TFLink use case 1

Network visualisation of the common target genes of two transcription factors

1 The aim of this use case

Here we want to check and visualise the common target genes of two transcription factors. We describe how to find transcription factors which share common target genes. We create a transcription factor - target gene interaction graph of the **STAT5A** and **STAT5B** transcription factors using the `igraph R` package. We also show how to create the same transcription factor - target gene interaction graph by using the Cytoscape software.

2 Input data

Let's download the human (*Homo sapiens*) *interaction table* including all (small- and large-scale) interactions from the **Download** part of the TFLink website, and read it in with the `fread` command of the `data.table` package that reads huge tables effectively. We also will use the `tidyverse` package.

```
library(data.table)
library(tidyverse)
Interaction_tab <- fread("TFLink_Homo_sapiens_interactions_All_simpleFormat_v1.0.tsv")
```

Let's see the first few rows of the interaction table.

```
Interaction_tab %>%
  slice(1:5)
```

Head of interaction table (continued below)

UniprotID.TF	UniprotID.Target	NCBI.GeneID.TF	NCBI.GeneID.Target	Name.TF	Name.Target	Detection.method
Q9H9S0	O94907	79923	22943	NANOG	DKK1	chromatin immunoprecipitation assay;inferred by curator
P37231	P10826	5468	5915	PPARG	RARB	chromatin immunoprecipitation assay;inferred by curator
P10242	P08047	4602	6667	MYB	SP1	chromatin immunoprecipitation assay;inferred by curator
P31269	Q02363	3205	3398	HOXA9	ID2	inferred by curator

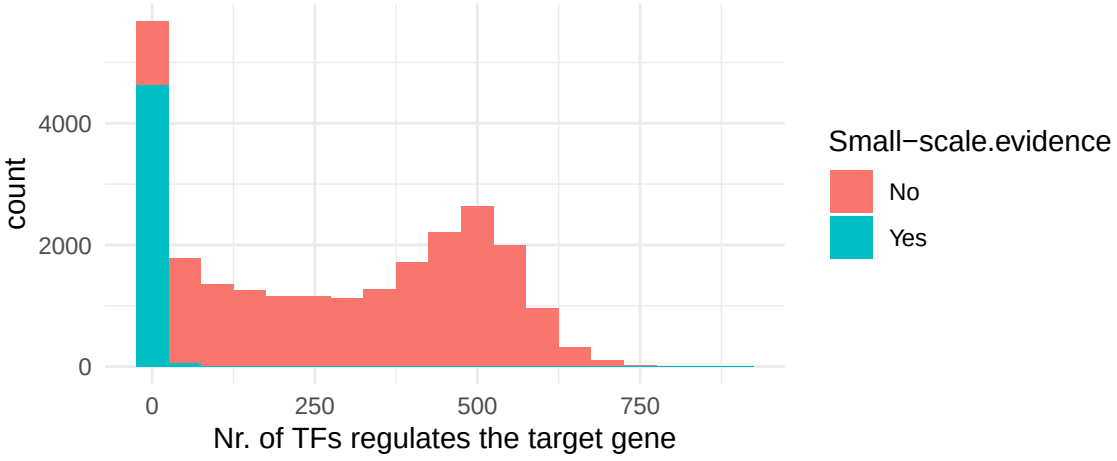
UniprotID.TF	UniprotID.Target	NCBI.GeneID.TF	NCBI.GeneID.Target	Name.TF	Name.Target	Detection.method
P03372	P17275	2099	3726	ESR1	JUNB	chromatin immunoprecipitation assay;inferred by curator

PubmedID	Organism	Source.database	Small-scale.evidence
19148141;29087512;29126285;27924024	Homo sapiens	GTRD;ReMap;TRRUST	Yes
17202159;12839938;29087512;27924024	Homo sapiens	GTRD;TRED;TRRUST	Yes
29126285;27924024;17202159	Homo sapiens	GTRD;ReMap;TRED	Yes
29087512;20565746	Homo sapiens	TRRUST	Yes
29126285;18971253;27924024;11477071;17202159;29087512	Homo sapiens	GTRD;PAZAR;ReMap;TRED;TRRUST	Yes

3 Checking the number of target genes each transcription factor has

Let's see the distribution of number of target genes among transcription factors.

```
Interaction_tab %>%
 group_by(UniprotID.Target, `Small-scale.evidence`) %>%
 summarise(`Nr. of TFs regulates the target gene` = n()) %>%
 arrange(desc(`Nr. of TFs regulates the target gene`)) %>%
  ggplot(aes(x = `Nr. of TFs regulates the target gene`,
             fill = `Small-scale.evidence`)) +
  geom_histogram(binwidth = 50) +
  theme_minimal()
```



4 Transcription factors having up to 50 target genes

To be able to plot all target genes of two transcription factors with readable names, we will narrow down our choices to interactions supported by **small-scale evidences**, and we will select from transcription factor having up-to **50 target genes**.

Therefore, we create a new interaction table containing interaction data supported by small-scale evidences. (Alternatively, this table can also be downloaded directly from the TFLink website.)

```
Interaction_tab_SS <- Interaction_tab %>%
  filter(`Small-scale.evidence` == "Yes")
```

Let's save the names of such transcription factors having up to 50 target genes to a `vector` variable.

```
TF_names <- Interaction_tab_SS %>%
 group_by(Name.TF) %>%
 summarise(`Nr. of target genes` = n()) %>%
 filter(`Nr. of target genes` < 50) %>%
 select(Name.TF) %>%
 pull()

# Checking the content
head(TF_names)
```

```
## [1] "AATF"   "ABL1"   "AHR"    "AIP"    "AIRE"   "ANKRD1"
```

Then, let's create a `list` variable containing all target genes of each transcription factors, to be able to find the ones regulating common target genes.

```
# Creating the list
TGs_per_TFs_list <- Interaction_tab_SS %>%
  filter(Name.TF %in% TF_names) %>%
  select(Name.TF, Name.Target) %>%
  group_by(Name.TF) %>%
  group_split() %>%
  lapply(., pull, var = Name.Target)

# Naming the elements
names(TGs_per_TFs_list) <- TF_names

# Checking the content
head(TGs_per_TFs_list, n = 3)
```

```
## $AATF
## [1] "MYC"    "TP53"   "BAX"    "CDKN1A" "KLKB1"  "KLK3"
##
## $ABL1
##  [1] "JUN"    "TP53"   "BAX"    "CSF1"   "CDKN1A" "BCL2"   "BCL6"   "PIM1"
##  [9] "FOXO3"  "CCND2"
##
## $AHR
##  [1] "AHRR"   "MYC"    "ABCG2"  "MFSD2A" "MT2A"   "IL1B"   "PCNA"   "FOS"
##  [9] "RFC3"   "CYP1A2" "CA9"    "UGT1A6" "UGT1A1" "CCNG2"  "IL13"   "CYP2B6"
## [17] "CYP1B1" "CYP1A1" "IL6"    "ARNT"   "CCND1"  "BRCA1"
```

Let's check which transcription factor pairs have the most common target genes.

```
# TF pairs in every combinations
TF_combination_list <- combn(names(TGs_per_TFs_list), 2, simplify = FALSE)

# Nr. of common genes of all TF pairs
Intersect_lengths <- list()
for(i in 1:length(TF_combination_list)){
   Intersect_lengths[i] <-
     intersect(TGs_per_TFs_list[[TF_combination_list[[i]][1]]],
               TGs_per_TFs_list[[TF_combination_list[[i]][2]]]) %>%
     length()
}
rm(i)

# Converting list to vector
Intersect_lengths <- Intersect_lengths %>%
  unlist()

# Adding names of TF pairs (separated by "_")
names(Intersect_lengths) <- sapply(TF_combination_list, paste, collapse = "_")

# Checking the maximum number of common genes
Intersect_lengths %>%
  sort() %>%
  tail()
```

```
##   NR1H2_NR1H3    FOSL2_JUND  DNMT1_DNMT3B SREBF1_SREBF2 TWIST1_TWIST2
##            16            17            19            19            25
## STAT5A_STAT5B
##            30
```

Let's create a `vector` containing the names of the transcription factor pair having the most common target genes.

```
TFs <- which.max(Intersect_lengths) %>%
  names() %>%
  str_split(pattern = "_") %>%
  unlist()

# Let's see
TFs
```

```
## [1] "STAT5A" "STAT5B"
```

STAT5A and **STAT5B** (*signal transducer and activator of transcription 5 A* and *B*) are two highly similar transcription factors. The **STAT5** proteins are involved in cytosolic signalling and they mediate the expression of specific genes (Nosaka et al. 1999).

5 Network visualization of the two transcription factors having the most common target genes

6 Using the `igraph` R package

Now we need more steps to create an *adjacency matrix* input for the `igraph` network plot.

Adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.

```
# Creating data.frame
# while excluding target genes with missing names ("-")
Wide_selected_Interaction_tab_SS <- Interaction_tab_SS %>%
  filter(Name.TF %in% TFs & Name.Target != "-") %>%
  select(Name.TF, Name.Target) %>%
  mutate(Value = 1) %>%
  pivot_wider(names_from = Name.Target, values_from = Value) %>%
  replace(is.na(.), 0) %>%
  data.frame()

# Adding row names
rownames(Wide_selected_Interaction_tab_SS) <- Wide_selected_Interaction_tab_SS$Name.TF
Wide_selected_Interaction_tab_SS$Name.TF <- NULL

# Converting to matrix
Wide_selected_Interaction_tab_SS <- Wide_selected_Interaction_tab_SS %>%
  as.matrix()

# We need a function for creating the (symmetrical) adjacency matrix
expand.matrix <- function(A){
  m <- nrow(A)
  n <- ncol(A)
  B <- matrix(0,nrow = m, ncol = m)
  C <- matrix(0,nrow = n, ncol = n)
  cbind(rbind(B,t(A)),rbind(A,C))
}

# Applying the expand.matrix function
Wide_selected_Interaction_tab_SS <- Wide_selected_Interaction_tab_SS %>%
  expand.matrix()

# Adding missing row names
rownames(Wide_selected_Interaction_tab_SS)[1:2] <- colnames(
  Wide_selected_Interaction_tab_SS)[1:2]

# Converting to symmetrical adjacency matrix to asymmetrical to be able to
# create a directed graph (by deleting values below diagonal)
Wide_selected_Interaction_tab_SS[lower.tri(Wide_selected_Interaction_tab_SS)] <- 0L
```

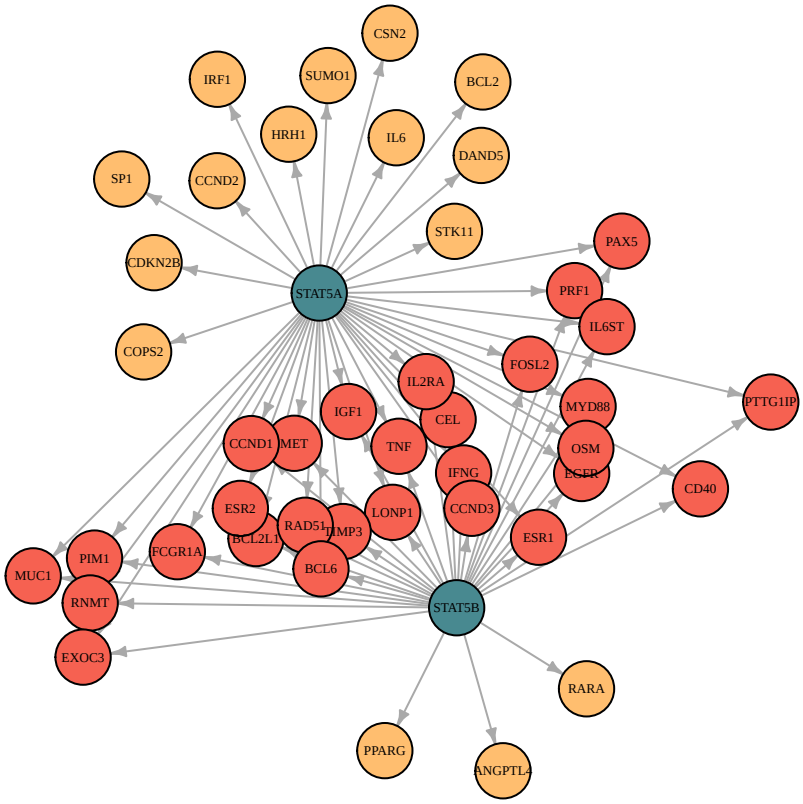
Let's see how does the resulted *adjacency matrix* looks like by checking its upper left part.

```
Wide_selected_Interaction_tab_SS[1:10, 1:10]
```

```
##         STAT5A STAT5B PIM1 PTTG1IP MET BCL2L1 IFNG FCGR1A CCND2 CEL
## STAT5A       0      0    1       1   1      1    1      1     1   1
## STAT5B       0      0    1       1   1      1    1      1     0   1
## PIM1         0      0    0       0   0      0    0      0     0   0
## PTTG1IP      0      0    0       0   0      0    0      0     0   0
## MET          0      0    0       0   0      0    0      0     0   0
## BCL2L1       0      0    0       0   0      0    0      0     0   0
## IFNG         0      0    0       0   0      0    0      0     0   0
## FCGR1A       0      0    0       0   0      0    0      0     0   0
## CCND2        0      0    0       0   0      0    0      0     0   0
## CEL          0      0    0       0   0      0    0      0     0   0
```

Let's create the `igraph` plot.

```
library(igraph)

# Creating the igraph variable
Network <- graph_from_adjacency_matrix(Wide_selected_Interaction_tab_SS,
                                       mode = "directed", diag = FALSE)
# Creating a vector containing three colours for vertices
Vcol <- case_when(colSums(Wide_selected_Interaction_tab_SS[1:2,]) == 0 ~ "#488990",
          colSums(Wide_selected_Interaction_tab_SS[1:2,]) == 1 ~ "#ffbe6f",
          colSums(Wide_selected_Interaction_tab_SS[1:2,]) == 2 ~ "#f66151")

# Reducing the margins of the plot
par(mar = rep(0.1, 4))

# Setting a random seed to reproduce the very same graph over and over again
set.seed(1)

# Creating the plot itself
plot.igraph(Network, layout = layout_with_dh,
            vertex.color = Vcol, vertex.label.color = "black",
            vertex.label.cex = 0.4, edge.arrow.size = 0.4)
```

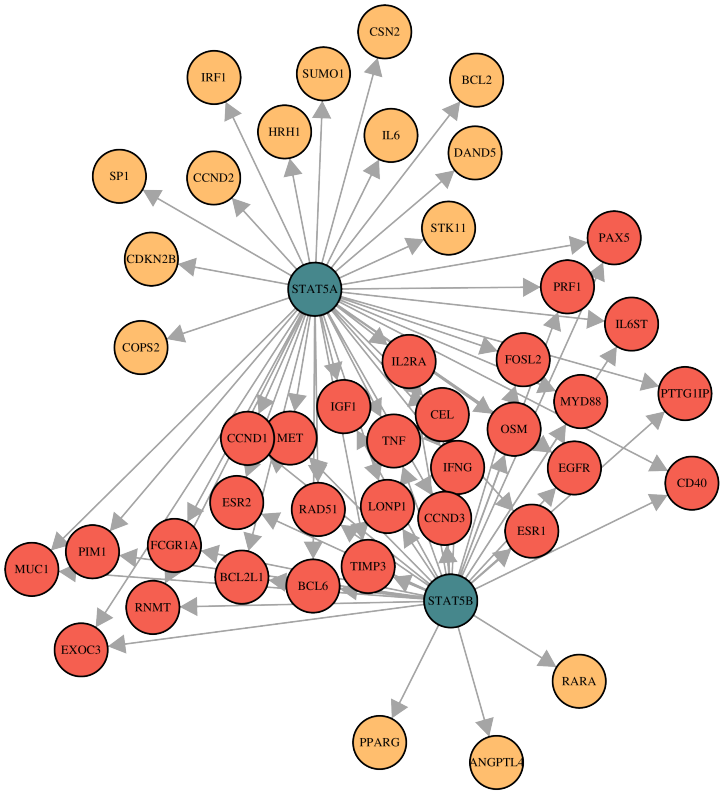


The two transcription factors are shown with green, their common target genes with red and their distinct target genes with yellow vertices.

To avoid such overlapping among vertices we can adjust the network by hand using the `tkplot` function of the `igraph` package.

```
set.seed(1)
tkplot(Network, layout = layout_with_dh,
       vertex.color = Vcol, vertex.label.color = "black",
       vertex.label.cex = 0.4)
```

After adjusting the common vertices by hand to avoid overlapping, we can produce a similar image.



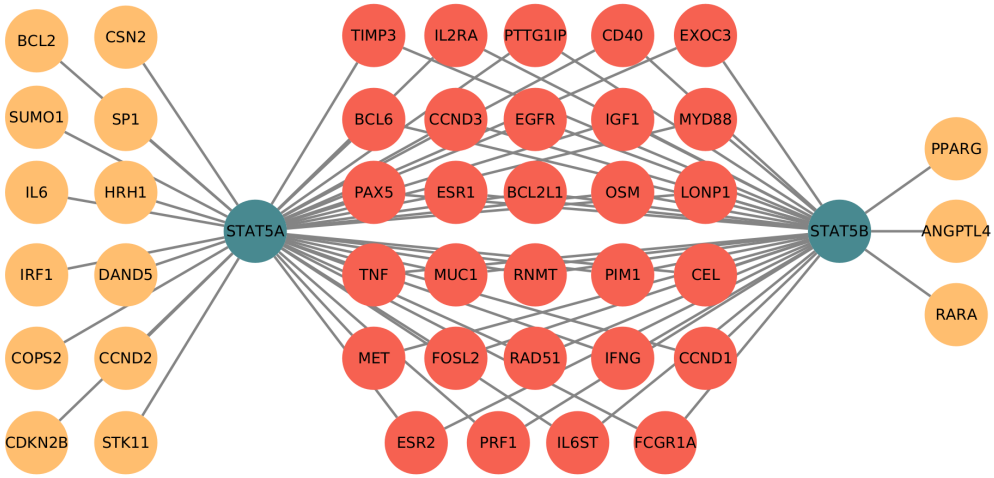
7 Using the Cytoscape software

Let's visualise the common and distinct target genes of the two transcription factors with the Cytoscape network visualization software by following these steps:

1. After downloading the *target gene interaction tables* of the **STAT5A** and **STAT5B** transcription factors from their TFLink entry pages, we open the Cytoscape software.
2. Then we import the *tsv* files as networks into the Cytoscape:
   - `File` → `Import` → `Network from file...` and select the downloaded file(s) (one file at once).
   - In the pop-up window click on the *column names* and *select* the following explanations for them:
     - *UniprotID.TF* → `Source Node Attribute`
     - *Name.TF* → `Source`
     - *UniprotID.Target* → `Target Node Attribute`
     - *NCBI.GeneID.Target* → `Target Node Attribute`
     - *Name.Target* → `Target`
     - then, all of the other columns will turn to `Edge Attribute`.
   - Click on the `OK` button.
3. Repeat the steps of point **2** with the other interaction table as well.
4. Select only the *small-scale* interactions from the imported networks:
   - On the left side in the `Network` panel click on the *network* what is needed to be filtered.
   - Go to the `Filter` panel, click on the `+` sign, select the `Edge: Small-scale evidence` option, and write "Yes" into the box.
   - Then, create a *net* network: Go to `File` → `Net Network` → `From Selected Nodes, Selected Edges`.

5. Repeat the steps of point **3** with the network of the other transcription factor.
6. Merge the filtered networks with `Tools` → `Merge` → `Network...`, and in the pop-up window put the two filtered networks (*targets_of_P42229.tsv(1)* and *targets_of_P51692.tsv(1)*) into the right side of the panel. Then click on the `Merge` button.
7. Go to the `Style` panel on the left side and set different colours, shapes, size parameters for the transcription factors and the target genes.
8. To create a meaningful layout, select the *nodes of the common target genes* in the *filtered network* at the `Network` panel and go to the `Layout` menu → `Grid layout` → `Selected Nodes only`. Then repeat this with the nodes of the distinct target genes. You can drag the nodes of the two transcription factors manually.

After all these steps we will end up a transcription factor - target gene interaction network like this:



8 Environment

In this *use case* the following software and package versions were applied:

- `R` version 4.1.3
- `tidyverse` version 1.3.1
- `data.table` version 1.14.2
- `igraph` version 1.2.11
- `Cytoscape` version 3.8.2

References

Nosaka, T, T Kawashima, K Misawa, K Ikuta, A L Mui, and T Kitamura. 1999. "Stat5 as a Molecular Regulator of Proliferation, Differentiation and Apoptosis in Hematopoietic Cells." *The EMBO Journal* 18 (17): 4754–65. https://doi.org/10.1093/emboj/18.17.4754.

TFLink use case 2

Similarities and differences in the function of target genes of a nuclear hormone receptor, **unc-55** in *Caenorhabditis elegans* and in human

1 The aim of this use case

Here we investigate the functional diversity of target genes of a *nuclear hormone receptor* transcription factor, the **unc-55** in *Caenorhabditis elegans* (*C. elegans*) and in human. We perform *Gene Ontology* (*GO*) overrepresentation analyses of the target genes in the two species in order to identify shared functional roles that likely represent the ancestral function of **unc-55**. Furthermore, this comparison will yield insights into the potentially divergent roles **unc-55** play in these two distant animal groups.

2 Input data

Let's download and read in to `R` the *target gene interaction table* of the **unc-55** transcription factor in *Caenorhabditis elegans* and the *target gene interaction table* of the **unc-55** ortholog **NR2F1** transcription factor in human from the TFLink website using the `read_tsv` command of the `tidyverse` package.

```
library(tidyverse)
# C.elegans unc-55
Interaction_tab_Ce <- read_tsv("https://tflink.net/protein/g5ecr9/")
# human NR2F1
Interaction_tab_Hs <- read_tsv("https://tflink.net/protein/p10589/")
```

For the overrepresentation analyses we will need *background* genes sets. Therefore, we download all *C. elegans* and human target genes available at the **Download** part of the TFLink website, and read it in with the `fread` command of the `data.table` package that reads huge tables effectively.

```
library(data.table)

# Creating a list for the Uniprot IDs of the background genes
Background_Uniprot <- list()

# Background genes for C. elegans
Background_Uniprot$Ce <- fread(
  "TFLink_Caenorhabditis_elegans_interactions_All_simpleFormat_v1.0.tsv") %>%
  select(UniprotID.Target) %>%
  unique() %>%
  pull()

# Background genes for human
Background_Uniprot$Hs <- fread(
  "TFLink_Homo_sapiens_interactions_All_simpleFormat_v1.0.tsv") %>%
  select(UniprotID.Target) %>%
```

```
  unique() %>%
  pull()
```

3 *Gene ontology* overrepresentation analysis of target genes

To have an idea about the type of *Biological processes* the **unc-55** and the **NR2F1** transcription factors regulate we perform a *Gene ontology* (*GO*) overrepresentation analysis of their target genes using the `clusterProfiler` `R` `Bioconductor` package.

3.1 Translating the target genes and the background genes to *Entrez IDs*

First, we need to translate the *Uniprot IDs* of the target genes to *Entrez IDs* using the `bitr` function of the `clusterProfiler` package. For this we need to install another two `R` `Bioconductor` packages containing various biological identifiers: the `org.Ce.eg.db` and `org.Hs.eg.db`.

```
library(clusterProfiler)

# Creating a list for the target genes
Genes_list <- list()

# Translating the target genes of C. elegans unc-55 to Entrez IDs
Genes_list$Ce_unc55 <- bitr(geneID = Interaction_tab_Ce$UniprotID.Target,
                            fromType = "UNIPROT",
                            toType = c("ENTREZID"),
                            OrgDb = "org.Ce.eg.db") %>%
  dplyr::select(ENTREZID) %>%
  unique() %>%
  pull()

# Translating the background genes of C. elegans to Entrez IDs
Genes_list$Ce_BG <- bitr(geneID = Background_Uniprot$Ce,
                            fromType = "UNIPROT",
                            toType = c("ENTREZID"),
                            OrgDb = "org.Ce.eg.db") %>%
  dplyr::select(ENTREZID) %>%
  unique() %>%
  pull()

# Translating the target genes of human NR2F1 to Entrez IDs
Genes_list$Hs_NR2F1 <- bitr(geneID = Interaction_tab_Hs$UniprotID.Target,
                            fromType = "UNIPROT",
                            toType = c("ENTREZID"),
                            OrgDb = "org.Hs.eg.db") %>%
  dplyr::select(ENTREZID) %>%
  unique() %>%
  pull()

# Translating the background genes of human to Entrez IDs
Genes_list$Hs_BG <- bitr(geneID = Background_Uniprot$Hs,
                            fromType = "UNIPROT",
                            toType = c("ENTREZID"),
```

```
                              OrgDb = "org.Hs.eg.db") %>%
  dplyr::select(ENTREZID) %>%
  unique() %>%
  pull()
```

3.2 The overrepresentation analyses

Let's use the `enrichGO` `clusterProfiler` function to see in which *Gene ontology* terms the target genes are overrepresented. We are not interested overrepresentation in too general or too specific *GO* terms, therefore we discard terms annotating more than 500 or less than 10 genes.

Be avare that it may take a few seconds to calculate the overrepresentation analyses.

```
# Creating a list for GO overrepresentation results
GO_results <- list()

# Overrepresentation analysis of the targets of C. elegans unc-55
GO_results$Ce_unc55 <- enrichGO(gene = Genes_list$Ce_unc55,
                                universe = Genes_list$Ce_BG,
                                OrgDb = "org.Ce.eg.db",
                                ont = "ALL",
                                minGSSize = 10,
                                maxGSSize = 500,
                                pAdjustMethod = "bonferroni",
                                pvalueCutoff = 0.01,
                                qvalueCutoff = 0.01,
                                readable = TRUE)

# Overrepresentation analysis of the targets of human NR2F1
GO_results$Hs_NR2F1 <- enrichGO(gene = Genes_list$Hs_NR2F1,
                                universe = Genes_list$Hs_BG,
                                OrgDb = "org.Hs.eg.db",
                                ont = "ALL",
                                minGSSize = 10,
                                maxGSSize = 500,
                                pAdjustMethod = "bonferroni",
                                pvalueCutoff = 0.01,
                                qvalueCutoff = 0.01,
                                readable = TRUE)
```

Let's see the results of the *GO* overrepresentation analyses by checking the number of *GO* terms each set of target genes were enriched in.

```
# Nr. of GO terms for target genes of C. elegans unc-55
GO_results$Ce_unc55@result %>%
  nrow()
```

```
## [1] 318
```

```
# Nr. of GO terms for target genes of human NR2F1
GO_results$Hs_NR2F1@result %>%
  nrow()
```

```
## [1] 201
```

3.3 Summarization of the *Gene ontology* overrepresentation results

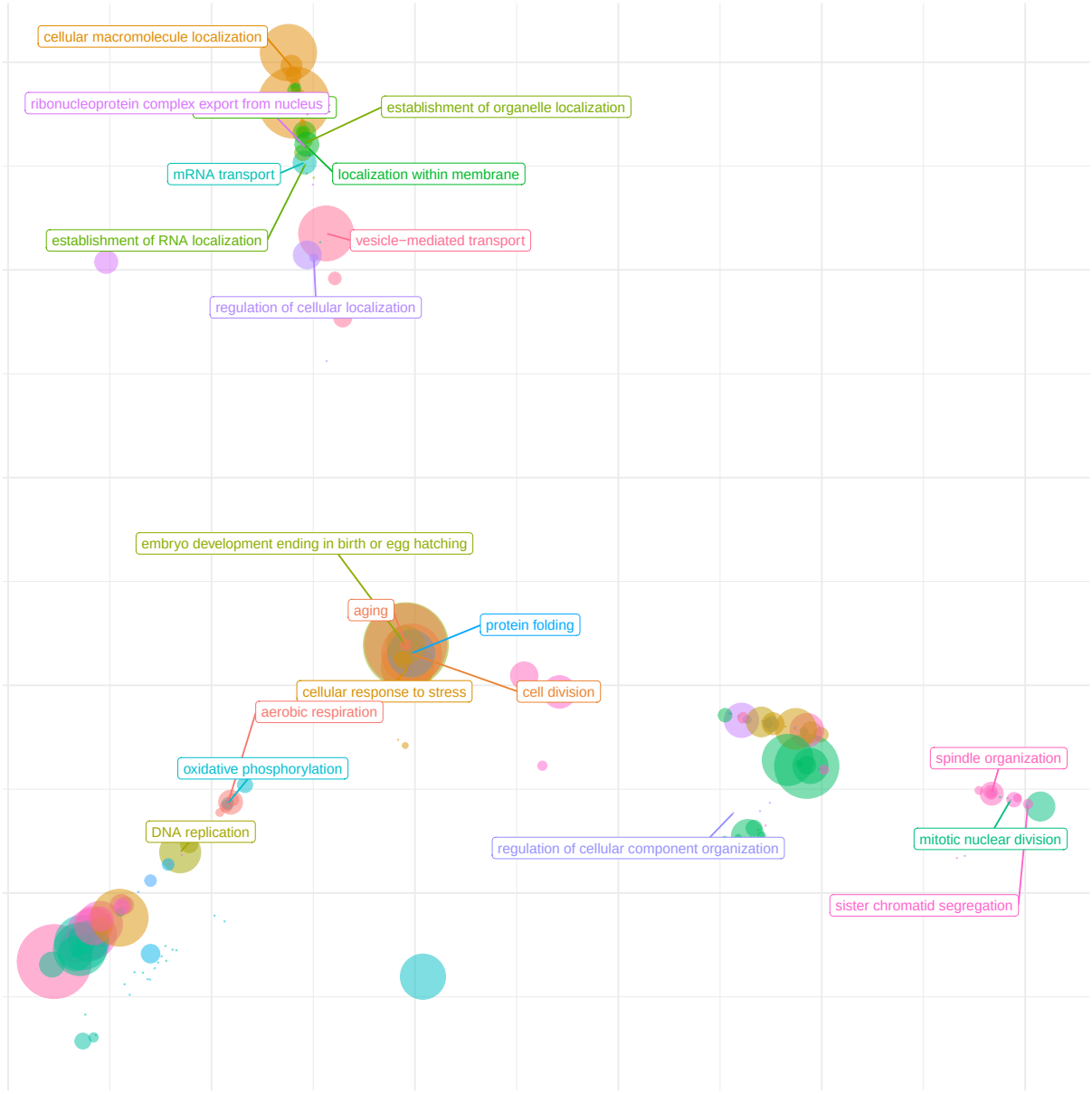
As we see, there are too many *GO* terms each set of target genes were enriched in. Therefore, we perform a following step and summarize the results by removing redundant *GO* terms. For this we use the `rrvgo` `R Bioconductor` package. Here we will summarise the *Biological process GO* terms only.

Be avare that it may take a few seconds to calculate these summaries.

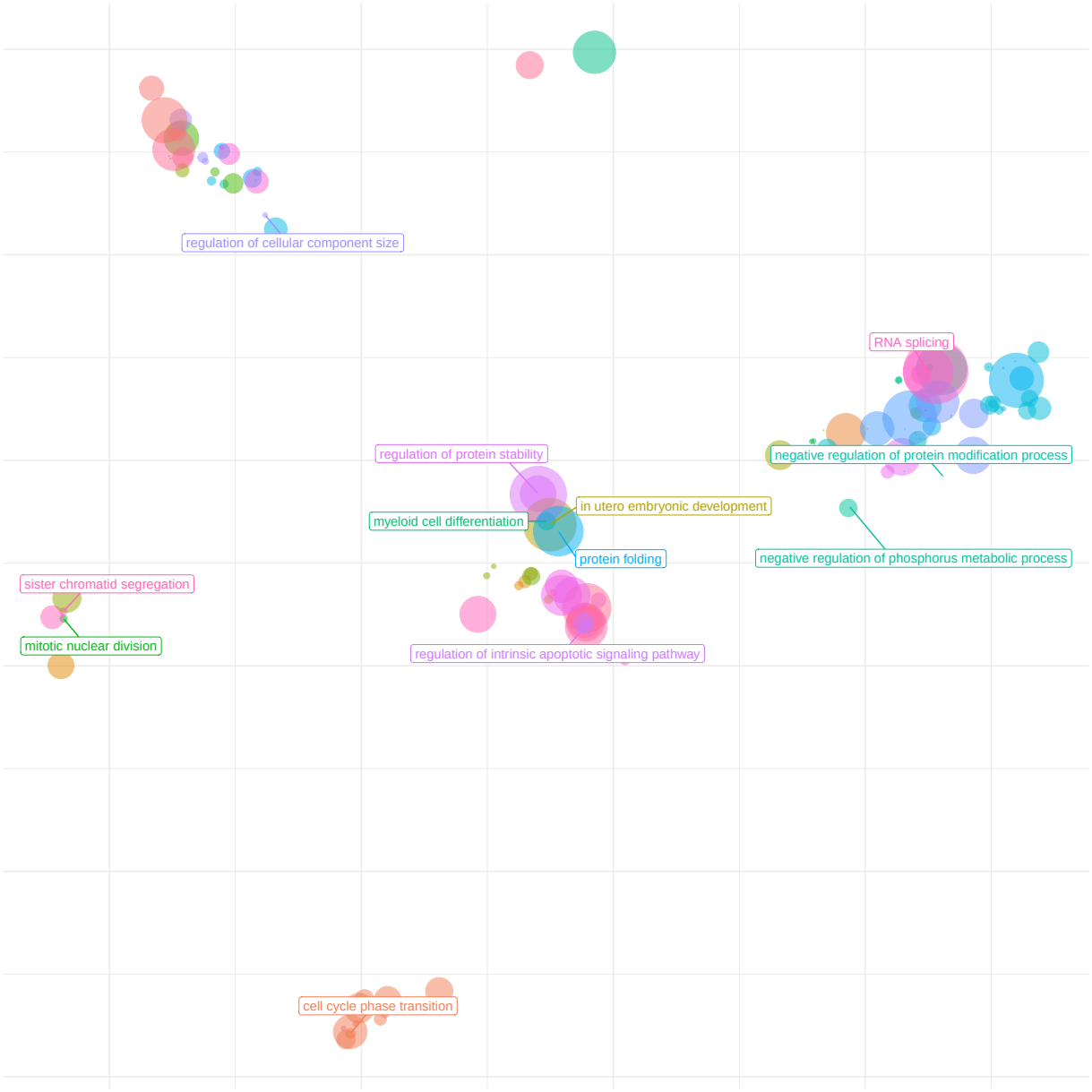
```
library(rrvgo)

# Creating a list for GO summarisation results
GO_sum <- list()

# Summarising the GO results of the targets of the C. elegans unc-55
GO_sum$Ce_simMatrix <- calculateSimMatrix(GO_results$Ce_unc55@result$ID,
                                          orgdb = "org.Ce.eg.db",
                                          ont = "BP",
                                          method = "Rel")
GO_sum$Ce_scores <- -log10(GO_results$Ce_unc55@result$qvalue) %>%
  setNames(., GO_results$Ce_unc55@result$ID)
GO_sum$Ce_reducedTerms <- reduceSimMatrix(simMatrix = GO_sum$Ce_simMatrix,
                                          scores = GO_sum$Ce_scores,
                                          threshold = 0.5,
                                          orgdb="org.Ce.eg.db")

# Summarising the GO results of the targets of the human NR2F1
GO_sum$Hs_simMatrix <- calculateSimMatrix(GO_results$Hs_NR2F1@result$ID,
                                          orgdb = "org.Hs.eg.db",
                                          ont = "BP",
                                          method = "Rel")
GO_sum$Hs_scores <- -log10(GO_results$Hs_NR2F1@result$qvalue) %>%
  setNames(., GO_results$Hs_NR2F1@result$ID)
GO_sum$Hs_reducedTerms <- reduceSimMatrix(simMatrix = GO_sum$Hs_simMatrix,
                                          scores = GO_sum$Hs_scores,
                                          threshold = 0.5,
                                          orgdb="org.Hs.eg.db")
```

Let's plot the main *GO* results and compare them.

```
# C. elegans unc-55
scatterPlot(GO_sum$Ce_simMatrix, GO_sum$Ce_reducedTerms)
```

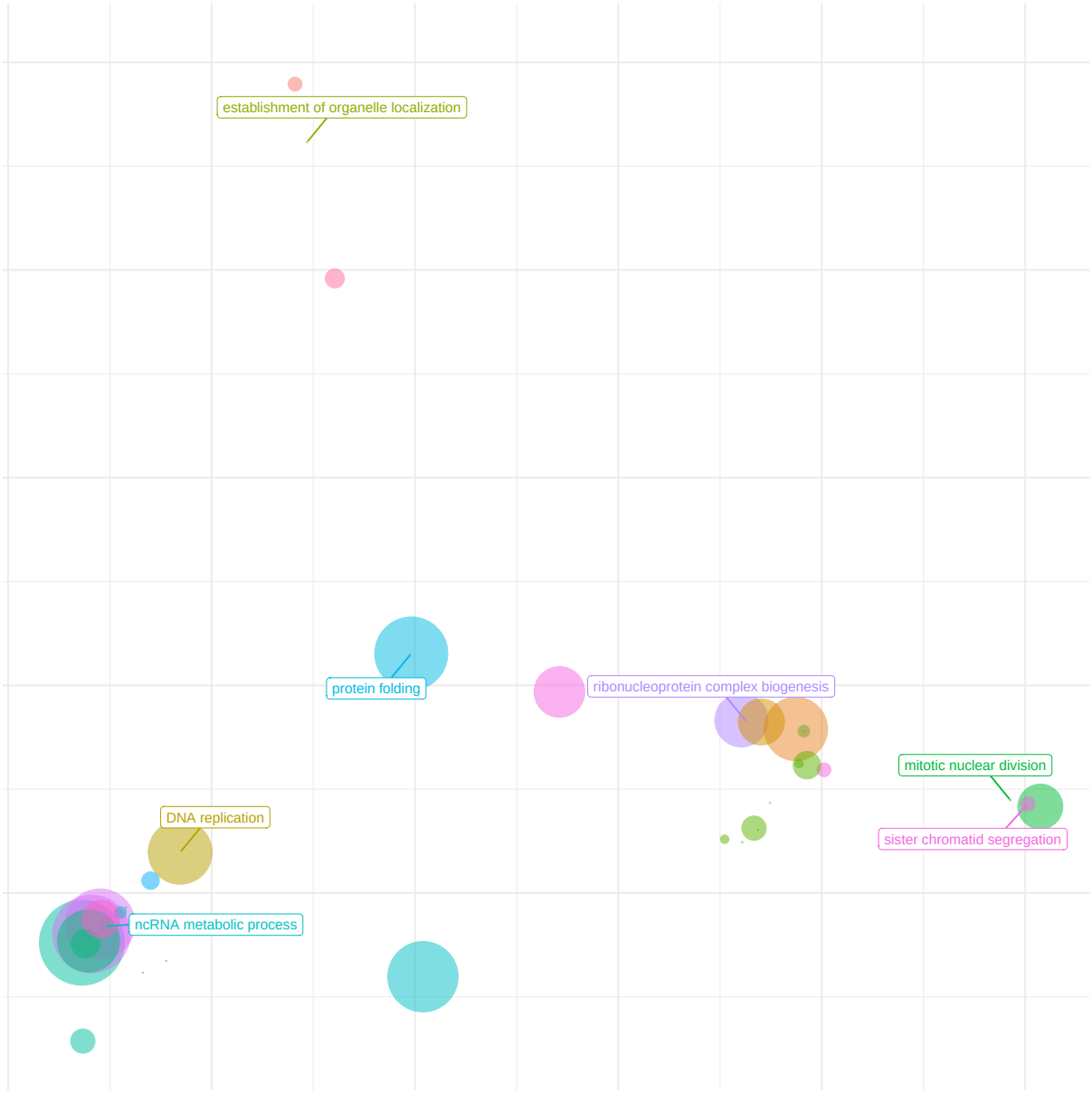



```
# human NR2F1
scatterPlot(GO_sum$Hs_simMatrix, GO_sum$Hs_reducedTerms, labelSize = 4)
```



Let's plot the main *GO* results that appeared to be common in the overrepresentation an summarisation analyses of the target genes of the *C. elegans* **unc-55** and the human **NR2F1** transcription factors.

```
# Filtering and plotting the common GO results
GO_sum$Ce_reducedTerms %>%
  filter(go %in% GO_sum$Hs_reducedTerms$go) %>%
  scatterPlot(GO_sum$Ce_simMatrix, .)
```



As we can see there are several conserved functions, and even more that diverged during the evolution the this transcription factor.

4 Environment

In this *use case* the following software and package versions were applied:

- `R` version 4.1.3
- `tidyverse` version 1.3.1
- `data.table` version 1.14.2
- `clusterProfiler` version 4.2.2
- `org.Ce.eg.db` version 3.14.0

- `org.Hs.eg.db` version 3.14.0
- `rrvgo` version 1.6.0

TFLink use case 3

Investigating the binding sites of the EGR1 transcription factor

1 The aim of this use case

Here we investigate the binding sites of the **EGR1** transcription factor. After converting the TFLink binding site table to **BED** and **BAM** files, we calculate the "coverage" to reveal the strength of evidence (number of supporting experiments) for each binding site. Then we plot the binding sites on the human chromosomes, indicating the number of supporting evidences each binding site has. Finally, we investigate specific binding sites using the IGV genome viewer tool.

2 The input data

Let's download the *binding site table* of human **EGR1** transcription factor from the TFLink website, and read it in to `R` with the `read_tsv` command of the `tidyverse` package.

```
library(tidyverse)
EGR1_BS_tab <- read_tsv("TFLink_P18146_TFBS_annot_v1.0.tsv")
```

Let's see the first few rows of the binding site table.

```
EGR1_BS_tab %>%
  slice(1:5)
```

Head of the binding site table (continued below)

TFLinkID	UniprotID.TF	Name.TF	Organism	Assembly	Chromosome	Start	End
TFLinkSS09532447	P18146	EGR1	Homo sapiens	hg38	chr14	59361534	59361673
TFLinkSS09543853	P18146	EGR1	Homo sapiens	hg38	chr14	37596110	37596200
TFLinkSS09543856	P18146	EGR1	Homo sapiens	hg38	chr14	96264096	96264186
TFLinkSS09543857	P18146	EGR1	Homo sapiens	hg38	chr14	37594940	37595000
TFLinkSS09543858	P18146	EGR1	Homo sapiens	hg38	chr14	37595930	37595990

Table continues below

Strand	Genome.browser	Detection.method	PubmedID	Source.database
+	https://genome.ucsc.edu/cgi-bin/hgTracks?db=hg38&position=chr14:59361534-59361673	inferred by curator	18971253	ORegAnno
+	https://genome.ucsc.edu/cgi-bin/hgTracks?db=hg38&position=chr14:37596110-37596200	inferred by curator	18971253	ORegAnno
+	https://genome.ucsc.edu/cgi-bin/hgTracks?db=hg38&position=chr14:96264096-96264186	inferred by curator	18971253	ORegAnno

Strand	Genome.browser	Detection.method	PubmedID	Source.database
+	https://genome.ucsc.edu/cgi-bin/hgTracks?db=hg38&position=chr14:37594940-37595000	inferred by curator	18971253	ORegAnno
+	https://genome.ucsc.edu/cgi-bin/hgTracks?db=hg38&position=chr14:37595930-37595990	inferred by curator	18971253	ORegAnno

Small-scale.evidence	Number.of.TFBS.overlaps	TFBS.overlaps
Yes	0	-
Yes	0	-
Yes	0	-
Yes	0	-
Yes	0	-

Let's check the number of binding sites of the **EGR1** transcription factor.

```
EGR1_BS_tab %>%
  nrow()
```

[1] 156418

3 Converting the binding site table to BED and BAM files

Let's check the names of the chromosomes used in the binding site table.

```
EGR1_BS_tab %>%
  group_by(Chromosome) %>%
  group_keys() %>%
  pull()
```

chr1, chr10, chr11, chr12, chr13, chr14, chr14_GL000009v2_random, chr14_GL000225v1_random, chr15, chr16, chr17, chr18, chr19, chr2, chr2_KI270894v1_alt, chr20, chr21, chr22, chr22_KI270735v1_random, chr22_KI270879v1_alt, chr3, chr4, chr4_GL000008v2_random, chr5, chr6, chr7, chr7_KI270803v1_alt, chr8, chr8_KI270821v1_alt, chr9, chrM, chrUn_GL000219v1, chrUn_KI270442v1, chrUn_KI270742v1, chrX, chrY

3.1 Excluding non-canonical chromosomes and saving the BED files

If we want to check the "coverage" for the *canonical* human chromosomes only, we need to exclude all other contigs from the *EGR1_BS_tab* table.

```
# Keeping rows where the name of the chromosome doesn't include "_", and
# excluding the mitochondrial chromosome (chrM) as well.

# Creating a singe BED file containing the data about the FORWARD and REVERSE
# strands.
EGR1_BS_tab %>%
  filter(!grepl(x = Chromosome, pattern = "_|M")) %>%
  mutate(Name = "EGR1", Score = 1000) %>%
```

```
  select(Chromosome, Start, End, Name, Score, Strand) %>%
  arrange(Chromosome, Start, End, Strand) %>%
  write_tsv("EGR1_BS.bed", col_names = FALSE)

# Creating a BED file containing the data about the FORWARD strand only.
EGR1_BS_tab %>%
  filter(!grepl(x = Chromosome, pattern = "_|M") & Strand == "+") %>%
  mutate(Name = "EGR1", Score = 1000) %>%
  select(Chromosome, Start, End, Name, Score, Strand) %>%
  arrange(Chromosome, Start, End, Strand) %>%
  write_tsv("EGR1_BS_forward.bed", col_names = FALSE)

# Creating a BED file containing the data about the REVERSE strand only.
EGR1_BS_tab %>%
  filter(!grepl(x = Chromosome, pattern = "_|M") & Strand == "-") %>%
  mutate(Name = "EGR1", Score = 1000) %>%
  select(Chromosome, Start, End, Name, Score, Strand) %>%
  arrange(Chromosome, Start, End, Strand) %>%
  write_tsv("EGR1_BS_reverse.bed", col_names = FALSE)
```

Let's convert the **BED** file to **BAM** using the `bedtobam` command of the bedtools software package.

For this we need a text file, called: **chrom.sizes** containing the name and the size (in base pairs) of the chromosomes. We can download this for the human genome version *hg38* from here.

Let's read the **chrom.sizes** file into `R`, exclude the non-canonical chromosomes, and write out a new **chrom.sizes** file.

```
chrom_sizes_tab <- read_tsv(
  "https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.chrom.sizes",
  col_names = c("Chromosome", "Size"))
chrom_sizes_tab %>%
  write_tsv("chrom.sizes", col_names = FALSE)
```

Now we can run the `bedToBam` command in the **terminal** to create the **BAM** files.

```
# Forward strand BAM
$ bedToBam -i EGR1_BS_forward.bed -g chrom.sizes > EGR1_BS_forward.bam
# Reverse strand BAM
$ bedToBam -i EGR1_BS_reverse.bed -g chrom.sizes > EGR1_BS_reverse.bam
```

4 Calculating and investigating the "coverage"

Let's calculate the number of evidences each binding site has using the samtools software.

```
# Forward strand BAM
$ samtools depth EGR1_BS_forward.bam > EGR1_BS_forward.coverage
# Reverse strand BAM
$ samtools depth EGR1_BS_reverse.bam > EGR1_BS_reverse.coverage
```

4.1 Reading in the "coverage" files to R.

Now we use the `fread` function of the `data.table` R package that reads huge tables effectively.

```
library(data.table)

# Reading the coverage file of the FORWARD strand
EGR1_BS_cover <- fread("EGR1_BS_forward.coverage")

# Reading the coverage file of the REVERSE strand
EGR1_BS_cover <- fread("EGR1_BS_reverse.coverage") %>%
  # Indicating that it is the coverage of the reverse strand by multiplying
  # the last column (containing information about the depth) with -1
  mutate(V3 = V3*-1) %>%
  # Binding the rows of the reverse strand variable to the previous variable
  # containing the coverage of the forward strand
  bind_rows(., EGR1_BS_cover) %>%
  # Renaming the columns according to their content
  rename(Chromosome = V1, Locus = V2, Depth = V3) %>%
  # Arranging the rows by chromosome and locus (position)
  arrange("Chromosome", "Locus")
```

Let's check the "coverage" (the number of evidences each binding sites chromosomal position has).

```
EGR1_BS_cover %>%
  group_by(Depth) %>%
  summarise(n = n()) %>%
  # indicating the strand
  mutate(Strand = case_when(Depth < 0 ~ "-",
                            Depth > 0 ~ "+"),
         Depth = abs(Depth)) %>%
  arrange(desc(Depth))
```

Number of evidences

Depth	n	Strand
12	1	-
11	2	-
10	4	-
9	9	+
8	18	-
8	297	+
7	1,648	-
7	1,980	+
6	3,126	-
6	3,370	+
5	3,164	-
5	4,949	+
4	6,346	-
4	20,330	+
3	18,586	-
3	132,572	+

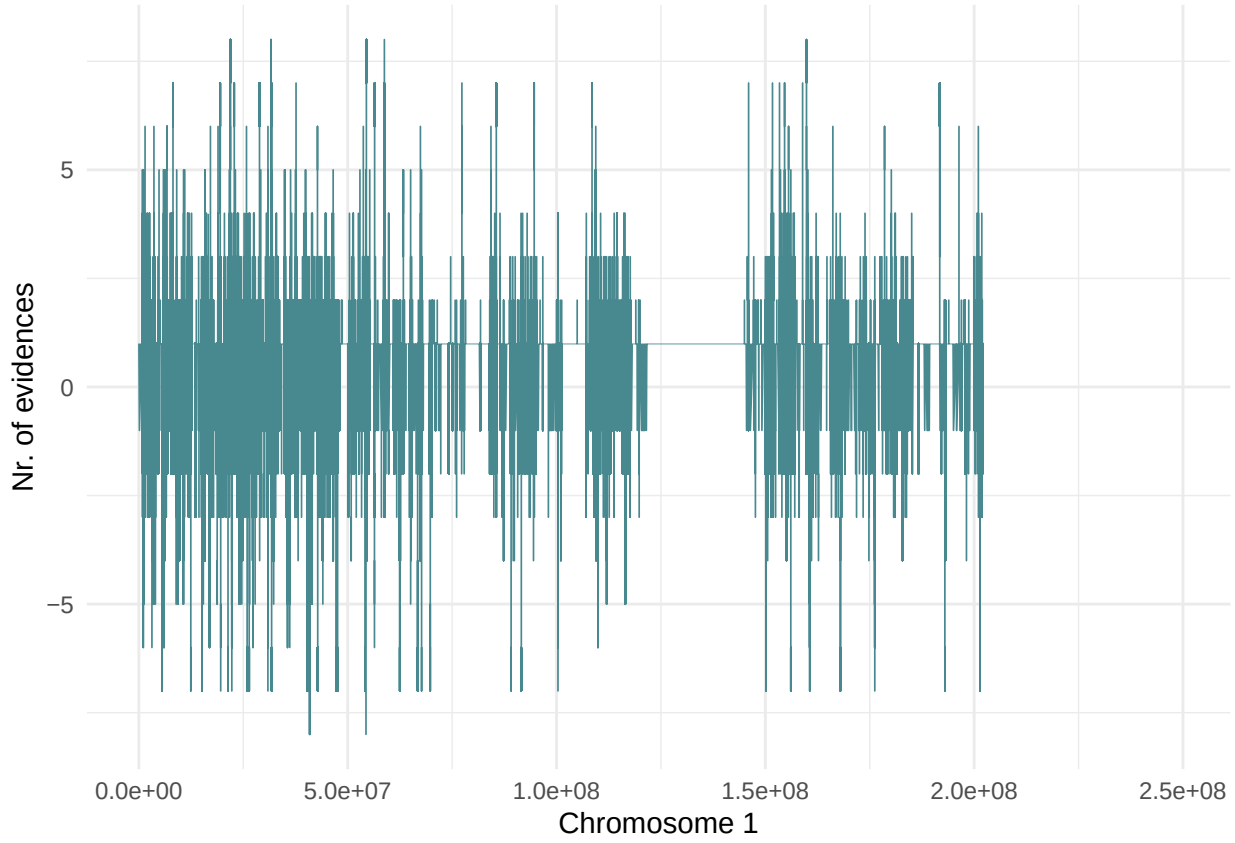
Depth	n	Strand
2	81,920	-
2	669,525	+
1	366,387	-
1	41,819,023	+

4.2 Plotting the number of evidences of binding sites on chromosome 1

On the figure, the binding sites on the *forward* and *reverse* strands appear as *positive* and *negative* values respectively.

Be avare that it may take a few seconds to render this plot.

```
EGR1_BS_cover %>%
  filter(Chromosome == "chr1") %>%
  ggplot(aes(x = Locus, y = Depth, group = Chromosome)) +
  geom_line(size = 0.3, color = "#488990") +
  xlab("Chromosome 1") +
  ylab("Nr. of evidences") +
  theme_minimal()
```

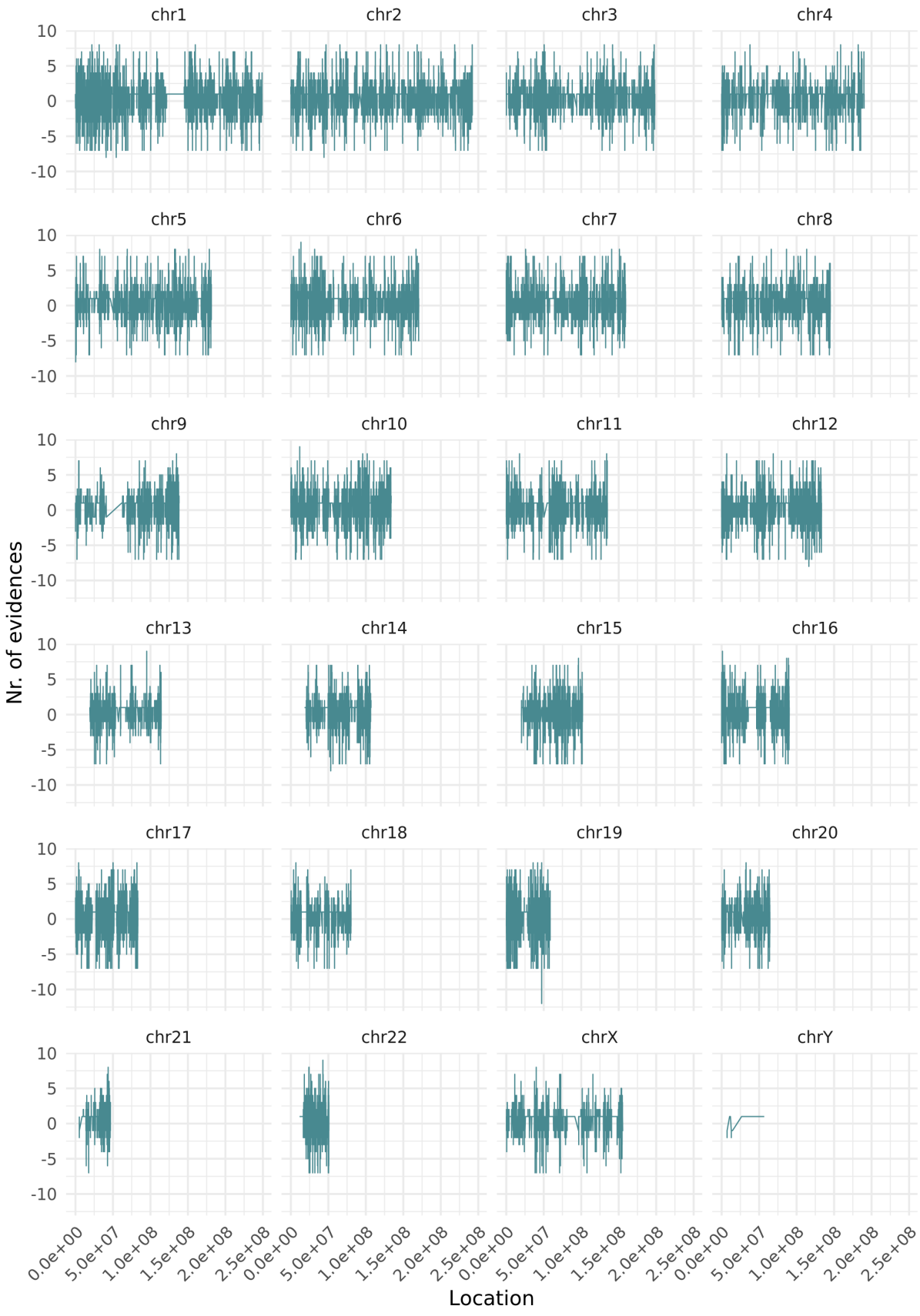


As you can see, there are no binding sites at the middle of the chromosome, where the centromere is located.

4.3 Plotting the number of evidences of binding sites on all chromosomes

Be avare that it may take a several minutes to render this plot and your computer even can run out of memory.

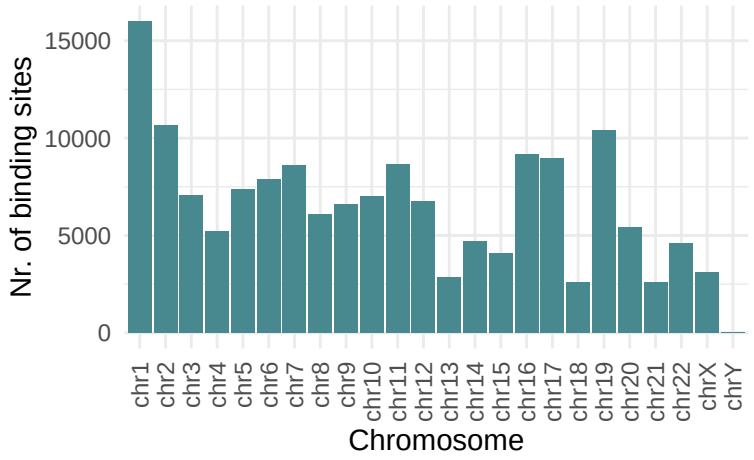
```
p <- EGR1_BS_cover %>%
  ggplot(aes(x = Locus, y = Depth)) +
  geom_line(size = 0.3, color = "#488990") +
  xlab("Location") +
  ylab("Nr. of evidences")
p +
  facet_wrap(. ~ Chromosome, ncol = 4) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust=1))
```


chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrX
chrY
Nr. of evidences
10
5
0
-5
-10
0.0e+00
5.0e+07
1.0e+08
1.5e+08
2.0e+08
2.5e+08
Location

5 How many EGR1 binding sites can be found on each chromosome?

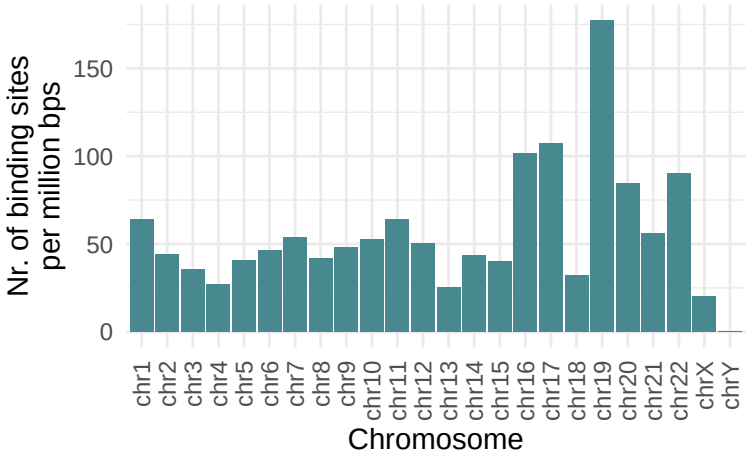
Let's use the downloaded *binding site table* again. We exclude non-canonical chromosomes.

```
# New variable with canonical chromosomes (as factor)
EGR1_BS_tab_canChro <- EGR1_BS_tab %>%
  filter(!grepl(x = Chromosome, pattern = "_|M")) %>%
  mutate(Chromosome = factor(Chromosome, levels = paste0("chr", c(1:22, "X", "Y"))))

# Barplot
EGR1_BS_tab_canChro %>%
  group_by(Chromosome) %>%
  summarise(n = n()) %>%
  ggplot(aes(x = Chromosome, y = n)) +
  geom_bar(stat="identity", fill = "#488990") +
  ylab("Nr. of binding sites") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



Chromosome 1 has the most binding site but what happens if we normalize the counts by the length of the chromosome (using the `chrom_sizes_tab` variable).

```
# Normalization and barplotting
EGR1_BS_tab_canChro %>%
  group_by(Chromosome) %>%
  summarise(n = n()) %>%
  left_join(., chrom_sizes_tab) %>%
  mutate(Chromosome = factor(Chromosome,
                             levels = paste0("chr", c(1:22, "X", "Y")))) %>%
  mutate(`Nr. of binding sites\nper million bps` = n / (Size / 1000000)) %>%
  ggplot(aes(x = Chromosome, y = `Nr. of binding sites\nper million bps`)) +
  geom_bar(stat="identity", fill = "#488990") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



Now we see that actually *chromosome 19* has the most binding sites of **ERG1**.

In the next chapter we check around which genes these binding sites are located.

6 Visually investigating the binding sites with the IGV genome viewer software

Now we have two choices to investigate the binding sites with the IGV genome viewer:

1. Reading in the transcription factor binding site **BAM** file (created above) as a "*read*" track.
2. Reading in the transcription factor binding site **GFF3** file (downloaded from the corresponding TFLink entry page) as an "*annotation*" track.

Let's do both!

First we need to index the **BAM** file using the samtools software.

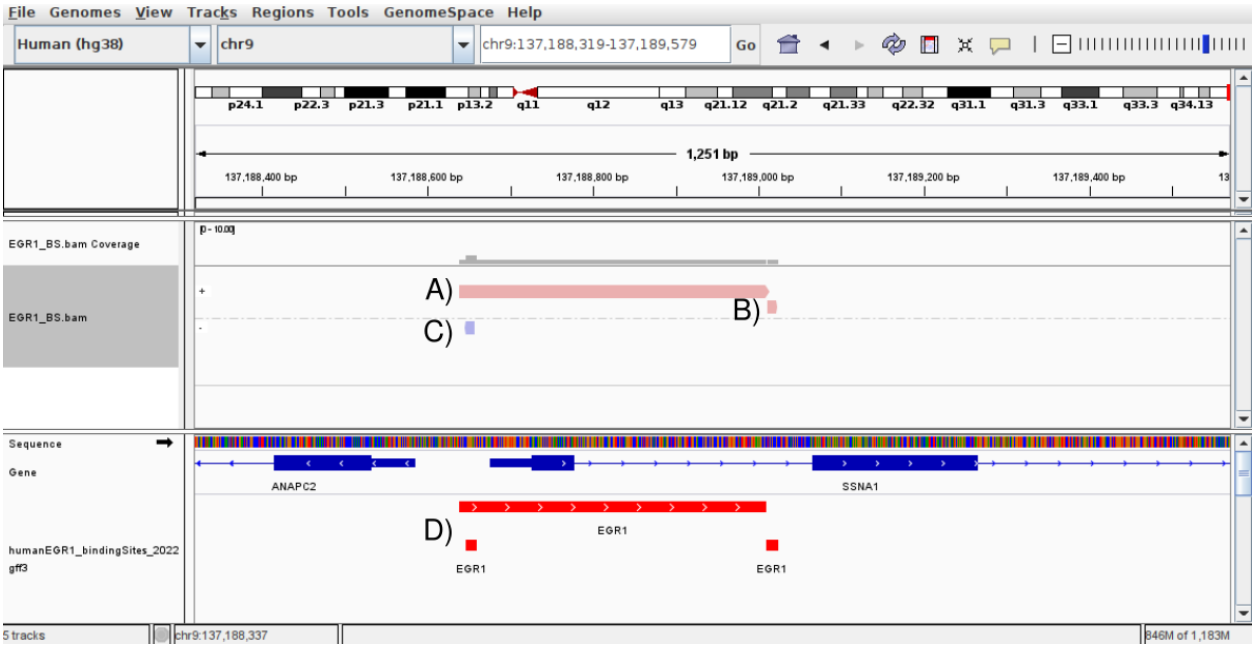
```
$ samtools index EGR1_BS.bam
```

With the above command a new file, named `EGR1_BS.bam.bai` was created.

After opening the IGV genome viewer:

1. let's set the human genome to *hg38* (open the drop down menu at the top left panel → `More`),
2. read the BAM file as "read" track (`File` → `Load from file...`),
3. color the "reads" by read strand (right click on the track in the left panel, `Color alignments by` → `read strand`),
4. group the "reads" by read strand (right click on the track in the left panel, `Group alignments by` → `read strand`),
5. read the downloaded GFF3 file as annotation track (`File` → `Load from file...`),
6. expand the annotation track (right click on the track in the left panel, `Expand`),
7. change the color of the annotation track to red (right click on the track in the left panel, `Change track color`),
8. zoom in to the region 137,188,319-137,189,579 on chromosome 9 (inserting `chr9:137,188,319-137,189,579` to the searching bar at top center, `Go`), and

we see the figure below.



What we see at the "read" track area?

A) There is a longer (371bp) binding site on the forward strand (colored with light red) overlapping with the putative promoter region, the first exon and first intron, of gene **SSNA1**.
B) Also there is a shorter (13bp) binding site on the forward strand in the first intron of gene **SSNA1**.
C) There is a shorter (13bp) binding site on the reverse strand (colored with light blue) in the putative promoter region of gene **ANAPC2**.
D) At the "annotation" track area we see the similar picture but the orientation is showed by white arrows on the binding sites.

Just to double check these findings by using an independent source of information: we can download the *target gene interaction table* of the **EGR1** transcription factor from the TFLink website, and we can look up these two genes among the target genes. Then indeed, we will find both the **SSNA1** and the **ANAPC2** genes among the target genes. Both evidences were inferred by chromatin immunoprecipitation assays:

Quitting from lines 375-378 (TFLink_Use_Case_3_Binding_Sites_of_EGR1.Rmd) Error: '2_target_genes.tsv' does not exist in current working directory ('/home/barizona/Eszter/Kutatas/TF/use_cases_for_paper/UC3

7 Environment

In this *use case* the following software and package versions were applied:

- `R` version 4.1.3
- `tidyverse` version 1.3.1
- `igraph` version 1.2.11
- `Cytoscape` version 3.8.2