Today I was struggling with a relatively simple operation: unnest() from the tidyr package. What it’s supposed to do is pretty simple. When you have a data.frame where one or multiple columns are lists, you can unlist these columns while duplicating the information in other columns if the length of an element is larger than 1.

library(tibble)

df <- tibble(

a = LETTERS[1:5],

b = LETTERS[6:10],

list\_column = list(c(LETTERS[1:5]), "F", "G", "H", "I")

)

df

## # A tibble: 5 x 3

## a b list\_column

##

## 1 A F

## 2 B G

## 3 C H

## 4 D I

## 5 E J

library(tidyr)

unnest(df, list\_column)

## # A tibble: 9 x 3

## a b list\_column

##

## 1 A F A

## 2 A F B

## 3 A F C

## 4 A F D

## 5 A F E

## 6 B G F

## 7 C H G

## 8 D I H

## 9 E J I

I came across this a lot while working on data from Twitter since individual tweets can contain multiple hashtags, mentions, URLs and so on, which is why they are stored in lists. unnest() is really helpful and very flexible in my experience since it makes creating, for example, a table of top 10 hashtags a piece of cake.

However, on large datasets, unnest() has its limitations (as I found out today). On a set with 1.8 million tweets, I was barely able to unnest the URL column and it would take forever on my laptop or simply crash at some point. In a completely new environment, unnesting the data took half an hour.

So let’s cut this time down to 10 seconds with data.table. In data.table, you would unlist like this[1](https://johannesbgruber.eu/post/a-faster-unnest/#fn1):

library(data.table)

dt <- as.data.table(df)

dt[, list(list\_column = as.character(unlist(list\_column))), by = list(a, b)]

## a b list\_column

## 1: A F A

## 2: A F B

## 3: A F C

## 4: A F D

## 5: A F E

## 6: B G F

## 7: C H G

## 8: D I H

## 9: E J I

This is quite a bit longer than the tidyr code. So I wrapped it in a short function (note, that most of the code deals with quasiquotation so we can use it the same way as the original unnest()):

library(rlang)

unnest\_dt <- function(tbl, col) {

tbl <- as.data.table(tbl)

col <- ensyms(col)

clnms <- syms(setdiff(colnames(tbl), as.character(col)))

tbl <- as.data.table(tbl)

tbl <- eval(

expr(tbl[, as.character(unlist(!!!col)), by = list(!!!clnms)])

)

colnames(tbl) <- c(as.character(clnms), as.character(col))

tbl

}

On the surface, it does the same as unnest:

unnest\_dt(df, list\_column)

## a b list\_column

## 1: A F A

## 2: A F B

## 3: A F C

## 4: A F D

## 5: A F E

## 6: B G F

## 7: C H G

## 8: D I H

## 9: E J I

But the function is extremely fast and lean. To show this, I do some benchmarking on a larger object. I scale the example ‘data.frame’ up from 5 to 50,000 rows since the overhead of loading a function will influence runtime much stronger on small-n data.

library(bench)

df\_large <- dplyr::sample\_frac(df, 10000, replace = TRUE)

res <- mark(

tidyr = unnest(df\_large, list\_column),

dt = unnest\_dt(df\_large, list\_column)

)

res

## # A tibble: 2 x 6

## expression min median `itr/sec` mem\_alloc `gc/sec`

##

## 1 tidyr 52.4s 52.4s 0.0191 16.77GB 6.38

## 2 dt 14.3ms 18.5ms 50.0 9.56MB 10.00

summary(res, relative = TRUE)

## # A tibble: 2 x 6

## expression min median `itr/sec` mem\_alloc `gc/sec`

##

## 1 tidyr 3666. 2832. 1 1796. 1

## 2 dt 1 1 2617. 1 1.57

As you can see, data.table is 3666 times faster. That is pretty insane. But what is often even more important, the memory consumption is negligible with the data.table function compared to tidyr.

Admittedly the function is not perfect. It is far less flexible than unnest, especially since it only runs on one variable at the time. However, this covers 95% of my usage of unnest and I would only consider including it in a script if performance is key.