplot2::sec_axis(~ . *1, name="Time (minutes)",
            breaks=c(6e+03, 3e+04, 6e+04, 9e+04),
            labels=c("6e+03"="0.1 min", "3e+04"="0.5 min", "6e+04"="1 min", "9e+04"="1.5 min")))
plt
```

```
## Error in FUN(X[[i]], ...): object 'Time' not found
```

I didn't like the labels, let's try these ones:

```
plt <- plt + ggplot2::scale_x_discrete(name = "Query Length",
       labels=c("10 names" = "10",
                "100 names" = "100",
                "200 names" = "200",
                "300 names" = "300",
                "400 names" = "400",
                "500 names" = "500",
                "700 names" = "700",
                "1000 names" = expression(1~0*0*0),
                "1500 names" = expression(1~500),
                "2000 names" = expression(2~0*0*0),
                "3000 names" = expression(3~0*0*0),
                "4000 names" = expression(4~0*0*0),
```

```
                  "5000 names" = expression(5~0*0*0),
                  "6000 names" = expression(6~0*0*0),
                  "7000 names" = expression(7~0*0*0),
                  "8000 names" = expression(8~0*0*0),
                  "9000 names" = expression(9~0*0*0),
                  "10000 names" = expression(10~0*0*0),
                  "12750" = expression(12~750)))
plt
```

```
## Error in FUN(X[[i]], ...): object 'Time' not found
```

Figure 7: plot of chunk unnamed-chunk-26