Supplemental R codes

Setting parameters

The codes provided here were used to analyze and visualize (Figure 1-3) SELEX originated aptamer data of this work concerning motif conservation, secondary DNA structure, and conservation of bulge loop size/sequence/position after removal of unpaired structures. The code is adjusted to NGS data pre-processed by AptaSUITE software and requires several R packages and the software packages UNAfold and MEME.

Several parameters has to be adjusted before running the code. First the AptaSUITE export file has to be imported. The cluster family of the file will be sorted by decreasing cluster size. The subset of cluster families that should be analyzed has to be stated.

The software MEME is used to find common sequence motifs in the cluster families. Therefore, the binary of the external software as well as the search parameters have to be adjusted. This first search will be used to give an overview of the found motifs. In a later stage of the code a second and final search for consensus sequences will be applied based on findings of the first overview search. In order to compare the logos with published logos, the JASPAR data base was used.

Predicted secondary structures of the clusters are essential components for the analysis. They will be provided by UNAfold software as single extended ct files, as required for this code. The foldings of a cluster family can differ by their Gibb's free energy. For statistical reasons we included only one folding per sequence in the analysis. The code allows for selecting only the folding with the lowest Gibb's free energy (default), lowest E-value created by MEME, or none.

```
# Essential R packages
library("R4RNA")
library("Biostrings")
library("ggseqlogo")
library("TFBSTools")
library("JASPAR2014")
library("ggplot2")
library("viridis")
library("RRNA")
library("gtools")
library("DNAshapeR")
library("ggpubr")
library("grid")
library("gridExtra")
library("cowplot")
library("seqRFLP")
library("VennDiagram")
library("reshape2")
library("stringr")
library("tidyr")
library("RColorBrewer")
library("stringr")


# AptaSuite file parameters
cluster_file<-"cluster_table.txt" # default export file name from AptaSuite
cluster_subset<-1:350 # cluster family subset to be read in

# UNAFold parameter for a liberal initial search
```

```
MEME_arguments<-list("-dna",
                     "-mod"=c("anr"),
                     "-nmotifs"=3,
                     "-evt"=0.05)

# UNAFold parameter for the strict second search
MEME_arguments2<-list( "-dna",
                       "-mod"=c("anr"),
                       "-w"=11,
                       "-nmotifs"=2,
                       "-evt"=0.05)

# other parameters
JASPAR_ID<-"MA0468.1" #JASPAR ID i.e. ID for human DUX4 (MA0468.1)
choic<-1 #selecting unique FASTA by dG ("1") or MEME score ("2")
```

The next section is for reading in the cluster file exported from AptaSUITE software used to pre-process the NGS data of a SELEX experiment.

The clusters were formatted into a conventional FASTA format in order to be fed into the external *de novo* motif search software MEME. MEME is accessed via the R package TFBSTools.

The output is formatted for further processing. For reason of comparison, the consensus binding site logo of the transcription factor DUX4 is loaded via TFBSTools and visualized.

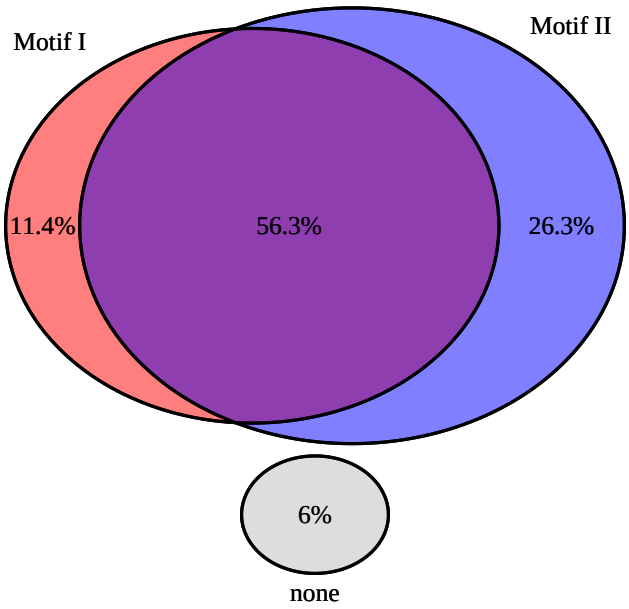
The next code snippet creates the file clusterfasta.txt that has to be loaded into UNAfold later.

```
# reading in and subsetting cluster file from AptaSuite
read.csv(cluster_file,sep="\t",stringsAsFactors = FALSE)->cluster
cluster[order(cluster$round.7.Size,decreasing=TRUE),]->cluster
cluster[cluster_subset,c(1,2)]->cluster2fasta #using cluster index as fasta name
as.character(cluster2fasta[,1])->cluster2fasta[,1]
make.unique(cluster2fasta[,1],sep="")->cluster2fasta[,1]
df.fasta = seqRFLP::dataframe2fas(cluster2fasta, file="clusterfasta.txt")
#rm(cluster, cluster2fasta)

# conversion into FASTA and modify for runMEME command
Biostrings::readDNAStringSet("clusterfasta.txt",use.names=TRUE)->fasta_var
unlink("clusterfasta.txt")
as.data.frame(fasta_var)->fasta_var2
as.data.frame(fasta_var@ranges)->fasta_var3
data.frame(x=fasta_var2,names=fasta_var3[,5])->fasta_var2
1:nrow(fasta_var2)->fasta_var2$index
rm(fasta_var3)

# run MEME via R
TFBSTools::runMEME(fasta_var,seqtype="DNA", arguments=MEME_arguments)->MEME_output_1
Biostrings::consensusMatrix(MEME_output_1)->MEME_output_1_cons_Matrix

list()->logo_list0
for(i in 1:length(MEME_output_1_cons_Matrix)){
  logo_list0[[i]]<-ggplot()+
    geom_logo(MEME_output_1_cons_Matrix[i])+
    theme_logo()+
    theme(plot.title = element_blank())
}
```

```
# for comparison extract the JASPAR data for the transcription factor
db <- file.path(system.file("extdata", package="JASPAR2014"),"JASPAR2014.sqlite")
TFBSTools::getMatrixByID(db, ID=JASPAR_ID)->db_profile
db_profile_pwm <- db_profile@profileMatrix
db_profile_pwm_rc <- Biostrings::reverseComplement(db_profile_pwm)
rm(db,db_profile)
logo_list1<-list()
ggseqlogo::ggseqlogo(db_profile_pwm_rc)->logo_list1[[2]]
ggseqlogo::ggseqlogo(db_profile_pwm)->logo_list1[[1]]

as.data.frame(MEME_output_1@motifList)->modf0
colnames(modf0)[3] <- "names"
as.data.frame(MEME_output_1@subjectSeqs)->modf02
fasta_var2$names->modf02$names
1:nrow(modf02)->modf02$index
merge(modf0,modf02,by.x="names",by.y="names")->memodf0
nchar(memodf0[,9])->memodf0$length
rm(modf02)
```

Creating a Venn diagram based on the data from MEME via the VennDiagram package.

```
memodf0[,c("index","group")]->memodf_sub0
memodf_sub0[memodf_sub0$group==2,]->sub10
memodf_sub0[memodf_sub0$group==1,]->sub20
unique(sort(sub10$index))->sub10
unique(sort(sub20$index))->sub20

length(which(sub10%in%sub20))->both0
length(sub10)->length_sub10
length(sub20)->length_sub20
nrow(cluster2fasta)-length(unique(memodf0$names))->lonely0

trivenn1<-VennDiagram::draw.triple.venn(area1=length_sub10,
                                        area2=length_sub20,
                                        area3=lonely0,
                                        n12=both0, n23=0, n13=0, n123=0,
                                        fill=c("red","blue","grey"),
                                        category = c("Motif I", "Motif II", "none"),
                                        alpha=c(0.3),
                                        print.mode="percent",
                                        sigdigs=3)
(grid::grid.draw(trivenn1))
```



For locating the motives within the aptamer sequences, the following plot was generated.

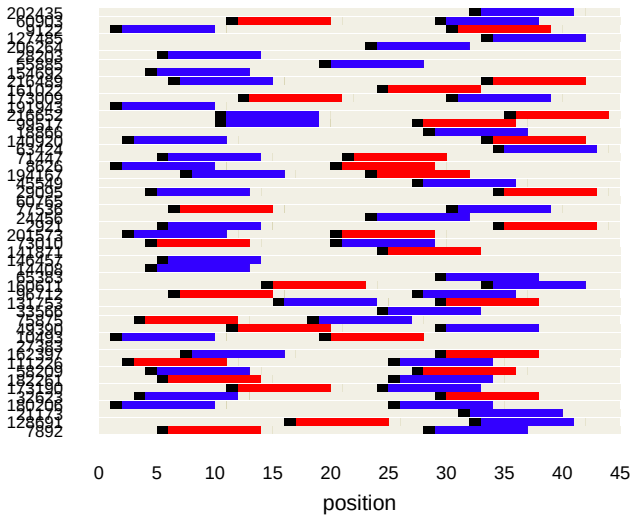
```
### location chart ######################################
as.integer(memodf0[,10])->memodf0[,10]
memodf0[order(memodf0[,1]),]->memodf0

ifelse(length(cluster_subset)>50,yes=var<-50,no=length(cluster_subset)->var)
```

```
## [1] 50
```

```
fasta_var2[1:var,]->fafa3
memodfshort<-memodf0[which(memodf0$names%in%fafa3$names),]

as.character(memodfshort$names)->memodfshort$names
as.character(fafa3$names)->fafa3$namess

box.size <- 1

df <- data.frame(xmin = c(NA),
                 xmax = c(NA),
                 ymin = c(NA),
                 ymax = c(NA),
                 fill = c(NA))
df->df0

hue_tempA<-0
hue_tempC<-0.15
hue_tempG<-0.24
hue_tempT<-0.7

hsv(hue_tempA, 1,1,1)->col_tempA
hsv(hue_tempC, 1,0.5,0.1)->col_tempC
hsv(hue_tempG, 1,0,1)->col_tempG
hsv(hue_tempT, 1,1,1)->col_tempT

for(i in 1:nrow(fafa3)){
```

```
as.character(fafa3[i,2])->j

if (j %in% memodfshort$names){

  memodfshort[which(j==memodfshort$names),]->memodfshort_sub
  memodfshort_sub[order(memodfshort_sub$start),]->memodfshort_sub
  count<-0
  df0->df0a
  for(k in 1:nrow(memodfshort_sub)){

    ifelse(memodfshort_sub[k,2]==1&memodfshort_sub[k,7]=="+",
           yes=c(col_tempC,col_tempG,col_tempT,col_tempC,col_tempC)->col_temp,
           no=
    ifelse(memodfshort_sub[k,2]==2&memodfshort_sub[k,7]=="+",
            yes=c(col_tempC,col_tempG,col_tempA,col_tempC,col_tempC)->col_temp,
            no=
    ifelse(memodfshort_sub[k,2]==1&memodfshort_sub[k,7]=="-",
              yes=c(col_tempC,col_tempC,col_tempT,col_tempG,col_tempC)->col_temp,
              no=
    ifelse(memodfshort_sub[k,2]==2&memodfshort_sub[k,7]=="-",
                yes=c(col_tempC,col_tempC,col_tempA,col_tempG,col_tempC)->col_temp,
                no=c(col_tempG,col_tempG,col_tempG,col_tempG,col_tempG)->col_temp))))


    df1a <- data.frame(xmin=c(count,memodfshort_sub[k,4],
                              (1+memodfshort_sub[k,4]),
                              (memodfshort_sub[k,5]-1),
                               memodfshort_sub[k,5]),
                       xmax=c(memodfshort_sub[k,4],
                              (1+memodfshort_sub[k,4]),
                              (memodfshort_sub[k,5]-1),
                              memodfshort_sub[k,5],
                              ifelse(k==nrow(memodfshort_sub),
                                  yes=nchar(fafa3[i,1]),no=c(memodfshort_sub[k,5]))),
                       ymin = rep(((i-0.9) * box.size),times=5),
                       ymax = rep((i * box.size),times=5),
                       fill = c(col_temp))

    df0a<-rbind(df0a,df1a)
    count<-memodfshort_sub[k,5]

  }
  df0a[-1,]->df0a
  df0a->df1
}else{
  df1 <- data.frame(xmin = 0,
                    xmax = nchar(fafa3[i,1]),
                    ymin = (i-0.9) * box.size,
                    ymax = i * box.size,
                    fill = c(col_tempC))

}
rbind(df,df1)->df
```

```
}
df[-1,]->df
barplo<-ggplot(df) +
  geom_rect(aes(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax, fill = fill)) +
  scale_fill_identity()+
  scale_y_continuous(breaks=c(1:nrow(fafa3))-0.5, labels=fafa3$names)+
  scale_x_continuous(name ="position",breaks=seq(0,50,by=5))+
  theme(axis.line=element_blank(),
        axis.ticks = element_blank(),
        axis.text.y =  element_text(size=rel(0.9))
  )
barplo
```



Bulge loop conservation: PART 1

This code part is used to import the ct files. Gibb's free energy, sequence, and FASTA names are extracted. Extended ct file format is converted to common ct file format in order to use the data for the R packages R4RNA and RRNA for visualization of the secondary structures.

```
### PART 1 ##################################################################
# reading in file names
ctfilenames <- list.files(pattern="\\.ct$")

# reading in files
myctfiles <- lapply(ctfilenames,
                    read.csv,
                    sep="\t",
                    header=FALSE,
                    skip=1)
lisnames <- lapply(ctfilenames,
                   read.csv,
                   nrow=1,
                   sep="\t",
                   header=FALSE,
                   stringsAsFactors=FALSE)

# extracting info from header
```

```
lapply(lisnames, "[[", 1)->yy
lapply(lisnames, "[[", 2)->xx
lapply(lisnames, "[[", 3)->zz
lapply(xx,as.character)->xx
lapply(yy,as.character)->yy
lapply(zz,as.character)->zz
sapply(c(xx),unlist)->xxx #list with dG values
sapply(c(yy),unlist)->yyy #list with sequence length
sapply(c(zz),unlist)->zzz #list with seqence name

#extracting info from header into vectors
make.unique(zzz,sep="a")->zzzz
cbind(cbind(zzzz,c(1:length(zzzz))),ctfilenames,zzz)->zzzi

cbind(yyy,xxx,zzz)->AS
matrix(apply(AS, 1, paste, collapse=" "), ncol=1)->namescoll

#removing additional columns genereated by UNAfold
lapply(myctfiles ,"[",c(1:6))->AA

# generating new and modified column names
rep(NA,5)->narow
rep(narow, times=nrow(namescoll))->NAmatrix
matrix(NAmatrix,ncol=5)->NAmatrix
cbind(namescoll,NAmatrix)->NAmatrix
as.vector(t(NAmatrix))->NN

# generating the final data frame "AAA" containing all data
AAA<-list()
jj<-1
for (i in AA){
  setNames(object=i,nm=NN[1:6])-> AAA[[jj]]
  jj+1->jj
  NN[-c(1:6)]->NN
}
```

Bulge loop conservation: PART2

UNAfold also creates a single file per run with the extension .det containing thermodynamic details of all foldings as well as structure information of each base per sequence. Loop type is also included, which is necessary for creating the Vienna structure format. The Vienna format is a way to indicate nucleotide structure elements as brackets (paired) and dots (unpaired) along a nucleotide sequence.

```
### PART 2 ##################################################################

# reading in det file
read.csv("clusterfasta.txt.det",sep="\t",header=FALSE)->df_det
df_det->df_det2
df_det[nrow(df_det),]->df_det[(nrow(df_det)+1),]
counter<-1->vector_df_det
for(i in 1:nrow(df_det)){
  ifelse(grep("Structure",as.character(df_det[i,]))==1,
         yes= i->vector_df_det[counter],
         no=NA->vector_df_det[counter] )
```

```
  counter+1->counter
}
vector_df_det[!is.na(vector_df_det)]->vector_df_det
nrow(df_det)->vector_df_det[length(vector_df_det)+1]

# tidy up det file variable: seperating foldings
list()->dfs_list
c(1)->header_dfs
for(j in 1:(length(vector_df_det)-1)){
  vector_df_det[j]->start_df
  vector_df_det[(j+1)]->end_df
  as.character(df_det[start_df,1])->header_dfs[j]
  df_det[(start_df + 1):(end_df-1),]->dfs_list[[j]]
}

# tidy up det file variable: extracting data into new data frame
list()->dfs_list_final
for(j in 1:length(dfs_list)){
  as.vector(dfs_list[[j]])->temp_df
  as.data.frame(matrix(NA,length(temp_df),9))->final_temp_df
  colnames(final_temp_df)<-c("category","ddG","ss_bases", "closing_base", "base_1",
                             "pos_1", "base_2", "pos_2", "bases")

  for(i in 1:length(temp_df)){
    unlist(strsplit(as.character(temp_df[i]),c(":")))->df_det2
    unlist(strsplit(as.character(df_det2[2]),c("*ddG =| ")))->df_det3
    df_det3[!df_det3==""]->df_det3
    final_temp_df[i,1]<-df_det2[1]->item


    if(item=="External loop"){

      #External loop
      df_det3[1]->final_temp_df[i,2]
      df_det3[2]->final_temp_df[i,3]
      df_det3[6]->final_temp_df[i,4]
    } else if(item=="Stack"){

      #Stack
      unlist(strsplit(as.character(df_det3[-1]),c("\\(|\\)-|\\)")))->df_det4
      c(df_det3[1],df_det4)->df_det3

      df_det3[1]->final_temp_df[i,2]
      df_det3[6]->final_temp_df[i,5]
      df_det3[7]->final_temp_df[i,6]
      df_det3[8]->final_temp_df[i,7]
      df_det3[9]->final_temp_df[i,8]
    } else if(item=="Bulge loop"){

      #Bulge loop
      unlist(strsplit(as.character(df_det3[-1]),c("\\(|\\)-|\\)")))->df_det4
      c(df_det3[1],df_det4)->df_det3
```

```
      df_det3[1]->final_temp_df[i,2]
      df_det3[6]->final_temp_df[i,5]
      df_det3[7]->final_temp_df[i,6]
      df_det3[8]->final_temp_df[i,7]
      df_det3[9]->final_temp_df[i,8]
    } else if(item=="Helix"){

      #Helix
      df_det3[1]->final_temp_df[i,2]
      df_det3[2]->final_temp_df[i,9]
    } else if(item=="Interior loop"){

      #Interior loop
      unlist(strsplit(as.character(df_det3[-1]),c("\\(|\\)-|\\)")))->df_det4
      c(df_det3[1],df_det4)->df_det3

      df_det3[1]->final_temp_df[i,2]
      df_det3[6]->final_temp_df[i,5]
      df_det3[7]->final_temp_df[i,6]
      df_det3[8]->final_temp_df[i,7]
      df_det3[9]->final_temp_df[i,8]
    } else if(item=="Hairpin loop"){

      #Hairpin loop
      unlist(strsplit(as.character(df_det3[-1]),c("\\(|\\)-|\\)")))->df_det4
      c(df_det3[1],df_det4)->df_det3

      df_det3[1]->final_temp_df[i,2]
      df_det3[5]->final_temp_df[i,5]
      df_det3[6]->final_temp_df[i,6]
      df_det3[7]->final_temp_df[i,7]
      df_det3[8]->final_temp_df[i,8]

    } else { "else"->final_temp_df[i,1]}
    final_temp_df->dfs_list_final[[j]]
  }
}

# ct file index vs. det file index

file_no<-c()
for(i in 1:length(ctfilenames)){
  x<-ctfilenames[i]
  k<-stringr::str_extract_all(x, "\\_[^()]+\\.ct")[[1]]
  substring(k, 2, nchar(k)-3)->file_no[i]
}

# creating new list AAAf for det data
AAA->AAAf
for(j in 1:length(AAAf)){ #master loop start
  AAAf[[j]]->AAAfb
  as.double(file_no[j])->indexf
  dfs_list_final[[indexf]]->hype
```

```
as.double(yyy[j])->length_fasta

subset(hype,category!="Helix"&category!="External loop")->hype2
hype2[,c(1,6,8)]->hype2

as.double(hype2$pos_1)->hype2$pos_1
as.double(hype2$pos_2)->hype2$pos_2
hype2[order(hype2$pos_1),]->hype2

rep(NA,1,length_fasta)->vector_seq
for (i in 1:(nrow(hype2)-1) ){

  what_1<-hype2[i,1]

  if (what_1=="Hairpin loop"){

    what<-hype2[i,1]

    a<-as.double(hype2[i,2])
    b<-as.double(hype2[i,3])

    a_b<-c((a+1):(b-1))
    vector_seq[c(a,b)]<-"Stack"
    vector_seq[a_b]<-what

  }else{

    a_1<-as.double(hype2[i,2])
    a_2<-as.double(hype2[(i+1),2])
    b_1<-as.double(hype2[i,3])
    b_2<-as.double(hype2[(i+1),3])

    a_a<-c((a_1+1):(a_2-1))
    b_b<-c((b_2+1):(b_1-1))

    if((a_1+1)>(a_2-1)&(b_1-1)<(b_2+1)){a_a<-NA; b_b<-NA;} #norm
    if((a_1+1)>(a_2-1)&(b_1-1)>=(b_2+1)){a_a<-NA; b_b<-b_b;} #right
    if((a_1+1)<=(a_2-1)&(b_1-1)<(b_2+1)){a_a<-a_a; b_b<-NA;} #left
    if((a_1+1)<=(a_2-1)&(b_1-1)>=(b_2+1)){a_a<-a_a; b_b<-b_b;} #interior

    vector_seq[c(a_1,b_1,a_2,b_2)]<-"Stack"
    vector_seq[a_a]<-what_1
    vector_seq[b_b]<-what_1
  }}

i<-nrow(hype2)
what<-hype2[i,1]

a<-as.double(hype2[i,2])
b<-as.double(hype2[i,3])

a_b<-c((a+1):(b-1))
vector_seq[c(a,b)]<-"Stack"
```

```
vector_seq[a_b]<-what

vector_seq[is.na(vector_seq)]<-"External loop"

vector_seq->AAAfb$structure

vector_seq->vec_test
for(k in 1:length(vec_test)){
  gsub("Hairpin loop","H",vec_test[k])->vec_test[k]
  gsub("External loop","E",vec_test[k])->vec_test[k]
  gsub("Interior loop","I",vec_test[k])->vec_test[k]
  gsub("Bulge loop","B",vec_test[k])->vec_test[k]
  gsub("Stack","S",vec_test[k])->vec_test[k]
}
vec_test->AAAfb$str_abbrev

# # # # # # # #
AAAfb->AAAf[[j]]
AAAfb[,8]->int_df
if("I"%in%int_df){

  paste(as.vector(t(as.vector(int_df))),collapse="", sep="")->int_df_string
  gregexpr("I", int_df_string)->int_df_str_greg
  as.vector(int_df_str_greg[[1]])->int_df_str_greg
  (c(0,diff(int_df_str_greg))!=1)->int_df_str_greg_diff
  c(1:length(int_df_str_greg))->int_df_str_greg_num
  data.frame(num=int_df_str_greg_num,
             greg=int_df_str_greg,
             diff=int_df_str_greg_diff)->df_grep
  df_grep[(nrow(df_grep)+1),]<-c(0,0,TRUE)

  as.vector(df_grep$diff)->df_grep_vec
  counter<-0
  counter1<-0
  counter2<-c(rep(1,time=length(df_grep_vec[df_grep_vec==1])))
  for(i in 1:length(df_grep_vec)){
    if(df_grep_vec[i]==1){
      counter<-1
      counter1<-counter1+1
      counter2[counter1]<-counter
    }else{
      counter+1->counter
      counter->counter2[counter1]
    }
  }
  counter2[1:(length(counter2)-1)]->counter_size_final
  df_grep[which(df_grep$diff==1),2]->counter_pos
  counter_pos[1:(length(counter_pos)-1)]->counter_pos_final
  (counter_size_final+counter_pos_final)-1->counter_end_final

}}
```

Bulge loop conservation: PART3

Part 3 contains the initialization for further calculations and removes unpaired structures from the aptamer sequences.

```
### PART 3 ###################################################################

#creating list without unpaired bases called AAAB
AAA->AAAB
AAAg<-AAA->AAAe

for (i in seq_along(AAA)){
  as.data.frame(AAA[[i]])->AAAa
  c()->AAAa$j

  # filling 7th column with 0 for unpaired and 1 for paired
  for(k in 1:nrow(AAAa)){
    AAAa[k,7] <- ifelse(AAAa[k,5]>0, 1, 0) #calculate base pairing score for 7th column
  }

  h<-rep(0,nrow(AAAa))
  for(m in 1:nrow(AAAa)){ #refining column for calculating the size of the bulge loop
    if(AAAa[m,7]==0){
      c(m)->h[m+1]
    }
    AAAa[-h,]->AAAb #removing unpaired bases and saving in AAAb
    AAAb->AAAB[[i]]
  }

 # creating list with Helix files for structure packages
  AAAa->AAAad
  AAAad[,5]->AAAad[,2]
  AAAad[,3]<-rep(1,length(AAAad[,1]))->AAAad[,4]
  AAAad[,1:4]->AAAad
  AAAg[[i]]<-AAAad

  # creating list with new ct files for structure packages
  AAA[[i]]->zu
  zu[max(zu[,1]),4]<-(max(zu[,1])+1)
  zu->AAAe[[i]]
}
```

Bulge loop conservation: PART4

The main part of the code uses the structure data to determine the loop conservation and summarizes the data in a single data frames.

The code uses the sequences that lack bases that are part of unpaired structures including bulge loops. The data is converted to be read into MEME for a second motif search. Sequence logos are created and the bulge loop information is mapped back onto the sequences to gain information about the locations and sizes of the loops within the motifs.

Bulge loop conservation: PART5

In this part only one folding per cluster is selected. The selection is according to the parameter set at the beginning of the code: either by dG or MEME score.

Bulge loop conservation: PART6

The data is visualized in bar charts and sequence logos. The summary data frame containing all data is finalized in this section.

```
### PART 4 ##########################################################################

# initialize and open loop across all items in ct file list
results1<-read.csv(text="name,seq")
colnames(results1)<-c("name","seq")
NA->results1[1,]

for(i in seq_along(AAAB)){ #master loop with AAAB (w/o loops)

  AAA[[i]][,2]->AASss
  Biostrings::DNAString(paste(AASss,collapse="",sep=""))->AAss
  AAAB[[i]][,2]->AASSS

  Biostrings::DNAString(paste(AASSS,collapse="",sep=""))->AASS
  AAABseq<-paste(AASSS,collapse="",sep="")

  zzzz[i]->fastanames_1
  cbind(fastanames_1,AAABseq)->res2
  colnames(res2)<-c("name","seq")
  rbind(results1,res2)->results1
}
results1[-1,]->forfasta
seqRFLP::dataframe2fas(forfasta, file="results1fasta.txt")

Biostrings::readDNAStringSet("results1fasta.txt",use.names=TRUE)->fafax
unlink("results1fasta.txt")
fafax->fafa2x
as.data.frame(fafa2x)->fafa2x
as.data.frame(fafax@ranges)->fafa3x
data.frame(x=fafa2x,names=fafa3x[,5])->fafa2x
1:nrow(fafa2x)->fafa2x$index

#selecting for seqs longer than 11 otherwise error:
which(width(fafax)<MEME_arguments2$`-w`)->woseq
TFBSTools::runMEME(fafax[-woseq],seqtype="DNA", arguments=MEME_arguments2)->mo2

Biostrings::consensusMatrix(mo2)->moco2
ggseqlogo::ggseqlogo(moco2)->molo
as.data.frame(mo2@motifList)->modf
colnames(modf)[3] <- "names"
as.data.frame(mo2@subjectSeqs)->modf2 #all sequences
names(mo2@subjectSeqs)->modf2$names #all names but not unique
make.unique(modf2$names, sep = "")->modf2$names #makes names unique again
merge(modf,modf2,by.x="names",by.y="names",all.x=TRUE)->memodf

merge(modf,modf2,by.x="names",by.y="names",all=TRUE)->memodf_complete
reshape2::dcast(memodf_complete, names ~ group)->yes_no_df

nchar(memodf[,9])->memodf$length
```

```
memodf_list<-list()->results3_list
plot_list<-list()->logo_list2
results1list<-list()->logo_list3

#loop over all motives
for(lindex in 1:length(mo2)){

  #indexing
  memodf[memodf[,2]==lindex,c(1,4,5,6,7,8,9,10)]->memodf1
  c("names_1","start_1",
    "end_1","width_1","strand_1",
    "score_1","x_1","length_1")->colnames(memodf1)
  memodf_list[[lindex]]<-memodf1->memodf_go
  results1meme<-read.csv(text=paste(sprintf("S%d",1:20),collapse=",",sep=""))
  colnames(results1meme)<-c("sequence","bulge position","bulge size","bulge seq","dG",
                            "total lenght","paired","mo start","mo end","motif",
                            "alignment score","fasta","ct file","motif in seq","index",
                            "bulge amount","seq w/o unpaired","structure","motif name",
                            "Vienna")->headercol
  NA->results1meme[1,]

  mo2@motifList@partitioning@NAMES->mo_names
  mo_names[lindex]->mo_names

  # master loop
  for(o in 1:nrow(memodf_go)){

    # find highest score and calculate position
    as.numeric(zzzi[which(zzzi[,1]%in%memodf_go[o,1]),2])->i
    AAAB[[i]]->AAAB_exit
    AAAB_exit[memodf_go[o,2],1]->mostart
    AAAB_exit[memodf_go[o,3],1]->moend
    memodf_go[o,6]->maxres

    AAA[[i]][,2]->AASss
    Biostrings::DNAString(paste(AASss,collapse="",sep=""))->AAss
    AAAB[[i]][,2]->AASSS
    Biostrings::DNAString(paste(AASSS,collapse="",sep=""))->AASS
    AAABseq<-paste(AASSS,collapse="",sep="")
    AAAf[[i]]->AAAfseq_df
    AAAfseq_df[,8]->AAAfseq_pre
    as.vector(AAAfseq_pre)->AAAfseq_pre
    paste(AAAfseq_pre,collapse="",sep="")->AAAfseq

    # creating row with info about base pairing/bulge loops
    AAA[[i]][,5]->D #basepairing info
    rep(0,times=length(D))->k #create vector k with info about size of each bulge loop
    for(l in seq_along(D)){
      if (D[l]==0){
        ifelse(l==1&D[l]==0,yes=(k[1]<-1),no=(k[l-1]+1->k[l]))
      }else{0->k[l]}
    }
```

```
as.data.frame(AAA[[i]])->AAAcc
cbind(AAAcc,k)->AAAcc
AAAcc[mostart:moend,]->AAAaa

AAAaa[,7]->u
seq_along(u)->kk
v<-0
for(m in seq_along(u)){
  ifelse(u[m]==0,yes=(kk[m]<-v+1->v),no=(0->kk[m]))
}
cbind(AAAaa,kk)->AAAaa

# differentiating if motif in sequence has bulges or not
if (sum(AAAaa[,7])!=0){

  # collecting data whether there are bulges
  max(as.numeric(levels(factor(AAAaa[,7]))))->maxsize
  as.numeric(maxsize)->maxsize
  length(which(AAAaa[,7]==1))->amountbulge

  # counting bulges, assess their positions (ww) and sizes (r) in the motif
  qdf<-data.frame(matrix(vector(), 1, amountbulge)) #initialize empty df
  for (q in seq_along(1:maxsize)){
    (which(AAAaa[,7]==q)-q)->qvec
    nn <- max(length(qdf[1,]), length(qvec)) #work-around vector recycling
    length(qvec) <- nn
    rbind(qdf,qvec)->qdf #contains info about position (value) and size (amount of value)
  }
  as.vector(t(qdf))->w
  w <- w[!is.na(w)]
  as.double(levels(factor(w)))->ww

  c(1:length(ww))->r
  counterw<-1
  for (z in ww){
    length(w[w==z])->r[counterw]
    counterw+1->counterw
  }


  # sum up everything
  # sequence
  paste(AAss,collapse="",sep="")->nucvec
  rep(nucvec,length(r))->nunu

  # dG values
  colnames(AAAcc)->string
  strsplit(sub(".*dG = ", "", string[1])," ")->qwer
  unlist(qwer)->qwert
  qwert[1]->dG
  rep(dG,length(r))->dGs

  # motif with bulges
```

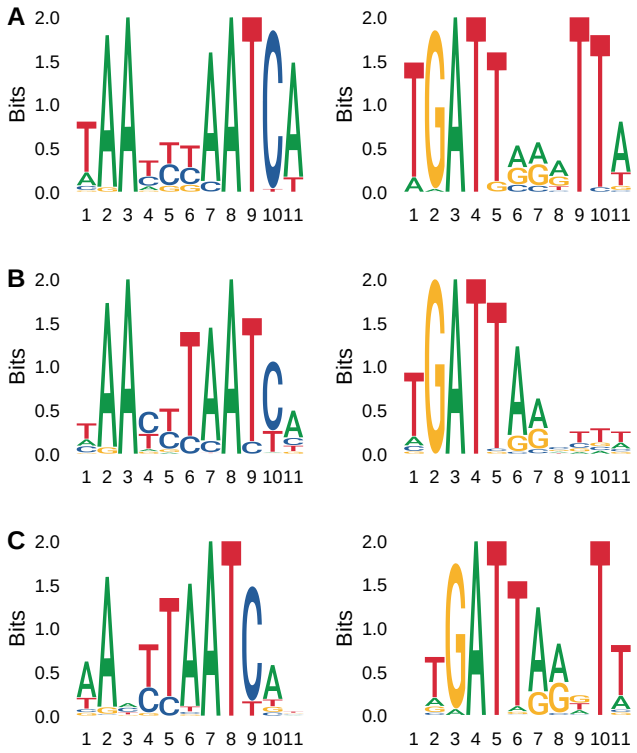
```
length(AAAcc[,7])->lennuc
round(100/(length(AAAcc[,7]))*(length(AAAcc[AAAcc[,7]>0,7])),2)->propaired
paste(AAAcc[mostart:moend,2],collapse="",sep="")->momo

#vienna format
AAAg[[i]]->ve
ve[which(ve[,2]==666),2]<-0
R4RNA::helixToVienna(R4RNA::as.helix(ve,nchar(nucvec)))->vienna

# motif with indicators for bulges
counterind<-1
unlist(strsplit(momo,""))->splitmomo
length(r)->rlen
((2+(rlen)+(2*rlen)+rlen)-1)->vlen
rep(1,time=vlen)->vv
r+ww->rw
length(splitmomo)->lenmomo
paste(splitmomo[1:ww[1]],collapse="")->vv[1]

for(t in seq_along(ww)){
  vv[counterind+1]<-"["
  vv[counterind+2]<- paste(splitmomo[(ww[t]+1):(rw[t])],collapse="")
  vv[counterind+3]<-"]"
  ifelse(t<max(seq_along(ww)),
         yes=(vv[counterind+4]<-paste(splitmomo[(rw[t]+1):(ww[t+1])],collapse="")),
         no=(paste(splitmomo[(max(rw)+1):lenmomo],collapse="")->vv[vlen]))
  counterind+4->counterind
}
paste(vv,collapse="")->momo2

AAAaa[ww,8]->wwe # real bluge position in motif w/o bulges
AAAaa[ww,1]->wwe2
rep(amountbulge,times=length(r))->amo
rep(mostart,times=length(r))->mosa
rep(moend,times=length(r))->moen
rep(lennuc,times=length(r))->lennu
r->rnew
rnew+wwe2->wwe2end

as.character(zzzi[i,3])->ctfis
as.character(zzzi[i,4])->fastanames
as.numeric(zzzi[i,2])->indexs

unlist(strsplit(x=nucvec,NULL))->nui
paste(c(nui[1:(mostart-1)],"(",nui[(mostart):moend],")",
        nui[(moend+1):lennuc]),collapse="")->comp

bulge_vector<-rep(NA,times=length(ww))
for (g in seq_along(ww)){
  paste(AAAaa[c((ww[g]+1):(ww[g]+r[g])),2],collapse="",sep="")->bulge_vector[g]
}
```

```
    # every bulge loop needs all the info in the final data frame
    rep(lennuc,length(r))->lennucs
    rep(propaired,length(r))->propairds
    rep(mostart,length(r))->mostarts
    rep(moend,length(r))->moends
    rep(maxres,length(r))->maxress
    rep(momo2,length(r))->momos
    rep(ctfis,length(r))->ctfiss
    rep(fastanames,length(r))->fastanamess
    rep(comp,length(r))->comps
    rep(indexs,length(r))->ii
    rep(amountbulge,length(r))->amountbulges
    rep(AAABseq,length(r))->AAABseqs
    rep(AAAfseq,length(r))->AAAfseqs
    rep(mo_names,length(r))->mo_namess
    rep(vienna,length(r))->viennas

    #creating final data frame for motives with bulges
    cbind(nunu,wwe,r,bulge_vector,dGs,lennucs,propairds,mostarts,moends,momos,maxress,
          fastanamess,ctfiss,comps,ii,amountbulges,AAABseqs,AAAfseqs,
          mo_namess,viennas)->results2
    colnames(results2)<-headercol
    rbind(results1meme,results2)->results1meme

  }else{

    # collecting everything for motives without bulges
    paste(AAss,collapse="",sep="")->nucvec
    colnames(AAAcc)->string
    strsplit(sub(".*dG = ", "", string[1])," ")->qwer
    unlist(qwer)->qwer
    qwer[1]->dG

    length(AAAcc[,7])->lennuc
    round(100/(length(AAAcc[,7]))*(length(AAAcc[AAAcc[,7]>0,7])),2)->propaired
    paste(AAAcc[mostart:moend,2],collapse="",sep="")->momo
    as.character(zzzi[i,3])->ctfi
    as.character(zzzi[i,4])->fastaname
    as.numeric(zzzi[i,2])->indexsi

    unlist(strsplit(x=nucvec,NULL))->nui
    paste(c(nui[1:(mostart-1)],"(",nui[(mostart):moend],")",
            nui[(moend+1):lennuc]),collapse="")->comp

    AAAg[[i]]->ve
    ve[which(ve[,2]==666),2]<-0
    R4RNA::helixToVienna(as.helix(ve,nchar(nucvec)))->vienna

    c(nucvec,0,0,"none",dG,lennuc,propaired,mostart,moend,momo,maxres,
      fastaname,ctfi,comp,indexsi,0,AAABseq,AAAfseq,mo_names,vienna)->results2
    rbind(results1meme,results2)->results1meme
  } # end of loop for bulges
} # end of master loop
```

```
  # final modifications of result data frame
  as.double(which(is.na(results1meme[,1])))->outNA
  results1meme[-(outNA),]->results1meme
  sapply(results1meme[,c(2,3,5,6,7,8,9,11,15,16)],
         as.numeric)->results1meme[,c(2,3,5,6,7,8,9,11,15,16)]
  results1meme->results1list[[lindex]]


### PART 5 ###################################################################
  #selection process

  results1meme->part
  ifelse(choic==1,yes="dG",no="alignment score")->choice
  part[order(part[choice],decreasing=FALSE),]->part2
  part2[!duplicated(part2$index),]->part3 #remove duplicate index
  part3[!duplicated(part3$sequence),]->part4
  subset(part, part$index %in% part4$index)->part5 #re-add lost bulges
  part5->results3
  results3->results3_list[[lindex]]



### PART 6 ###################################################################
  # depicting the results

  theme_set(theme_classic())
  as.numeric(results3[,2])->results3[,2]
  as.factor(results3[,3])->results3[,3]
  g <- ggplot(results3, aes(`bulge position`)) +
    scale_fill_viridis(option="magma",
                       discrete=TRUE,begin=0.1)
  g<-g + geom_histogram(aes(fill=`bulge size`),
                        binwidth = .5,
                        col="black",
                        size=.1) +  # change binwidth

        theme(axis.text=element_text(size = 13),
          text = element_text(size = 13),
          axis.ticks.x = element_blank(),
          axis.line=element_blank(),
          panel.grid.major.y=element_line(colour = "black"))+
    scale_x_continuous(name="bulge position",
                       breaks=0:10,
                       labels=c("no\nloop","1","2","3","4","5","6","7","8","9","10"))+
    guides(fill=guide_legend(title="bulge size"))+
    scale_y_continuous(name="counts",
                       limits=c(0,90),
                       breaks=seq(0,90,by=5))+
    geom_vline(aes(xintercept=0.5))

  g->plot_list[[lindex]]
  ggseqlogo::ggseqlogo(moco2[[lindex]])->logo_list2[[lindex]]
```

```
}


# create lists after selection
results3_list[[1]]->res3_1
results3_list[[2]]->res3_2
rbind(res3_1,res3_2)->results4
results4[!duplicated(results4[,c(1,12)]),]->res32fasta

#include fastas without motifs for summary reasons
k<-1
results4->results5
for (k in 1:nrow(yes_no_df)){
  if(yes_no_df$"NA"[k]==1){
    which(zzzi[,1]%in%as.character(yes_no_df[k,1]))->ind
    as.character(zzzi[ind,3])->ctfis
    as.character(zzzi[ind,4])->fastanames
    as.numeric(zzzi[ind,2])->indexs

    AAA[[ind]]->AAAout

    AAAout[,5]->D
    rep(0,times=length(D))->k
    for(l in seq_along(D)){
      if (D[l]==0){
        ifelse(l==1&D[l]==0,yes=(k[1]<-1),no=(k[l-1]+1->k[l]))
      }else{0->k[l]}
    }
    as.data.frame(AAAout)->AAAcc
    cbind(AAAcc,k)->AAAcc
    round(100/(length(AAAcc[,7]))*(length(AAAcc[AAAcc[,7]>0,7])),2)->propaired
    nrow(AAAout)->lennuc
    # dG values
    colnames(AAAout)->string
    strsplit(sub(".*dG = ", "", string[1])," ")->qwer
    unlist(qwer)->qwert
    qwert[1]->dG

    AAAout[,2]->AASss
    Biostrings::DNAString(paste(AASss,collapse="",sep=""))->AAss
    paste(AAss,collapse="",sep="")->nucvec

    AAAB[[ind]][,2]->AASSS
    Biostrings::DNAString(paste(AASSS,collapse="",sep=""))->AASSS
    AAABseq<-paste(AASSS,collapse="",sep="")

    AAAf[[ind]]->AAAfseq_df
    AAAfseq_df[,8]->AAAfseq_pre
    as.vector(AAAfseq_pre)->AAAfseq_pre
    paste(AAAfseq_pre,collapse="",sep="")->AAAfseq

    AAAg[[ind]]->ve
    ve[which(ve[,2]==666),2]<-0
```

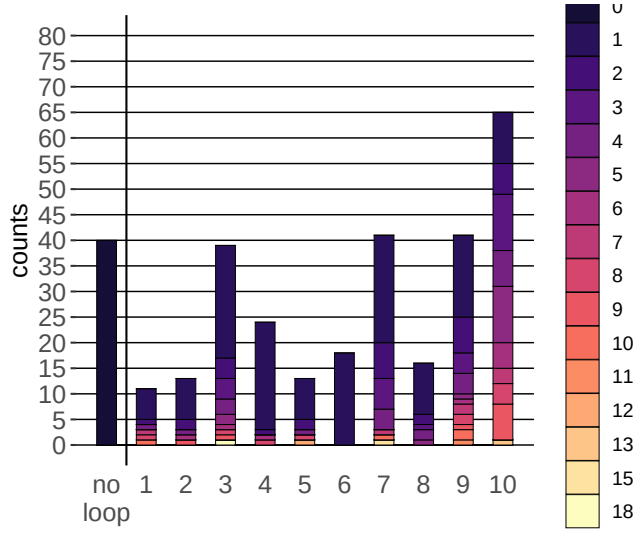
```
    R4RNA::helixToVienna(R4RNA::as.helix(ve,nchar(nucvec)))->vienna

    c(nucvec,NA,NA,NA,dG,lennuc,propaired,NA,NA,NA,NA,fastanames,ctfis,NA,indexs,NA,
    AAABseq,AAAfseq,NA,vienna)->results_temp
    rbind(results5,results_temp)->results5
  }
}


seqRFLP::dataframe2fas(res32fasta$"sequence", file="wobulgeselec.txt")
seqRFLP::dataframe2fas(res32fasta$"seq w/o unpaired", file="wobulgeselec2.txt")

Biostrings::readDNAStringSet("wobulgeselec.txt",use.names=TRUE)->fafa_res3_1
Biostrings::readDNAStringSet("wobulgeselec2.txt",use.names=TRUE)->fafa_res3_2

unlink("wobulgeselec.txt")
unlink("wobulgeselec2.txt")
TFBSTools::runMEME(fafa_res3_1,seqtype="DNA", arguments=MEME_arguments2)->mo3
TFBSTools::runMEME(fafa_res3_2,seqtype="DNA", arguments=MEME_arguments2)->mo4

Biostrings::consensusMatrix(mo3)->moco3
ggseqlogo::ggseqlogo(moco3)->gmoco3

Biostrings::consensusMatrix(mo4)->moco4
ggseqlogo::ggseqlogo(moco4)->gmoco4


logo_list3<-list()
ggseqlogo::ggseqlogo(moco3[[2]])->logo_list3[[1]]
ggseqlogo::ggseqlogo(moco3[[1]])->logo_list3[[2]]

logo_list4<-list()
ggseqlogo::ggseqlogo(moco4[[1]])->logo_list4[[1]]
ggseqlogo::ggseqlogo(moco4[[2]])->logo_list4[[2]]

cowplot::plot_grid(plotlist=c(logo_list1,
                              rev(logo_list0),
                              logo_list4),
                   ncol=2,
                   nrow=3,
                   labels=c("A","","B","","C","")
                   )
```



Caption: The sequence logos of the motifs found by MEME at different stages of the analysis. (A) DUX4 logos from the JASPAR database (left logo) and its reverse complementary counterpart (right logo). (B) Logos found by MEME at the initial run of PART 1 of the R script. (C) Logos found by MEME after the final run with data in which all base-paired regions were removed during PART 3 and single foldings per aptamer has been selected for either the lowest Gibb's free energy or MEME score during PART 5 of the R script.

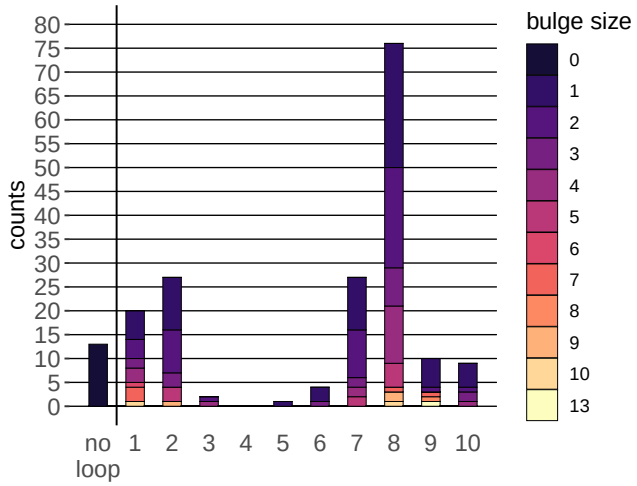
```
#forward
ggseqlogo::ggseqlogo(moco4[[1]])->bulge_conserve_logof

# reverse
moco4[[2]][,11:1]->revmoco2
ggplot() + geom_logo(data = revmoco2) +
  theme_logo() +
  scale_y_reverse()->bulge_conserve_logor
#bulge_conserve_logor

#forward
results3_list[[1]]->results3
theme_set(theme_classic())
as.numeric(results3[,2])->results3[,2]
as.factor(results3[,3])->results3[,3]
g <- ggplot(results3, aes(`bulge position`)) +
  scale_fill_viridis(option="magma",discrete=TRUE,begin=0.1)
g<-g + geom_histogram(aes(fill=`bulge size`),
                      binwidth = .5,
                      col="black",
                      size=.1) +  # change binwidth
    theme(axis.text=element_text(size = 13),
        text = element_text(size = 13),
```

```
        axis.ticks.x = element_blank(),
        axis.line=element_blank(),
        panel.grid.major.y=element_line(colour = "black"))+
  scale_x_continuous(name="",
                     breaks=0:10,
                     labels=c("no\nloop","1","2","3","4","5","6","7","8","9","10"))+
  guides(fill=guide_legend(title="bulge size"))+
  scale_y_continuous(name="counts", limits=c(0,80),breaks=seq(0,80,by=5))+
  geom_vline(aes(xintercept=0.5))
g
```




```
#reverse
results3_list[[2]]->results3
as.factor(results3[,3])->results3[,3]
g <- ggplot(results3, mapping=aes(x=`bulge position`)) +

  geom_histogram(aes(fill=`bulge size`),
                 binwidth = .5,
                 col="black",
                 size=.1) +
  scale_fill_viridis(option="magma",discrete=TRUE,begin=0.1)+
  scale_x_reverse(name="",
                  position="top",
                  breaks=0:10,
                  labels=c("no\nloop","10","9","8","7","6","5","4","3","2","1"))
g<-g + scale_y_reverse(name="counts", limits=c(80,0),breaks=seq(0,80,by=5))
g<-g + theme(axis.text=element_text(size = 13),
             text = element_text(size = 13),
             axis.ticks.x = element_blank(),
             axis.line=element_blank(),
             panel.grid.major.y=element_line(colour = "black"))
g<-g + guides(fill=guide_legend(title="bulge size"))
g<-g + geom_vline(aes(xintercept=0.5))
```

```
#reverse
results3_list[[2]]->results3
as.factor(results3[,3])->results3[,3]
g <- ggplot(results3, mapping=aes(x=`bulge position`)) +

  geom_histogram(aes(fill=`bulge size`),
                 binwidth = .5,
                 col="black",
                 size=.1) +
  scale_fill_viridis(option="magma",discrete=TRUE,begin=0.1)+
  scale_x_continuous(name="",
                     position="bottom",
                     breaks=0:10,
                     labels=c("no\nloop","1","2","3","4","5","6","7","8","9","10"))
g<-g + scale_y_continuous(name="counts", limits=c(0,80),breaks=seq(0,80,by=5))
g<-g + theme(axis.text=element_text(size = 13),
             text = element_text(size = 13),
             axis.ticks.x = element_blank(),
             axis.line=element_blank(),
             panel.grid.major.y=element_line(colour = "black"))
g<-g + guides(fill=guide_legend(title="bulge size"))
g<-g + geom_vline(aes(xintercept=0.5))
g
```



Bulge sequence conservation

Circular plots are created to visualize bulge sequence conservation. Motif and position have to be set to plot the results of a single bulge loop location.

```
# Parameters:
mindex<-2 # which motif i.e. #2
bupo<-8 # which position to look at i.e position 7

results3_list[[mindex]]->results3
ins<-moco4[[mindex]]
bulgesize<-4
ma<-ins
sum(ma[,1])->maxscore
```

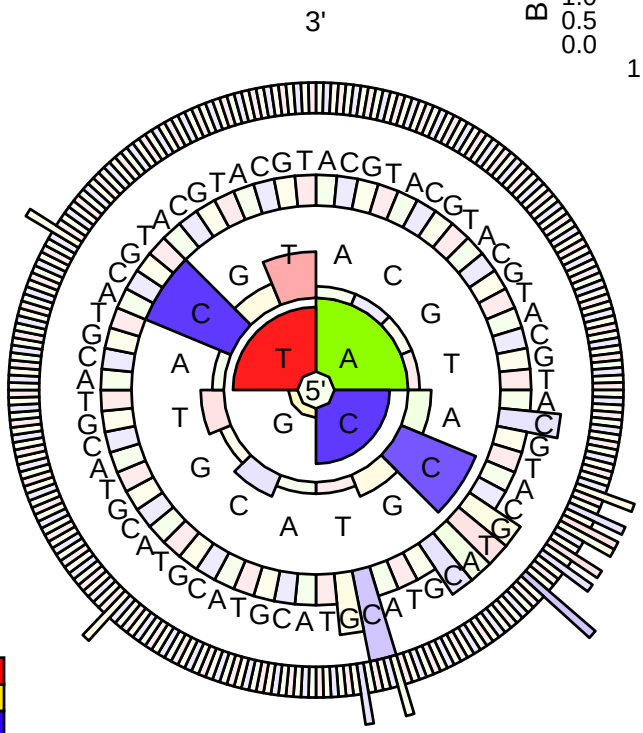
```
xx <- c('A', 'C', 'G', 'T')
permlist<-list()
for(i in 1:bulgesize){
  permutations(n=4,r=i,v=xx,repeats.allowed=TRUE)->perm
  permlist[[i]]<-perm
}
permlist

list()->collapsepermlist
count<-1
for(j in permlist){
  j->perm
  as.data.frame(perm)->perm
  apply(perm,1,paste,collapse="")->collapsepermlist[[count]]
  count+1->count
}
vector()->collapsepermlistvec
for (k in collapsepermlist){
  c(collapsepermlistvec,k)->collapsepermlistvec
}
as.data.frame(collapsepermlistvec)->collapsepermlistvec

as.vector(results3[results3$`bulge position`==bupo,4])->to_count
sort(unique(to_count))->vecchar
veccharvec<-1->count
for (i in vecchar){
  sum(lengths(regmatches(to_count,gregexpr(paste0("^",i,"$",collapse=""),to_count))))->veccharvec[count]
  count+1->count
}

cbind(vecchar,veccharvec)->vecsummary
as.data.frame(vecsummary)->vecsummary
vecsummary$vecchar<-as.character(vecsummary$vecchar)
vecsummary$veccharvec<-as.numeric(as.character((vecsummary$veccharvec)))
c("seq","value")->colnames(vecsummary)
collapsepermlistvec$value<-0
"seq"->colnames(collapsepermlistvec)[1]

for ( i in 1:nrow(collapsepermlistvec)){
  ifelse(collapsepermlistvec[i,1]%in%vecsummary[,1],
    yes=(vecsummary[which(collapsepermlistvec[i,1]==vecsummary[,1]),2]->collapsepermlistvec[i,2]),
    no= 0->collapsepermlistvec[i,2] )
}


collapsepermlistvec->dftest
for(i in 1:nrow(dftest)){
  if(grepl("A$", as.character(dftest[i,1]))==TRUE){
    0.24->hue
  }else{
    if(grepl("C$", as.character(dftest[i,1]))==TRUE){
      0.7->hue
    }else{
```

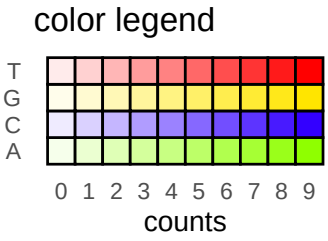
```
      if(grepl("G$", as.character(dftest[i,1]))==TRUE){
        0.15->hue
      }else{
        if(grepl("T$", as.character(dftest[i,1]))==TRUE){
          0->hue
        }else{
          0.8->hue
        }
      }
    }
  }

  max(dftest$value)->maxdftest
  (1/maxdftest*dftest[i,2])->lumi
  hue->dftest$hue[i]
  lumi->dftest$lumi[i]
}
dftest[order(as.character(dftest$seq), -rank(as.character(dftest$seq)),
             decreasing = FALSE), ]->dftest

for(i in 1:nrow(dftest)){
  if(dftest$lumi[i]==0){
    dftest$lumi[i]<-0.08
  }
}


for (i in 1:nrow(dftest)){
  hsv(dftest[i,3], dftest[i,4],1,1)->dftest$col[i]
}


### INSET table ########################
collapsepermlistvec[order(collapsepermlistvec$value,decreasing=TRUE),]->summary_table
summary_table[1:5,]->summary_table
grobt<-ggplotGrob(ggtexttable(summary_table,rows=c(),cols=c("seq","counts")))


### main plot ########################
(collapsepermlistvec$value)+1->collapsepermlistvec$value

main_plot<-ggplot()+
  geom_rect(data=collapsepermlistvec[340,],
        mapping=aes(fill=collapsepermlistvec[85,1],
            ymax=20, ymin=0, xmax=1, xmin=0))+
  geom_rect(data=collapsepermlistvec[1:4,],
        mapping=aes(fill=collapsepermlistvec[1:4,1],
            ymax=c(5,10,15,20), ymin=c(0,5,10,15),
            xmax=(1/max(collapsepermlistvec[1:4,2])*collapsepermlistvec[1:4,2])*5,
            xmin=1),
            color="black")+
  geom_rect(data=collapsepermlistvec[5:20,],
       mapping=aes(fill=collapsepermlistvec[5:20,1],
```

```
            ymax=c(seq(from=1.25,to=20,by=1.25)),
            ymin=c(seq(from=0,to=20-1.25,by=1.25)),
            xmax=5+((1/max(collapsepermlistvec[5:20,2])*collapsepermlistvec[5:20,2])*5),
            xmin=5),color="black")+
  geom_rect(data=collapsepermlistvec[21:84,],
        mapping=aes(fill=collapsepermlistvec[21:84,1],
            ymax=c(seq(from=0.3125,to=20,by=0.3125)),
            ymin=c(seq(from=0,to=20-0.3125,by=0.3125)),
            xmax=10+((1/max(collapsepermlistvec[21:84,2])*collapsepermlistvec[21:84,2])*5),
            xmin=10),color="black")+
  geom_rect(data=collapsepermlistvec[85:340,],
        mapping=aes(fill=collapsepermlistvec[85:340,1],
            ymax=c(seq(from=0.078125,to=20,by=0.078125)),
            ymin=c(seq(from=0,to=20-0.078125,by=0.078125)),
            xmax=15+((1/max(collapsepermlistvec[85:340,2])*collapsepermlistvec[85:340,2])*5),
            xmin=15),color="black")+
  scale_fill_manual(values=c(dftest$col),
                    guide=FALSE)+
  coord_polar(theta="y")+
  annotate("text", x = 2.5, y = c(2.5,7.5,12.5,17.5),
           label=c('A', 'C', 'G', 'T'))+
  annotate("text", x = 7.5, y = seq(from=0.625,to=20-0.625, by=1.25),
           label=rep(c('A', 'C', 'G', 'T'),times=4))+
  annotate("text", x = 12.5, y = seq(from=0.15625,to=20-0.15625, by=0.3125),
           label=rep(c('A', 'C', 'G', 'T'),times=16))+
  annotate("text", x = 17.5, y = seq(from=0.0390625,to=20-0.0390625, by=0.078125),
           label=rep(c(' ', ' ', ' ', ' '),times=64),size=1)+
  annotate("text", x = 0, y = 0, label="5'")+
  annotate("text", x = 20, y = 0, label="3'")+
  theme_void()


### INSET 2 ########################
ggplot() +
  geom_logo(data = ma) +
  theme_logo() +
  geom_vline(aes(xintercept=bupo+0.5),color="black",size=2)->inset2


### INSET 3 ########################
box.size <- 0.5
seq(0.08,1,length.out = maxdftest+1)->lumi_seq
seq(-0.5,maxdftest+0.5,length.out = maxdftest+2)->x_seq
df <- data.frame(xmin = c(NA),
                 xmax = c(NA),
                 ymin = c(NA),
                 ymax = c(NA),
                 fill = c(NA))

for(i in 1:(maxdftest+1)){


  hue_tempA<-0.24
```

```
  hue_tempC<-0.7
  hue_tempG<-0.15
  hue_tempT<-0

  hsv(hue_tempA, lumi_seq[i],1,1)->col_tempA
  hsv(hue_tempC, lumi_seq[i],1,1)->col_tempC
  hsv(hue_tempG, lumi_seq[i],1,1)->col_tempG
  hsv(hue_tempT, lumi_seq[i],1,1)->col_tempT

  df1 <- data.frame(xmin = rep(x_seq[i], 4),
                    xmax = rep(x_seq[i+1] , 4),
                    ymin = (seq(1:4)-1) * box.size,
                    ymax = seq(1:4) * box.size,
                    fill = c(col_tempA, col_tempC, col_tempG, col_tempT))

  rbind(df,df1)->df

}
df[-1,]->df
inset3<-ggplot(df) +
  geom_rect(aes(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax, fill = fill),
            color="black") +
  scale_fill_identity()+
  scale_y_continuous(breaks=c(0.25,0.75,1.25,1.75), labels=c("A","C","G","T"))+
  scale_x_continuous(name ="counts",breaks=c(0:(maxdftest)))+
  theme(axis.line=element_blank(),
        axis.ticks = element_blank()
  )+
  labs(title="color legend")



### all together ########################
cowplot::ggdraw() +
  draw_plot(main_plot) +
  draw_plot(inset3, x = 0, y = 0, width = 0.28, height = 0.28)+
  draw_plot(inset2, x = 0.65, y = .78, width = 0.35, height = 0.22)+
  draw_plot(grobt, x = -0.1, y = .55, width = 0.4, height = 0.5,scale=1)
```

seq	counts
A	9
T	8
C	7
TC	7
CC	6



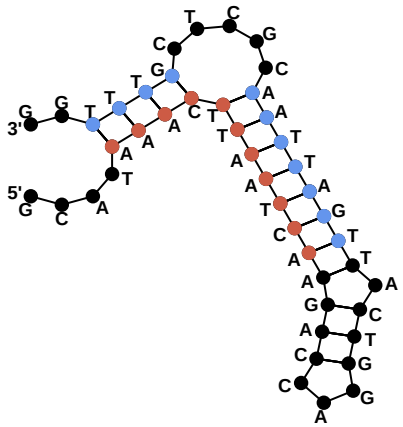


Structure visualization

Single structures can be visualized by using the RRNA package and the corresponding index of the structure. Here a function called "struplo" was created to visualize structures and highlight protein binding sites.

```
#parameters of function:
  #ID from analysis above
  #width and height for output
  #re and bl are the positions of the forward and reverse motifs, e.g. c(4:10)
  #recol and blcol are the colors of the motifs, e.g. "blue"

struplo<-function(ID,width,height,re,bl,recol,blcol){

  AAAe[[ID]]->AAAad
  AAAad[which(AAAad[,5]==666),5]<-0
  write.table(AAAad,file="test.cr",sep=" ",row.names = FALSE)
  RRNA::ct2coord(loadCt("test.cr"))->terr
  unlink("test.cr")

  RRNA::RNAPlot(terr,
           add=FALSE,
           nt=TRUE,main="",
           tsize=0.6,
           lineWd=1)
  RRNA::RNAPlot(terr,
```

```
          add=TRUE,
          nt=FALSE,
          main="",
          modspec = TRUE,
          modp = c(re,bl),
          mod = c(rep(16,times=length(re)),rep(16,times=length(bl))),
          modcol = c(rep(recol,times=length(re)),rep(blcol,times=length(bl))),
          pointSize = 1,
          lineWd=1)


}

# Structures depicted in this work:

struplo(1,7,9,c(5:15),c(28:34,40:43),"coral3","cornflowerblue")
```




```
#struplo(334,7,12,c(2:9,12:14),c(31:37,40:43),"coral3","cornflowerblue")
#struplo(112,5,10,c(16:26),c(32:38,41:44),"coral3","cornflowerblue")
#struplo(445,5,10,c(29:39),c(1:4,18:25),"coral3","cornflowerblue")
#struplo(35,6,11,c(6:13,15:17),c(24:34),"coral3","cornflowerblue")
#struplo(24,7,10,c(29:39),c(3:9,13:16),"coral3","cornflowerblue")
#struplo(79,5,8,c(27:37),c(),"coral3","cornflowerblue")
#struplo(57,6,10,c(5:15),c(25:31,35:38),"coral3","cornflowerblue")
#struplo(113,3.5,6,c(2:12),c(25:31,36:39),"coral3","cornflowerblue")
```

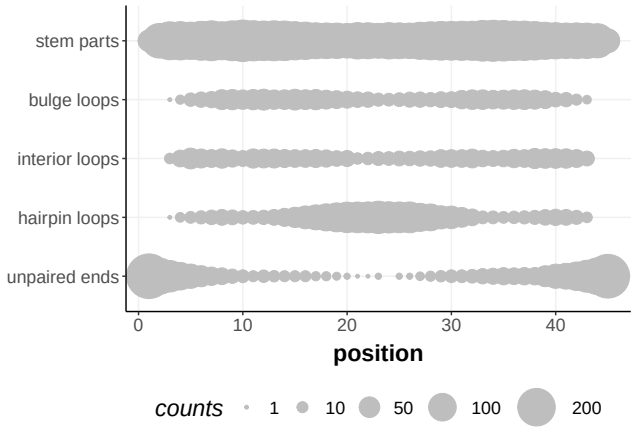
Visualization of structural element conservation

The next code snippet was used to show the conservation of the different structural elements: stem parts, bulge loops, interior loops, hairpin loops, and unpaired ends.

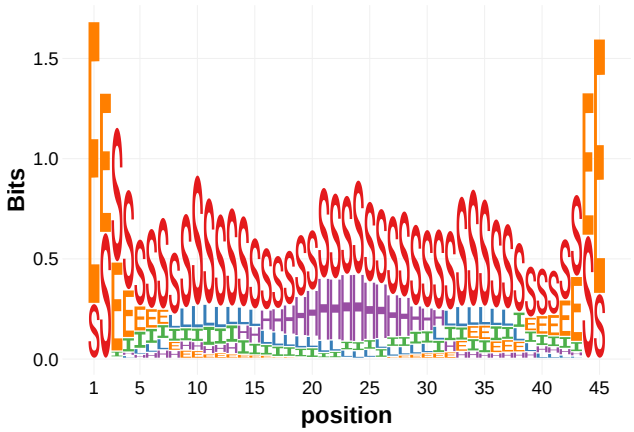
Two different types of visualizations were used: as dot plot where circle size depicts the amount of structural elements per position or as logo.

```
# visualization as plot
results5->part
choic<-1
ifelse(choic==1,yes="dG",no="alignment score")->choice
part[order(part[choice],decreasing=FALSE),]->part2
part2[!duplicated(part2$index),]->part3 #remove duplicate index
part3[!duplicated(part3$sequence),]->part4 #remove duplicate sequences with higher dG
```

```
subset(part,  part$index %in% part4$index)->part5

part4[,c("fasta","sequence",  "structure")]->part4sub
part4sub[,c("fasta","structure")]->part4subsub


for(i in 1:nrow(part4subsub)){
  gsub("B","L",part4subsub[i,2])->part4subsub[i,2]

}


part4subsubma<-matrix(1,nrow(part4subsub),45)
for(i in 1:nrow(part4subsub)){
  unlist(strsplit(part4subsub[i,2],""))->part4subsubma[i,]
}
as.data.frame(part4subsubma)->part4subsubdf
colnames(part4subsubdf)<-1:45

tidyr::gather(part4subsubdf)->part4subsubdfga
as.numeric(part4subsubdfga[,1])->part4subsubdfga[,1]
ggplot(data=part4subsubdfga,aes(x=key, y=value))+
  geom_count(color="grey")+
  labs(x="position",y="",size="counts")+
  scale_y_discrete(labels=c("unpaired ends",
                            "hairpin loops",
                            "interior loops",
                            "bulge loops",
                            "stem parts"))+
  scale_size_continuous(breaks=c(1,10,50,100,200),range=c(1,15))+
  theme(axis.title = element_text(face = "bold",size = rel(1.5)),
        axis.title.x = element_text(vjust = -0.2),
        axis.title.y = element_text(),
        axis.text.y = element_text(size=rel(1.5)),
        axis.text.x = element_text(size=rel(1.5)),
        legend.text=element_text(size=rel(1.2)),
        panel.grid.major = element_line(colour="#f0f0f0"),
        legend.position = "bottom",
        legend.direction = "horizontal",
        legend.spacing = unit(0, "cm"),
        legend.title = element_text(face="italic",size=rel(1.5)),
        plot.margin=unit(c(10,5,5,5),"mm")
)
```




```
# visualization as logo
resdf<-matrix(0,5,ncol(part4subsubdf))
as.data.frame(resdf)->resdf
rownames(resdf)<-c("S","L","I","H","E")
for(i in 1:ncol(part4subsubdf)){
  paste(part4subsubdf[,i],collapse="",sep="")->outv
  stringr::str_count(outv,"S")->resdf[1,i]
  stringr::str_count(outv,"L")->resdf[2,i]
  stringr::str_count(outv,"I")->resdf[3,i]
  stringr::str_count(outv,"H")->resdf[4,i]
  stringr::str_count(outv,"E")->resdf[5,i]

}

brewer.pal(5,"Set1")->pal
cs1 = make_col_scheme(chars=c("S","L","I","H","E"), cols=pal)
ggseqlogo(as.matrix(resdf),seq_type='aa',namespace=c("S","L","I","H","E"), col_scheme=cs1)+
  theme(axis.title = element_text(face = "bold",size = rel(1.5)),
        axis.title.x = element_text(vjust = -0.2),
        axis.title.y = element_text(),
        axis.text.y = element_text(size=rel(1.5)),
        axis.text.x = element_text(size=rel(1.5)),
        panel.grid.major = element_line(colour="#f0f0f0"),
        legend.position = "bottom",
        legend.direction = "horizontal",
        legend.spacing = unit(0, "cm"),
        legend.title = element_text(face="italic",size=rel(1.5)),
        plot.margin=unit(c(10,5,5,5),"mm"))+
  labs(x="position")+
  scale_x_continuous(breaks=c(1,5,10,15,20,25,30,35,40,45))
```




```
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: CentOS Linux 7 (Core)
##
## Matrix products: default
## BLAS/LAPACK: /scicore/soft/apps/OpenBLAS/0.3.1-GCC-7.3.0-2.30/lib/libopenblas_sandybridgep-r0.3.1.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
##  [1] grid      stats4    parallel  stats     graphics  grDevices utils
##  [8] datasets  methods   base
##
## other attached packages:
##  [1] RColorBrewer_1.1-2   tidyr_0.8.3          stringr_1.4.0
##  [4] reshape2_1.4.3       VennDiagram_1.6.20   futile.logger_1.4.3
##  [7] seqRFLP_1.0.1        cowplot_0.9.4        gridExtra_2.3
## [10] ggpubr_0.2           magrittr_1.5         DNAshapeR_1.10.0
## [13] GenomicRanges_1.34.0 GenomeInfoDb_1.18.2  gtools_3.8.1
## [16] RRNA_1.0             viridis_0.5.1        viridisLite_0.3.0
## [19] ggplot2_3.2.0        JASPAR2014_1.18.0    TFBSTools_1.20.0
## [22] ggseqlogo_0.1        R4RNA_1.10.0         Biostrings_2.50.2
## [25] XVector_0.22.0       IRanges_2.16.0       S4Vectors_0.20.1
## [28] BiocGenerics_0.28.0  tinytex_0.16         knitr_1.22
##
## loaded via a namespace (and not attached):
##  [1] bitops_1.0-6                matrixStats_0.54.0
##  [3] DirichletMultinomial_1.24.1 bit64_0.9-7
##  [5] httr_1.4.0                  tools_3.5.2
##  [7] R6_2.4.0                    seqLogo_1.48.0
##  [9] DBI_1.0.0                   lazyeval_0.2.2
## [11] colorspace_1.4-1            withr_2.1.2
## [13] tidyselect_0.2.5            bit_1.1-14
## [15] compiler_3.5.2              Biobase_2.42.0
```

```
## [17] formatR_1.6                DelayedArray_0.8.0
## [19] labeling_0.3               rtracklayer_1.42.2
## [21] caTools_1.17.1.2           scales_1.0.0
## [23] readr_1.3.1                digest_0.6.19
## [25] Rsamtools_1.34.1           rmarkdown_1.12
## [27] R.utils_2.8.0              pkgconfig_2.0.2
## [29] htmltools_0.3.6            maps_3.3.0
## [31] BSgenome_1.50.0            rlang_0.3.4
## [33] RSQLite_2.1.1              VGAM_1.1-1
## [35] BiocParallel_1.16.6        dplyr_0.8.0.1
## [37] R.oo_1.22.0                RCurl_1.95-4.12
## [39] GO.db_3.7.0                GenomeInfoDbData_1.2.0
## [41] dotCall64_1.0-0            Matrix_1.2-15
## [43] Rcpp_1.0.1                 munsell_0.5.0
## [45] R.methodsS3_1.7.1          stringi_1.4.3
## [47] yaml_2.2.0                 SummarizedExperiment_1.12.0
## [49] zlibbioc_1.28.0            plyr_1.8.4
## [51] blob_1.1.1                 crayon_1.3.4
## [53] CNEr_1.18.1                lattice_0.20-38
## [55] splines_3.5.2              annotate_1.60.0
## [57] hms_0.4.2                  KEGGREST_1.22.0
## [59] pillar_1.4.1               futile.options_1.0.1
## [61] TFMPvalue_0.0.8            XML_3.98-1.19
## [63] glue_1.3.1                 packrat_0.5.0
## [65] evaluate_0.13              lambda.r_1.2.3
## [67] spam_2.2-2                 png_0.1-7
## [69] gtable_0.3.0               poweRlaw_0.70.2
## [71] purrr_0.3.2                assertthat_0.2.1
## [73] xfun_0.6                   xtable_1.8-4
## [75] tibble_2.1.1               GenomicAlignments_1.18.1
## [77] AnnotationDbi_1.44.0       memoise_1.1.0
## [79] fields_9.6.1
```