

Motif Identification and Validation

MotIV

Eloi Mercier*and Raphael Gottardo†

October 18, 2010

Contents

I	Licensing	3
II	Introduction	3
III	Quick View	3
0.1	Load MotIV package	3
0.2	Load the database	3
0.3	Load database scores	4
0.4	Read input PWM	4
0.5	Analysis	4
0.6	View results	4
0.7	Apply filters	5
IV	Step-by-step Guide	5
1	MotIV package	5
2	Database	6
3	Database Scores	6
4	Input Motifs	6
4.1	From a gadem object	6
4.2	From a PWM file	7
4.2.1	GADEM type	7
4.2.2	TRANSFAC type	7
4.3	Trim Input	7
5	MotIV Analysis	7
5.1	Summary	8

*eloi.mercier@ircm.qc.ca

†raphael.gottardo@ircm.qc.ca

6	Filters	8
6.1	SetFilter	8
6.2	Operators & and 	9
6.3	Filter	9
6.4	Split	9
6.5	Combine	9
7	Results	9
7.1	Logo	9
7.2	Alignment	10
7.3	Distribution	10
7.4	Distance	11
8	RangedData	12
8.1	Ranged Data	12
9	Saving and Exporting Results	12
9.1	motiv object	12
9.2	Into Transfac Type Files	13
9.3	Into a BED File	13
10	Miscellaneous	13
10.1	viewMotifs	13
10.2	names	13
10.3	similarity	13
10.4	select	14
V	Appendix	14
11	GSL Installation	14
VI	References	15

Part I

Licensing

MotIV is a free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. **Motiv** is based on the C++ functions of the STAMP algorithm [1] and it also use a modified version of the SeqLogo package [2]. Please cite the following papers if you use **MotIV** for publication :

S. Mahony, P.V. Benos "STAMP: a web tool for exploring DNA-binding motif similarities." Nucl Acids Res, (2007) 35:W253-258

S Mahony, PE Auron, PV Benos, "DNA familial binding profiles made easy: comparison of various motif alignment and clustering strategies", PLoS Computational Biology (2007) 3(3):e61

Part II

Introduction

One of the most challenging part of the molecular biology is to understand the genetic regulation mechanisms. That's why is it important to work on the identification of the regulatory sequences such as transcription factors. It's in general short sequences located upstream the transcription initiation factor and recruiting proteic complex. Furthermore, this factors are themselves regulate by other proteic complex forming 'module' and adding a new level of complexity to the understanding of the genetic regulation system [3]. This modules still are hard to detect because of the complexity of the current identification algorithms.

MotIV have been developed to facilitate the identification and the validation of transcription factors. The **MotIV** package contains a motifs matches algorithm which is the primary tool of the software as well as visualizing results functions. The **MotIV** package is fully compatible to exploit the **rGADEM** package results.

Therefore, **MotIV** can take different input as well as object of type **gadem** (provided by **rGADEM**) or a file containing PWMs in standard **GADEM** output or in Transfac format. We strongly recommend to use **rGADEM** object because it offers more information needed by some functions.

Part III

Quick View

0.1 Load **MotIV** package

```
library(MotIV)
path <- system.file(package = "MotIV")
```

0.2 Load the database

```
jaspar <- readPWMfile(paste(path, "/extdata/jaspar2010.txt", sep = ""))
```

0.3 Load database scores

```
jaspar.scores <- readDBScores(paste(path, "/extdata/jaspar2010_PCC_SWU.scores",  
sep = ""))
```

0.4 Read input PWM

```
example.motifs <- readPWMfile(paste(path, "/extdata/example_motifs.txt",  
sep = ""))
```

0.5 Analysis

```
example.jaspar <- motifMatch(inputPWM = example.motifs, align = "SWU",  
cc = "PCC", database = jaspar, DBscores = jaspar.scores, top = 5)
```

```
Ungapped Alignment  
Scores read  
Database read  
Motif matches : 5
```

0.6 View results

```
summary(example.jaspar)
```

```
Number of input motifs : 25  
Input motifs names : m1 m2 m3 m4 m5 m6 m7 m8 m9 m10 m11 m12 m13 m14 m15 m16 m17 m18 m19  
Number of matches per motif: 5  
Matches repartition :  
      SP1      Stat3      FEV      SPI1      NFE2L2      AP1  
      8      8      6      6      5      3  
      EBF1      ESR1      Foxd3 Hand1::Tcfe2a      Myf      NFATC2  
      3      3      3      3      3      3  
PPARG::RXRA      SPIB      ELK4      EWSR1-FLI1      FOXA1      FOXI1  
      3      3      2      2      2      2  
      Foxq1      INSM1      Klf4      NF-kappaB      NFKB1      NHLH1  
      2      2      2      2      2      2  
      NR4A2      PLAG1      Pax4      REL      RELA      RREB1  
      2      2      2      2      2      2  
      STAT1 Tal1::Gata1      Tcfcp2l1      znf143      Arnt::Ahr      CREB1  
      2      2      2      2      1      1  
      CTCF Ddit3::Cebpa      ELF5      Egr1      Esrrb      Evi1  
      1      1      1      1      1      1  
      Foxa2      HNF4A      IRF1      MEF2A      NFIC      PPARG  
      1      1      1      1      1      1  
      Pax5      Pax6      REST      RORA_1      RUNX1      RXRA::VDR  
      1      1      1      1      1      1  
      SOX10      SRY      TAL1::TCF3      TLX1::NFIC      ZEB1  
      1      1      1      1      1  
Arguments used :  
-metric name : PCC  
-alignment : SWU
```

```
viewAlignments(example.jaspar)[[1]]
```

```
      Pax4                               SP1
seq   "-----TYCTCCYNCCTCNNCCTCCCN--" "TYCTCCYNCCTCNNCCTCCCN"
match "RAAWAWWWNNMNNNNNNNNNNMNNNNNYMC" "-----CCCCNCCCC--"
evaluate "2.9546e-03"                    "5.4587e-03"

      NFE2L2          PPARG::RXRA          PLAG1
seq   "TYCTCCYNCCTCNNCCTCCCN" "NGGGAGGNGAGGNRGGAGRA" "NGGGAGGNGAGGNRGGAGRA"
match "-----RTGACWNAGCA-----" "-NNRGGNCAAAGGKCA-----" "--GGGGCCNAAGGGGG-----"
evaluate "5.7277e-03"          "1.0955e-02"          "1.4671e-02"
```

```
plot(example.jaspar, ncol = 2, top = 5)
```

0.7 Apply filters

```
foxa1.filter <- setFilter(tfname = "FOXA")
ap1.filter <- setFilter(tfname = "AP1")
foxa1.ap1.filter <- foxa1.filter | ap1.filter
example.filter <- filter(example.jaspar, foxa1.ap1.filter, exact = F)
summary(example.filter)
```

```
Number of input motifs : 5
Input motifs names : m4 m7 m8 m18 m25
Number of matches per motif: 5
Matches repartition :
AP1 EWSR1-FLI1      FOXA1      Foxd3      NFE2L2      SP1      SPI1
  3      2      2      2      2      2      2
SPIB      Evi1      FEV      FOXI1      Foxq1      NFATC2      PPARG
  2      1      1      1      1      1      1
SRY TLX1::NFIC
  1      1
Arguments used :
-metric name : PCC
-alignment : SWU
```

```
plot(example.filter, ncol = 2, top = 5)
```

Part IV

Step-by-step Guide

1 MotIV package

To load the MotIV package, you should use this command line:

```
library(MotIV)
```

2 Database

First step is to load the database that you will use into the R environment. It could be a general database (JASPAR, TRANSFAC,...) [4] [5] or you can create your own one. Only Transfac file format are supported currently but other formats will be available soon.

To load the database, use the `readPWMfile` function :

```
jaspar <- readPWMfile(paste(path, "/extdata/jaspar2010.txt", sep = ""))
```

Note that the JASPAR is load by default when loading `MotIV`.

It returns a list of matrix corresponding to the database PWMs. For more information about the Transfac file format, please refer to <http://www.benoslab.pitt.edu/stamp/help.html>.

3 Database Scores

A database scores file is needed to compute E-value. Scores depend of the metric name and the alignment type given. Scores reflect the bias of the database used. To create a new database scores file, you should use the `generateDBScores` function. This function need a PWMs list as input, a metric, an alignment type and the number of random PWM to generate (see `?generateDBScores` for details). You have to use the same parameters for the entire analysis.

```
jaspar.scores <- generateDBScores(inputDB = jaspar, cc = "PCC",  
  align = "SWU", nRand = 1000)
```

WARNING : Because of each matrix is compared to each other, computing time is exponential. You should be aware of this fact before provided a high `nRand`. 5000 is a good time/accurate rate choice. (~30min)

To avoid wasted time, you can save the database score calculated for next similar analysis by typing :

```
writeDBScores(jaspar.scores, paste(path, "/extdata/jaspar_PCC_SWU.scores",  
  sep = ""))
```

For the following analysis, you will need to load the scores file by typing :

```
jaspar.scores <- readDBScores(paste(path, "/extdata/jaspar2010_PCC_SWU.scores",  
  sep = ""))
```

Remember that scores are associated to a specific database, metric and alignment type. By default, `jaspar.scores` is load with `MotIV`.

4 Input Motifs

Now that you have construct the database and the database scores, you have to load the PWM motifs you want to analyze. There are different ways to do it depending of the kind of data you have.

4.1 From a `gadem` object

`MotIV` software is designed to extend the features of the `rGADEM` package. Thus, you can use the object returned by a previous analysis with the `rGADEM` package. You need to load the `gadem` object load in your current R session. Load the motifs PWMs contained in an object called "gadem" by typing :

```
load(paste(path, "/data/FOXA1_rGADEM.rda", sep = ""))  
motifs <- getPWM(gadem)
```

4.2 From a PWM file

If you don't have a `gadem` object, you probably have a file containing PWM. `MotIV` currently supports two PWMs formats.

4.2.1 GADEM type

A file containing PWMs as provide by the standard output of the GADEM software. Usually named 'observedPWMs.txt'. In this case, you should use the `readGademPWMFile` on the file containing the motifs PWMs.

```
motifs.gadem <- readGademPWMFile(paste(path, "/extdata/observedPWMs.txt",
  sep = ""))
```

4.2.2 TRANSFAC type

`MotIV` also supported Transfac format file to load PWMs. For more information about the Transfac file format, please refer to <http://www.benoslab.pitt.edu/stamp/help.html>. If your data are in this format, proceed like in IV.2 :

```
motifs.example <- readPWMfile(paste(path, "/extdata/example_motifs.txt",
  sep = ""))
```

4.3 Trim Input

You can trim the edges of the input PWMs to improve the information content of your PWM. It could improve the results by removing the noise and generating better alignments. The default threshold is an information content of 1.

```
motifs.trimed <- trimPWMedge(motifs, threshold = 1)
```

5 MotIV Analysis

At this step, you should have all what you need to start the motifs matches analysis : input motifs, a database and the associated database scores file. To use the `motifMatch` function, be sure to provided the same alignment method and metric name used to the calculation of the database scores. The argument `top` indicates the number of motifs matches to find. To run the analysis, type :

```
foxa1.analysis.jaspar <- motifMatch(inputPWM = motifs, align = "SWU",
  cc = "PCC", database = jaspar, DBscores = jaspar.scores, top = 5)
```

```
Ungapped Alignment
Scores read
Database read
Motif matches : 5
```

or simply

```
foxa1.analysis.jaspar <- motifMatch(motifs)
```

```

Ungapped Alignment
Scores read
Database read
Motif matches : 5

```

for an analysis with default parameter using the JASPAR database.

This function will return an object of type `motiv` needed for next functions. Let's have a look to the content :

5.1 Summary

You can have a quick view to the content of your results. By typing :

```
summary(foxa1.analysis.jaspar)
```

```

Number of input motifs : 7
Input motifs names : m1 m2 m3 m4 m5 m6 m7
Number of matches per motif: 5
Matches repartition :
Egr1      Foxd3      INSM1      NFE2L2      T  Tal1::Gata1
  2         2         2         2         2         2
AP1       ESR1      EWSR1-FLI1  FEV        FOXA1      FOXD1
  1         1         1         1         1         1
FOXI1     FOXO3     Foxa2      Klf4      Myf NFE2L1::MafG
  1         1         1         1         1         1
PLAG1     PPARG    PPARG::RXRA  Pax2      REST      RREB1
  1         1         1         1         1         1
SP1       SPI1     SPIB       SRF       Stat3
  1         1         1         1         1
Arguments used :
-metric name : PCC
-alignment : SWU

```

you obtain the number of input motifs, their names, the number of matches per motif, the metric name and the alignment type used. The `summary` also offers the counting of identified transcription factors.

6 Filters

This functions are used to apply filters on a `motiv` object.

6.1 SetFilter

`setFilter` is used to define a filter. You can indicate the name(s) of the motifs to select, the TF name contained in the alignment, a maximum e-value, length and number of gap associated. The `top` argument defined the depth of the filter (i.e. the `top` first motif on witch the conditions should be applied). You should provided at least one argument.

```
f.foxa1 <- setFilter(tfname = "FOXA1", top = 3, evalueMax = 10^-5)
f.ap1 <- setFilter(tfname = "AP1", top = 3)
```

You will obtain an object of type `filter` used in the next described functions. . Use the `summary` function to have a view on the content.

6.2 Operators & and |

You can decide to combine different filters in order to define more complex filters. The & operator indicates that all filters conditions should be validated. To the opposite, with the | operator, one filter satisfied is enough to select the motif.

```
f.foxal.ap1 <- f.foxal | f.ap1
```

You also can combine more than two filters.

6.3 Filter

The `filter` function selects motifs that correspond to the set of filters. If `exact=TRUE`, search only for perfect name match.

```
foxal.filter <- filter(foxal.analysis.jaspar, f.foxal.ap1, exact = FALSE,
  verbose = TRUE)
```

It returns a `motiv` object with the selected motifs only.

6.4 Split

`split` is almost equivalent to the `filter` function. `split` is an easy way to select motifs according a list of filters. It will select all motifs that satisfy each filter and returns a list of `motiv` objects. If `drop=FALSE`, the non-selected motifs will also be returned.

```
foxal.split <- split(foxal.analysis.jaspar, c(f.foxal, f.ap1), drop = FALSE,
  exact = FALSE, verbose = TRUE)
```

6.5 Combine

The `combine` function is quite a bit different than the two previous functions. `combine` is used to consider many motifs as a single motif. For each filter of the list passed in argument, the `combine` function 'virtually' regroups motifs that satisfied the filters conditions.

```
foxal.filter.combine <- combine(foxal.filter, c(f.foxal, f.ap1),
  exact = FALSE, name = c("FOXA1", "AP1"), verbose = TRUE)
```

You should be careful that a same motif is not combined many times. Changes are not visible until `group` is not set on `TRUE`.

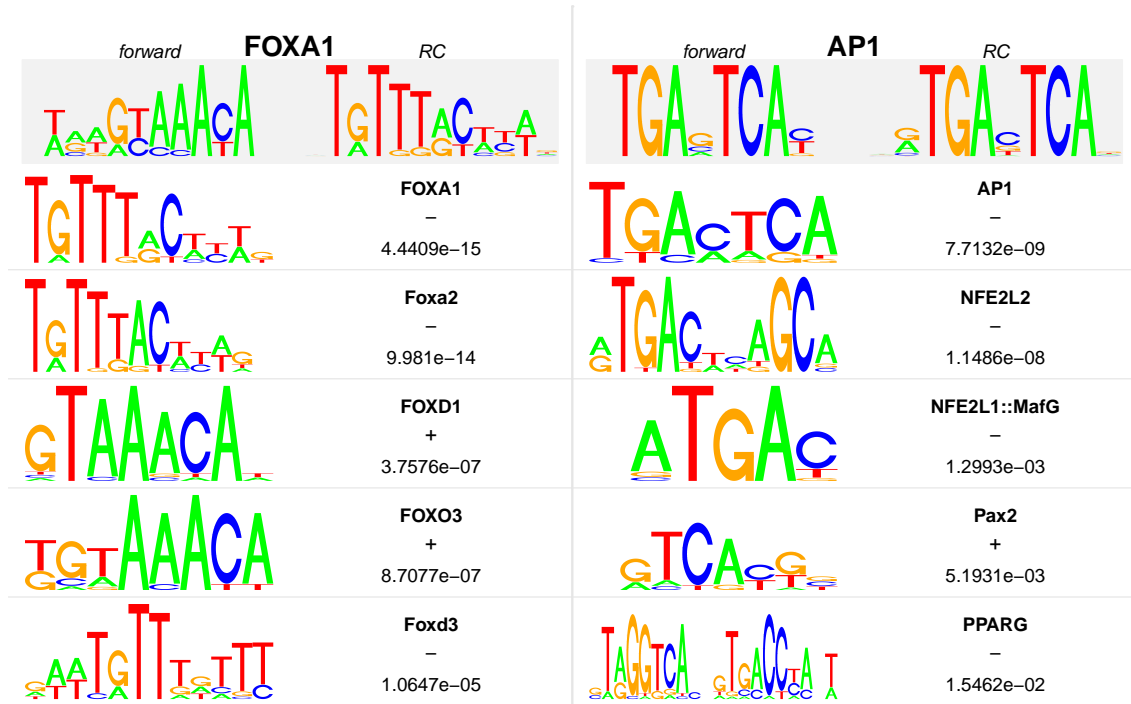
7 Results

7.1 Logo

`Plot` is the main function to visualize the results. This function provides a summary of each identified transcription factors associated to the input motifs, the sequence logo, the name of the motif match and the p-value of the alignment. The `top` argument allow you to choose the number of motif matches to print. The `rev` argument indicates if the logo should be plot according the motif strand or only print original TF logo.

```
plot(foxal.filter.combine, ncol = 2, top = 5, rev = FALSE, main = "Logo",
  bysim = TRUE)
```

Logo



7.2 Alignment

An other way to visualize the quality of the results is to look the alignments. E-value give an estimation of the match. You can explore further with :

```
foxa1.alignment <- viewAlignments(foxa1.filter.combine)
print(foxa1.alignment[[1]])
```

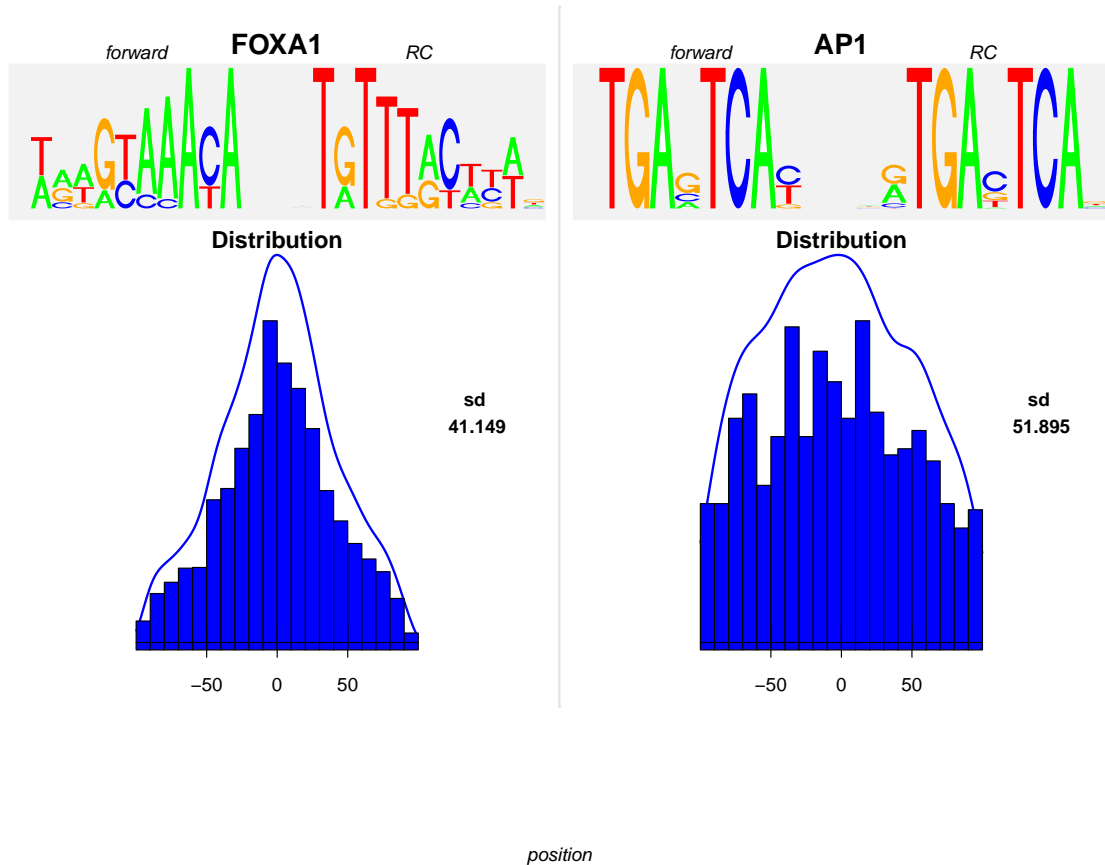
	FOXA1	Foxa2	FOXD1	FOXO3	Foxd3
seq	"NWRWGTAACAN"	"NTGTTTACWYWN-"	"NTGTTTACWYWN"	"NWRWGTAACAN"	"NWRWGTAACAN--"
match	"NWRWGYAAACA-"	"-TGTTTACWYWN"	"NTGTTTAC----"	"---TGTAACA-"	"--AAANAAACAWTN"
evaluate	"4.4409e-15"	"9.981e-14"	"3.7576e-07"	"8.7077e-07"	"1.0647e-05"

7.3 Distribution

As this function need an object of type `gadem`, you can use it only with a `rGADEM` analysis. The `plot` function offers to visualize the repartition of TF found. You should provided a `MotIV` and a `gadem` object and a valid layout. If you don't specify a sufficient layout, some motifs may be not plot (ie. specify a 2,2 layout will not plot the 5th motifs and more of the result).

```
plot(foxa1.filter.combine, gadem, ncol = 2, type = "distribution",
     correction = TRUE, group = FALSE, bysim = TRUE, strand = FALSE,
     sort = TRUE, main = "Distribution of FOXA")
```

Distribution of FOXA



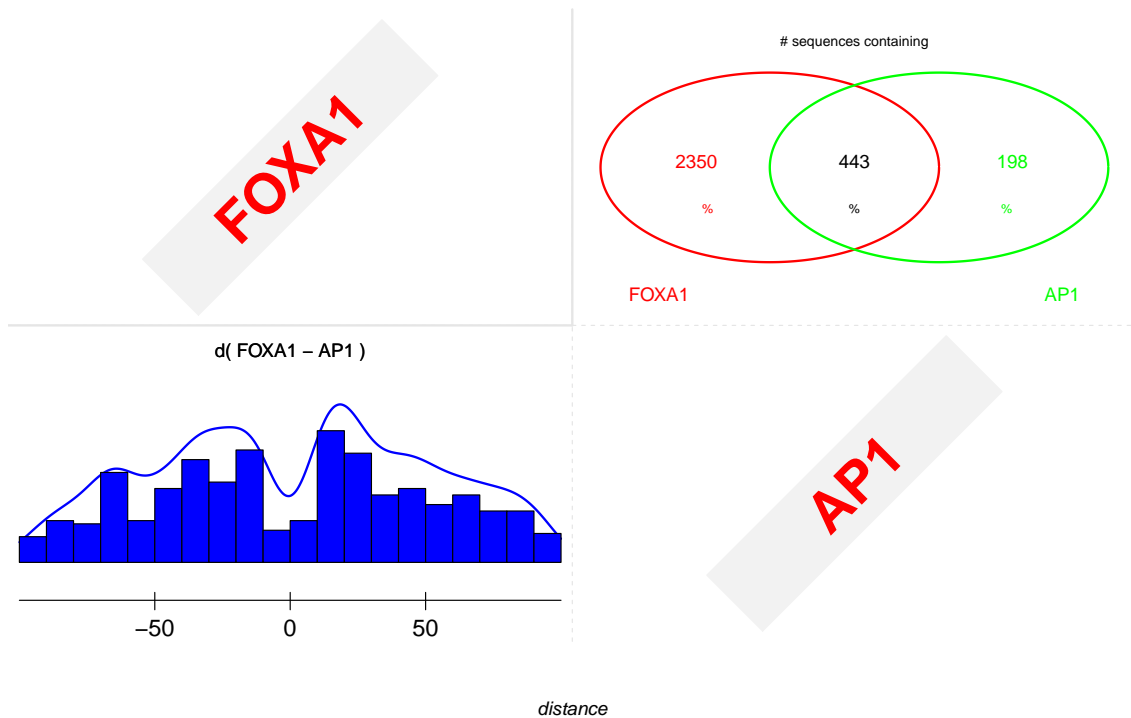
This function could help to distinguish between real motifs and background noise. Because of in theory peaks are center around motifs, distribution should be a gaussian. To the opposite, random motifs have a relative uniform distribution.

7.4 Distance

As this function need an object of type `gadem`, you can use it only with a `rGADEM` object. Use the `plot` function with `type='distance'` to visualized the distance between motifs. It also provides a vern diagram showing the number of single motifs as well as the number of motif present on the same peak. This function take a `MotIV` and a `gadem` object as arguments.

```
plot(foxa1.filter.combine, gadem, type = "distance", correction = TRUE,
     group = TRUE, bysim = TRUE, main = "Distance between FOXA and AP-1",
     strand = FALSE, xlim = c(-100, 100), bw = 8)
```

Distance between FOXA and AP-1



This function is an useful way to discover motifs co-occurrences. Studies showed that distance between two co-occurrent motifs are relatively constant. Thus, a bimodal curve around the peak center could indicate a potential co-occurrence.

8 RangedData

8.1 Ranged Data

A `rangedData` is an object created by the `IRanges` library [6]. To create a `rangedData` object, use the `exportAsRangedData` function on a `motiv` and `rGADEM` object.

```
foxa1.rd <- exportAsRangedData(foxa1.filter.combine["FOXA1"], gadem)
ap1.rd <- exportAsRangedData(foxa1.filter.combine["AP1"], gadem)
```

9 Saving and Exporting Results

9.1 motiv object

The best way to save your results is to use the `save` function. You should type :

```
save(foxa1.filter.combine, file = "foxa1_analysis.rda")
```

It will save the `MotIV` object into a file at your working directory. To load previous saved analysis, use the `load` function on the corresponding file.

9.2 Into Transfac Type Files

If you prefer export your results in a more readable format, use the `exportAsTransfacFile` function. It will write two files. The first file contains alignments for each input motifs. The second one references the entire PWMs corresponding to every identified transcription factors in Transfac format.

```
exportAsTransfacFile(foxa1.filter.combine, file = "foxa1_analysis")
```

9.3 Into a BED File

Once you created a `rangedData` object, you might want to write a BED file to save your data. To do it, simply use the `rtracklayer` export function.

```
library(rtracklayer)
export(foxa1.rd, file = "FOXA.bed")
```

10 Miscellaneous

10.1 viewMotifs

The `viewMotifs` function returns the list of all TF in a `motiv` object.

```
viewMotifs(foxa1.filter.combine, n = 5)
```

```
(Other)      AP1      FOXA1      FOXD1      FOXO3      Foxa2
           5         1         1         1         1         1
```

10.2 names

`names` returns the names of the motifs contained in a `motiv` object.

```
names(foxa1.filter.combine)
```

```
[1] "m1" "m6"
```

10.3 similarity

The `similarity` function shows the names of the similar motifs in a `motiv` object.

```
similarity(foxa1.filter.combine)
```

```
[1] "FOXA1" "AP1"
```

10.4 select

Use `[` to select a specific motif of a `motiv` object. By default, it will select the exact name of similar motifs. Choose `bysim=FALSE` to select the original name of the motifs. If `drop=FALSE`, the corresponding motifs will be drop of the object.

```
foxa1.selected <- foxa1.filter.combine["FOXA1"]
other.selected <- foxa1.filter.combine["FOXA1", drop = T]
```

Combine with other functions, it can be really useful. To know how many motifs FOXA1 you got, try by instance :

```
foxa1.names <- names(foxa1.filter.combine["FOXA1"])
sum(length(gadem[foxa1.names]))
```

```
[1] 1
```

Part V Appendix

11 GSL Installation

You need the GNU Scientific Library (GSL) for the `MotIV` package. Make sure it is installed on your machine if you want to use `MotIV`. GSL is free and can be downloaded at <http://www.gnu.org/software/gsl/> for Unix distributions and at <http://gnuwin32.sourceforge.net/packages/gsl.htm> for Windows.

Windows users

To install a pre-built binary of `MotIV` and to load the package successfully you need to tell R where to link GSL. You can do that by adding `/path/to/libgsl.dll` to the Path environment variable. To add this you may right click on "My Computer", choose "Properties", select the "Advanced" tab, and click the button "Environment Variables". In the dialog box that opens, click "Path" in the variable list, and then click "Edit". Add `/path/to/libgsl.dll` to the Variable value field. It is important that the file path does not contain any space characters; to avoid this you may simply use the short forms (8.3 DOS file names) found by typing "dir /x" at the Windows command line. For example, I added the following on my Windows machine: `C:/PROGRAM/GNUWIN32/bin` and used ";" to separate it from existing paths.

To build the `MotIV` package from source (using Rtools), in addition to adding `/path/to/libgsl.dll` to Path, you need to tell `MotIV` where your GSL library and header files are. You can do that by setting up two environment variables `GSL_LIB` and `GSL_INC` with the correct path to the library files and header files respectively. You can do this by going to the "Environment Variables" dialog box as instructed above and then clicking the "New" button. Enter `GSL_LIB` in the Variable name field, and `/path/to/your/gsl/lib/directory` in the Variable value field. Likewise, do this for `GSL_INC` and `/path/to/your/gsl/include/directory`. Remember to use `/` instead of `\` as the directory delimiter.

You can download Rtools at <http://www.murdoch-sutherland.com/Rtools/> which provides the resources for building R and R packages. You should add to the Path variable the paths to the various components of Rtools. Please read the "Windows Toolset" appendix at <http://>

[//cran.r-project.org/doc/manuals/R-admin.html#The-Windows-toolset](http://cran.r-project.org/doc/manuals/R-admin.html#The-Windows-toolset) for more details.

Unix/Linux/Mac users

When building the package, it will look for a BLAS library on your system. By default it will use `gslcblas`, which is not optimized for your system. To use an optimized BLAS library, you can use the `--with-blas` argument which will be passed to the `configure.ac` file. For example, on a Mac with `vecLib` pre-installed the package may be installed via: `RCMDINSTALLMotIVx.y.z.tar.gz--configure-args=" --with-blas='-frameworkvecLib'`

Part VI

References

References

- [1] S. Mahony, P.V. Benos. *STAMP: a web tool for exploring DNA-binding motif similarities*. Nucl Acids Res. 2007. 35:W253-258
- [2] Oliver Bembom. SeqLogo package available on www.bioconductor.org
- [3] Banerjee, N. & Zhang, M. Q. *Identifying cooperativity among transcription factors controlling the cell cycle in yeast*. Nucleic Acids Res. 2003. 31, 7024-31.
- [4] Portales-Casamar E. *et al. JASPAR 2010: the greatly expanded open-access database of transcription factor binding profiles*. Nucleic Acids Res. 2010 Jan;38(Database issue):D105-10. Epub 2009 Nov 11.
- [5] Matys V. *et al. TRANSFAC: transcriptional regulation, from patterns to profiles*. Nucleic Acids Res. 2003 Jan 1;31(1):374-8
- [6] H. Pages, P. Aboyoun and M. Lawrence, available at <http://www.bioconductor.org/packages/bioc/html/IRanges.html>.