R Notebook

In this exercise we try to identify operons in E. coli by searching for binding sites of the sigma70 binding factor, which is necessary for the initiation of transcription in most E. coli operons.

The binding site is defined by two motifs (10 and 35 nucleotides upstream from the START codon, ATG) and a spacer 13-19 nucleotides long.

The position weight matrices for this motif were taken from:

http://2013.igem.org/Team:XMU_Software/Project/promoter
(http://2013.igem.org/Team:XMU_Software/Project/promoter)

Loading the required library (Biostrings) and the genomes of several strains of E. coli

```
require(Biostrings)
```

```
## Loading required package: Biostrings
```

```
## Loading required package: BiocGenerics
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, cbind, colMeans,
##     colnames, colSums, do.call, duplicated, eval, evalq, Filter,
##     Find, get, grep, grepl, intersect, is.unsorted, lapply,
##     lengths, Map, mapply, match, mget, order, paste, pmax,
##     pmax.int, pmin, pmin.int, Position, rank, rbind, Reduce,
##     rowMeans, rownames, rowSums, sapply, setdiff, sort, table,
##     tapply, union, unique, unsplit, which, which.max, which.min
```

```
## Loading required package: S4Vectors
```

```
## Loading required package: stats4
```

```
##
## Attaching package: 'S4Vectors'
```

```
## The following object is masked from 'package:base':
##
##     expand.grid
```

```
## Loading required package: IRanges
```

```
## Loading required package: XVector
```

```
##
## Attaching package: 'Biostrings'
```

```
## The following object is masked from 'package:base':
##
##     strsplit
```

```
require(BSgenome.Ecoli.NCBI.20080805)
```

```
## Loading required package: BSgenome.Ecoli.NCBI.20080805
```

```
## Loading required package: BSgenome
```

```
## Loading required package: GenomeInfoDb
```

```
## Loading required package: GenomicRanges
```

```
## Loading required package: rtracklayer
```

Loading the genome of E. coli, strain K-12 substrain MG1655

```
eColi = Ecoli$NC_000913
```

Loading the motifs - tataat is the (-10) motif, and ttgaca is the (-35) motif. The variables are named after the consensus sequence for the motif.

```
#loading the matrices with frequency of occurence of each nucleotide at positions in the moti
f from files
#rownames = 1 means that the first line of the file will be the names of the rows

tataat = read.table("TATAAT.csv", row.names = 1)
ttgaca = read.table("TTGACA.csv", row.names = 1)

tataat
```

	V2 <int>	V3 <int>	V4 <int>	V5 <int>	V6 <int>	V7 <int>	V8 <int>	V9 <int>
A	18	19	1	116	35	52	44	7
C	19	24	15	0	13	24	28	3
G	53	26	3	0	17	12	26	0
T	26	47	97	0	51	28	18	106

4 rows

```
#converting these matrices into probability matrices

tataatPWM = PWM(as.matrix(tataat[-c(1, 2)]))
ttgacaPWM = PWM(as.matrix(ttgaca[-c(1, 2, 3)]))

tataatPWM
```

```
##            V4         V5         V6         V7         V8          V9
## A -0.03381054  0.1901632 0.13119863 0.15064670 0.14243487  0.05305220
## C  0.08979535 -0.1133393 0.08284864 0.11271539 0.12025976  0.01340508
## G  0.01340508 -0.1133393 0.09588477 0.07897104 0.11663141 -0.11333926
## T  0.18134489 -0.1133393 0.14969181 0.12025976 0.09866939  0.18571851
```

Looking for the positions of the motifs in the genome using the function matchPWM:

minscore is the minimum score a match needs to have in order to be counted It was given here as the percentage of the highest possible score

```
#define minscore cutoff
minscore = "80%"

#find positions
tataatPos = matchPWM(pwm = tataatPWM, subject = eColi, min.score = minscore)
ttgacaPos = matchPWM(pwm = ttgacaPWM, subject = eColi, min.score = minscore)

#look at the acquired result
tataatPos
```

```
##   Views on a 4639675-letter DNAString subject
## subject: AGCTTTTCATTCTGACTGCAACGGGCAATATG...CAAATAAAAAACGCCTTAGTAAGTATTTTTC
## views:
##             start     end width
##      [1]       8      13     6 [CATTCT]
##      [2]      26      31     6 [CAATAT]
##      [3]      46      51     6 [TAAAAA]
##      [4]      96     101     6 [TAAATT]
##      [5]     101     106     6 [TAAAAT]
##      ...     ...     ...   ... ...
## [134422] 4639602 4639607     6 [CAATGT]
## [134423] 4639622 4639627     6 [CATGAT]
## [134424] 4639627 4639632     6 [TATTGA]
## [134425] 4639649 4639654     6 [TAAAAA]
## [134426] 4639668 4639673     6 [TATTTT]
```

The function matchPWM returns a data structure called “Views” which contains the start and end position of each match.

Next, we need to find the positions in the genome where the matches found are such that ttgacaPos is upstream from tataat, and there are 13-19 nucleotides in between them. We will do this by filtering the matches we have found.

We will use the function findOverlaps. The option minoverlap set to 0 will allow us to find positions which are not overlapping (since in fact we want them to be at least 13 nt apart), and maxgap = 19 will filter the positions which are more than 19 nt apart.

```
adjacentMotifs = findOverlaps(query = tataatPos, subject = ttgacaPos, maxgap = 19, minoverlap
 = 0)
adjacentMotifs
```

```
## Hits object with 971593 hits and 0 metadata columns:
##            queryHits subjectHits
##            <integer>   <integer>
##        [1]         1           1
##        [2]         1           2
##        [3]         1           3
##        [4]         1           4
##        [5]         1           5
##        ...       ...         ...
##   [971589]    134425      621334
##   [971590]    134425      621335
##   [971591]    134425      621336
##   [971592]    134426      621335
##   [971593]    134426      621336
##   -------
##   queryLength: 134426 / subjectLength: 621336
```

This function (findOverlaps) returns a structure in which the indices of the query and subject “Views” objects which are found to have “overlaps” (in fact with our parameters they are no more than 19 nt apart) are listed in the same row. In this step we only took care that the motifs aren’t more than 19 nt apart. The next step is to make sure that they are at least 13 nt apart and that ttgaca is upstream from tataat.

To do this we need to get the ranges (in variables tataatPos and ttgacaPos) from the indices (in variable adjacentMotives). When working with “Hits” objects, we can extract the query and subject hit indices using functions queryHits and subjectHits. These functions return arrays of numbers (no special object type)

```
tataatIndices = queryHits(adjacentMotifs)
ttgacaIndices = subjectHits(adjacentMotifs)
```

Now we want to get the ranges corresponding to these indices, so we will subset tataatPos and ttgacaPos with the indices.

```
tataatPosFiltered = tataatPos[tataatIndices]
ttgacaPosFiltered = ttgacaPos[ttgacaIndices]

#note that there are more "filtered" positions than original ones
#this is because, with the given minscore cutoff, so many motifs are found that one tataat mo
tif can be next to multiple ttgaca motives at this step in the filtering process
length(tataatPosFiltered)
```

```
## [1] 971593
```

```
length(tataatPos)
```

```
## [1] 134426
```

```
#same goes for the ttgaca motif
length(ttgacaPosFiltered)
```

```
## [1] 971593
```

```
length(ttgacaPos)
```

```
## [1] 621336
```

Now that we have acquired the indices of the ranges which are not more than 19 nt apart, we will use these ranges to further filter the range-pairs to make sure ttgaca is upstream from tataat and that they are at least 13 nt apart.

Both these conditions will be filtered in one step.

If we look at the following scheme we can see that the difference between tataat-start and ttgaca-end should be between 13 and 19 nucleotides.

start end__13-19__start end TTGACA nt TATAAT

Positions in the genome are counted from left to right. Subtracting the start of tataat from the end of ttgaca will allow us to filter the right motifs: the result should be greater than zero (which ensures that ttgaca is upstream from tataat) and greater than 13 (we have already ensured that the motifs are no more than 19 nt apart).

It follows that we need to filter the pairs of ranges for which the difference between the start of tataat and the end of ttgaca is equal to or larger than 13.

```
#first we will take the start positions of tataat and end positions of ttgaca from the previo
usly filtered ranges
tataatStart = start(tataatPosFiltered)
ttgacaEnd = end(ttgacaPosFiltered)

#this command will give us a logical vector (passFilter) in which the positions which pass th
e filter will be marked as TRUE
passFilter = tataatStart - ttgacaEnd >= 13
```

Using the logical vector passFilter, we will construct a new Ranges object containing our presumed sigma70 binding sites, which are defined as starting at the start of ttgaca and ending at the end of tataat.

```
#taking the start positions of ttgaca sites that passed all filters
sigmaStart = start(ttgacaPosFiltered[passFilter])
#taking the end positions of tataat sites that passed all filters
sigmaEnd = end(tataatPosFiltered[passFilter])


#constructing a new Ranges object
promotorPos = IRanges(start = sigmaStart,
                      end = sigmaEnd)
```

Now we have found some positions for which we assume that they are sigma70 binding sites, which means they are promoter sequences. Let us look at how many we have found:

```
length(promotorPos)
```

```
## [1] 145999
```

This is a very large number. The E. coli genome doesn't have nearly as many operons, so we might conclude that some of our filtering criteria were too lenient. We can play around with some of them, like the minscore cutoff, later. Let us now check if we at least caught most of the promoters really present in the genome with this large number of predicted promoter sites.

To do this we will use the annotation for the genome of this particular E. coli strain downloaded from NCBI.

https://www.ncbi.nlm.nih.gov/nuccore/NC_000913.3 (https://www.ncbi.nlm.nih.gov/nuccore/NC_000913.3)

We will load the annotation using the read.table() function. comment.char sets the character which will denote a line is a comment line. and these lines will not be loaded into the table. sep sets the denominator quote = "" disables R from adding quotes to columns read as character types, which can result in unwanted behavior stringsAsFactors = F disables R from treating strings as factors, which can be troublesome if we are switching between types of data structures or merging tables

```
annotation = read.table("sequence.gff3", comment.char = "#", sep = "\t", quote = "", stringsA
sFactors = F)
annotation
```

V1 <chr>	V2 <chr>	V3 <chr>	V4 <int>	V5 <int>	V6 <chr>	V7 <chr>	V8 <chr>
NC_000913.3	RefSeq	region	1	4641652	.	+	.
NC_000913.3	RefSeq	gene	190	255	.	+	.
NC_000913.3	RefSeq	CDS	190	255	.	+	0
NC_000913.3	RefSeq	gene	337	2799	.	+	.
NC_000913.3	RefSeq	CDS	337	2799	.	+	0
NC_000913.3	RefSeq	gene	2801	3733	.	+	.
NC_000913.3	RefSeq	CDS	2801	3733	.	+	0
NC_000913.3	RefSeq	gene	3734	5020	.	+	.
NC_000913.3	RefSeq	CDS	3734	5020	.	+	0

V1 <chr>	V2 <chr>	V3 <chr>	V4 <int>	V5 <int>	V6 <chr>	V7 <chr>	V8 <chr>
NC_000913.3	RefSeq	gene	5234	5530	.	+	.

1-10 of 9,726 rows | 1-8 of 9 columns Previous **1** 2 3 4 5 6 … 973 Next

If we look at what we got, we can see that come of the annotated features are genes, some are CDS (coding sequences), some are mobile genetic elements etc. If we look at the specification of teh GFF file format online (the loaded table is taht type of file) we will learn that the 4th and 5th column contaion the start and end of the feature.

We will filter the annotation so that we only get information for the coding sequences (CDS) because we are not interested in non-coding genes. We will also only take the start and end positions of the features.

When subsetting a table or dataframe in R, we will use the [] operator. In teh first positions we will put the filtering criterium for the first dimension (the rows), and in teh second position the filtering criterium for the second dimension (teh columns).

We will only take those rows for which V3 (type of feature) is “CDS”, and also only genes on the “+” strand (since we were also only searching for motifs on teh “+, strand”). The logical expression will be: annotation V7 == "+" & annotation V3 == “CDS” 7th column is equal to “+” AND 3rd column is equal to “CDS”

As for the columns, we will take the 4th and 5th columns which contain the start and end positions of the features. c(4,5)

```
#putting teh start and end positions of CDS into teh variable genes
genes = annotation[annotation$V7 == "+" & annotation$V3 == "CDS", c(4,5)]

#we need to transform this variable into a Ranges object in order to work with it further
#we will use the first column of genes as teh start positions and the second column as the en
d positions in teh arnges object
#we use [[]] instead of [] for subsetting because this way we will get a vector (using [] we
 would get a list, with one element, a vector)
genePos = IRanges(start = genes[[1]], end = genes[[2]])
```

Next, we will use very similar command as before to find how many of our identified promoter position are upstream of annotated CDSs. They should also be around 10 nt upstream from teh start of teh CDS.

Here again we can play with both the maximum distance allowed between the presumed promoter site and the CDS, as well as the minimum distance between them

```
maxDistance = 20
minDistance = 9

#we will look for overlaps between the promoters and the CDSs, requiring no minimup overlap a
nd allowing no more than maxDistacen distance between them
geneOverlaps = findOverlaps(query = genePos, subject = promotorPos, maxgap = maxDistance, min
overlap = 0)
```

As before, we will make sure that the promoter is upstream from the CDS and that the minimum distance between them is at least minDistance in 1 step

start end__10__start end promoter nt CDS

We will substract the start of the CDS with the end of the promoter ranges, and the number we get will have to be positive and greater than minDistance

```
cdsIndices = queryHits(geneOverlaps)
promIndices = subjectHits(geneOverlaps)

genePosFiltered = genePos[cdsIndices]
promotorPosFiltered = promotorPos[promIndices]

geneStart = start(genePosFiltered)
promotorEnd = end(promotorPosFiltered)


pass2 = geneStart - promotorEnd >= minDistance
```

Now we can look at what we have:

```
#all presumed sigma70 binding sites
length(promotorPos)
```

```
## [1] 145999
```

```
#binding sites upstream and close to annotated gene
sum(pass2)
```

```
## [1] 764
```

```
#annotated coding genes
dim(geneStart)
```

```
## NULL
```

Of the 145 999 presumed sigma70 binding sites, 764 have been found to be upstream and close to an annotated gene. There have been 2120 annotated genes on the “+” strand in total. However, genes in bacteria are organized into operons, and each operons only has 1 promoter.

In conclusion, this method might have captured most of the TRUE sigma70 binding sites, but it has a lot of false positives.

We should either make the filtering criteria (minscore, minDistance, maxDistance) more strict or combine this approach with ORF finding.