Vignette: Sasquatch R-Tool

Version:	0.1
Date:	June 16, 2017
Authors:	Ron Schwessinger, Maria Suciu, Simon J McGowan, Jelena M Telenius, Douglas R. Higgs & Jim R. Hughes
Groups:	Genome Biology and Computational Biology Research Group, WIMM, Oxford
Contact:	Ron Schwessinger, ron.schwessinger@msdtc.ox.ac.uk CBRG, managers@molbiol.ox.ac.uk Jim Hughes, jim.hughes@imm.ox.ac.uk
License:	GPLv3
Requires:	ggplot2, RColorBrewer
Recommends:	Biostrings, TFBSTools, pbapply

Contents

Introduction

Digestion enzymes like DNase I, cut the genomic DNA. This cutting is obstructed by proteins binding to the DNA like transcription factors and nucleosomes. Therefore, the cutting is more frequent in accessible (or open) regions of the chromatin. Since these regions are associated with regulatory activity, open-chromatin assays like DNase-seq or ATAC-seq are frequently used to map these regions of open-chromatin. Bringing this type of analysis to a higher resolution, footprinting analysis can be used to map the cut sites at a base pair resolution. At sufficient coverage, footprints can be identified as short sequences protected from DNase I cleavage within open-chromatin, pontially pointing to transcription factor or nucleosome occupied sequences.

Sasquatch brings this analysis to a global scale using a single DNase-seq experiment as input. In contrast, to previously proposed methods that average the DNase I cut sites over transcription factor motif matches (using PWMs) across the genome, Sasquatch uses an exhaustive and unbiased k-mer based approach. Sasquatchs takes all possible short sequences (k-mers) of length 5-7 bp and scans the genome-wide open-chromatin for occurrences of every k-mer. For every k-mer match, all DNase I cuts in a 250 bp window surrounding the k-mer are recorded (normalized for sequence bias) and attributed to the respective centric k-mer. Thus Sasquatch piles-up a repository of average cut profiles for every k-mer in every tissue pre-processed.

These data can than be used by this R-tool or our Web implementation, to retrieve and calculate the relative cut probability profiles associated with every k-mer of interest. Furthermore, we can quantify these profiles and infer information about the footprinting potential of k-mers, assessing their average bound or unbound state in the context of open-chromatin. Using a sliding k-mer window approach, this can be extended to longer sequences. Using comperative analysis, we can predicit the impact of sequence variations on the footprinting potnential of a sequence and thus rank large batches of variants according to their potential of influencing transcription factor binding. Furthermore, we can perform *in silico* mutations of genomic loci, predicting potentially occupied sequence from their susceptibility to single base pair substitutions. Since a single experimental input is used for every dataset, all analysis an be carried out in a highly tissue specfic fashion.

Here we introduce the R implementation of Sasquatch and guide through some basic workflows.

Initialize

First of all, we source the Sasquatch functions distributed in "functions_sasq_r_utility.R"

```
source(
"/home/ron/fusessh/Sasquatch_offline/Sasquatch/R_utility/functions_sasq_r_utility.R"
)
```

We set some parameters like the fragmentation type ["DNase" or "ATAC"] (relevant for merging and background) to select the desired data. Here, we analyse DNase-seq data.

```
frag.type <- "DNase"
```

We also set a *pnorm.tag* which indicates which background kmer propensities were used to normalise the recorded DNase cuts. For human this is "h_ery_1".

```
pnorm.tag <- "h_ery_1"
```

We also set the path to the database directory from where to read the preprocessed data, which depends on the fragmentation type and should contain subdirectories for every preprocessed or downloaded tissue.

```
data.dir <- file.path(
  "/home/ron/fusessh/database_assembly/idx_correct_assembly/human",
  frag.type
  )
```

```
print(data.dir)

## [1] "/home/ron/fusessh/database_assembly/idx_correct_assembly/human/DNase"

#example contents
list.files(path = data.dir, full.names = FALSE)[c(1,7,36,41)]

## [1] "archive"                    "ENCODE_Duke_A549_merged"
## [3] "ENCODE_Duke_GM19239_merged" "ENCODE_Duke_H9ES_merged"
```

Wrapper Functions like Web-Utility

The distribution contains wrapper functions, assembled to mirror the web-tool functionality. We will start by using them to mirror the web-tool workflow.

Workflow 1: Analyse Single k-mers

We first select our tissue of interest which is expected to be a unique subdirectory in your data repository.
Here we are interest in the primary erythroid data.

```
tissue <- "WIMM_primary_erythroid_Fibach_Fade8"
```

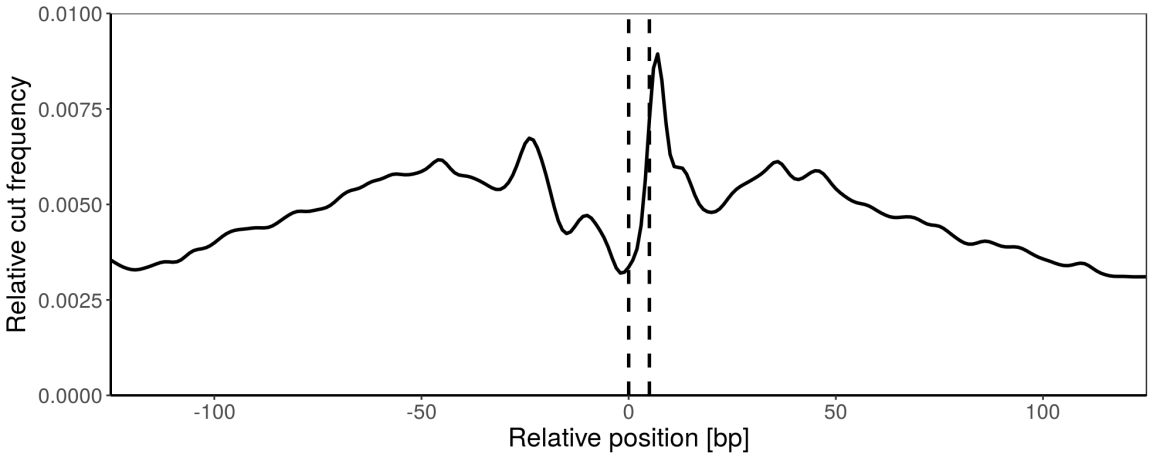
Plot profile of a single k-mer

GATA-factors are prominent transcription factors in erythroid cells. They bind to the onsensus sequence of "WGATAA". First, we plot the average profile of relative DNase I frequencies. To plot the average profile a selected k-mer, we use the "PlotSingleKmer" function. Sasquatch supports k-mers of length 5-7.

```
single.plot <- PlotSingleKmer(
  kmer="WGATAA",  #selected k-mer
  tissue=tissue,  #selecet tissue
  pnorm.tag=pnorm.tag,
  plot.shoulders=FALSE,  #decide if to estimate and
  # plot the footprint shoulder regions
  data.dir=data.dir,  #path to data repository
  frag.type=frag.type,  #fragmentation type ["DNase" or "ATAC"]
  smooth=TRUE  #flag if to smooth the cut profile
  )
```

Note that we can query an ambivalent FASTA character that is decoded into "AGATAA" and "TGATAA" and Sasquatch retrieves the average of the corresponding profiles. Sasquatch can handle all FASTA characters (A, C, G, T, U, R, Y, K, M, S, W, B, D, H, V, N).

```
single.plot
```



Calculate the Shoulder-to-Footprint Ratio of a single k-mer

To estimate the strength of a footprint we utilize the Shoulder-to-Footprint Ratio (SFR). To calculate the SFR for a single k-mer we use the "GetSFR" function.

```
sfr <- GetSFR(
  kmer = "WGATAA",
  tissue = tissue,
  data.dir = data.dir,
  pnorm.tag=pnorm.tag,
  vocab.file = TRUE,  #flag if to use a vocabulary file
  frag.type = frag.type
  )

print(sfr)

## [1] 1.638966
```

Note that we set the "vocab.flag = TRUE" to read the pre-calculated SFRs from the pre-calulated vocabulary file and speed the calculation time up dramatically.
The default path to the vocabulary file is set as

```
vocab.file=paste0(data.dir,"/",tissue,"/vocabulary_",tissue,".txt")

#example header
head(readLines(vocab.file), 5)

## [1] "AAAAAAA\t1.44029" "AAAAAAC\t1.18363" "AAAAAAG\t1.13017"
## [4] "AAAAAAT\t1.3364"  "AAAAACA\t1.07518"
```

If no vocabulary file has been precalculated yet, set the flag to "FALSE" to run the calculation from the raw cut profiles.

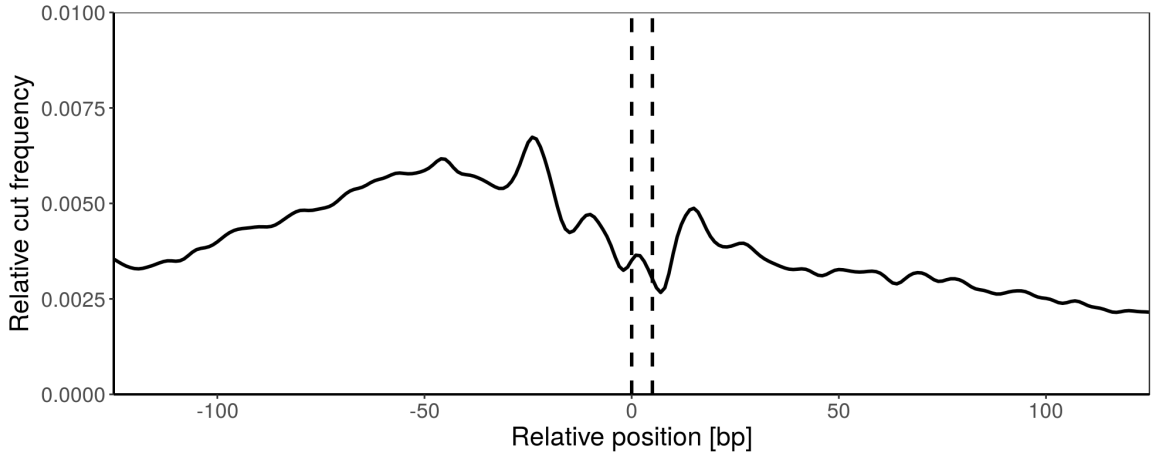
Plot strand-sprecific and background profiles of a single k-mer

DNase I exhibits a strinking strand bias. To investigate the impact of this imbalance, we can also plot the average cut profiles separately for both DNA strands.

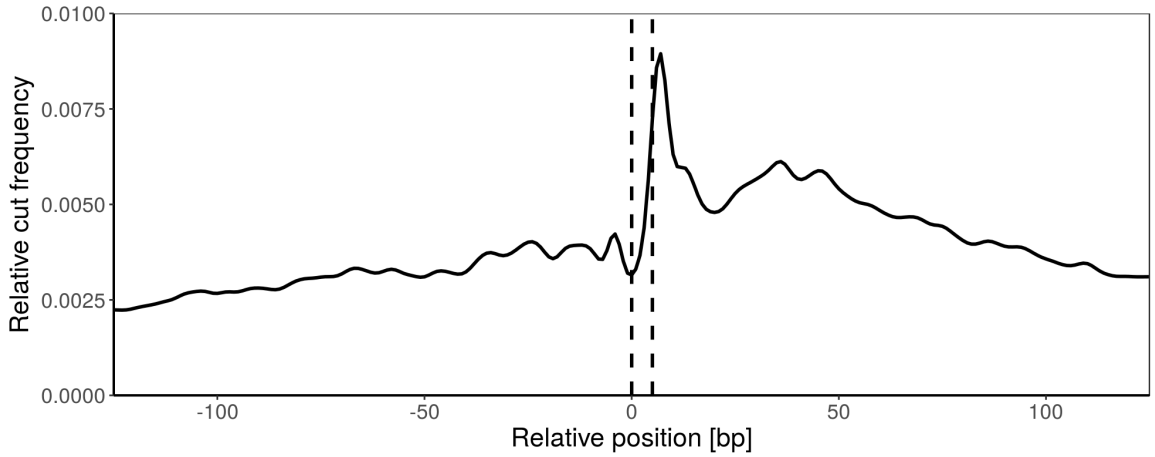
```
single.strands.plot <- PlotSingleStrands(
  kmer = "WGATAA",
  tissue = tissue,
```

```
  data.dir = data.dir,
  pnorm.tag=pnorm.tag,
  frag.type = frag.type,
  smooth=TRUE
  )
```

```
single.strands.plot$plot.plus
```




```
single.strands.plot$plot.minus
```



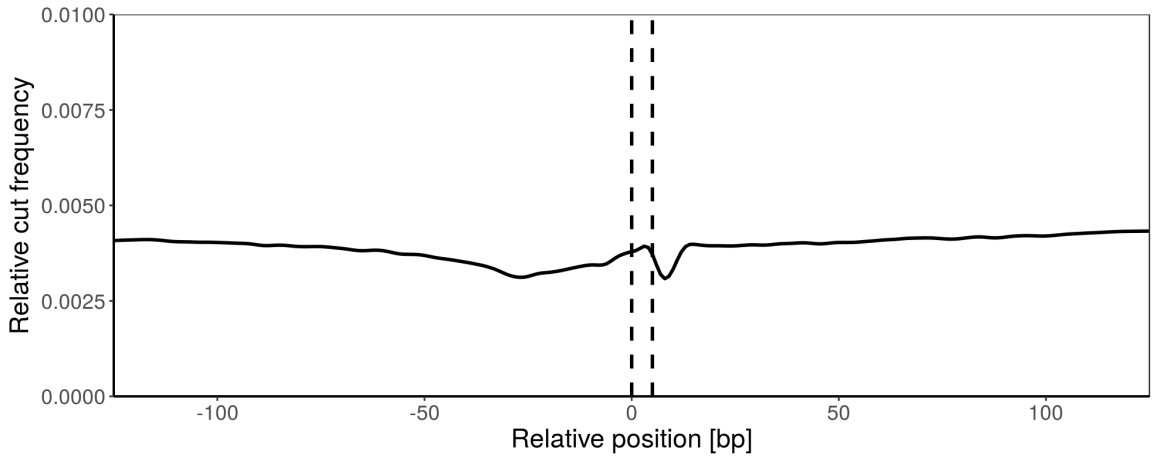
Note that the combined/merged average profiles are merged from the strand-specific. The merging procedure differs for DNase-seq and ATAC-seq derived data. ATAC-seq profiles are merged by averaging both strand profiles. In contrast, DNase-seq profiles are merged with respect to the assay's strand imbalance, see manual/publication.

To rule out that the DNase I sequence bias or other biases affect the estimated footprint, we can plot the relative cut profiles around the respecitve k-mer from the deproteinized, genome-wide background digestion assays. We can use the same function but set the "background.flag = TRUE" and define "data.dir" as the path to the background repository and select the id of the genome-wide background experiment (e.g. for human "default = "hg18_human_JH60"). Note that hg18 refers only to the reference genome used for mapping which does not affect the overall propensities.

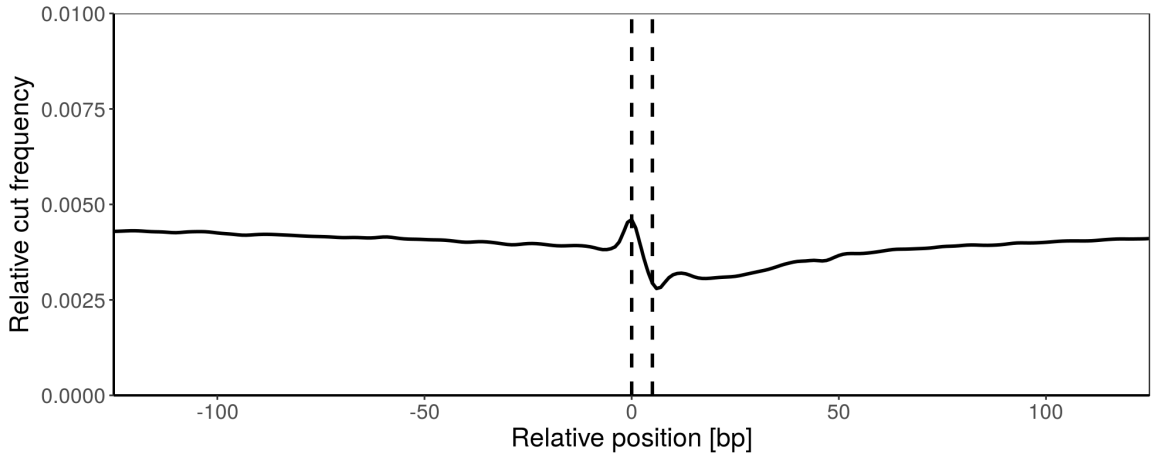
```
background.tissue <- "hg18_h_ery_1"
background.dir="/home/ron/fusessh/database_assembly/idx_correct_assembly/background/"
```

```
background.plot <- PlotSingleStrands(
  kmer = "WGATAA",
  tissue = background.tissue,
  data.dir = background.dir,
  pnorm.tag=pnorm.tag,
  frag.type = frag.type,
  smooth = TRUE,
  background.flag = TRUE
  )
```

```
background.plot$plot.plus
```




```
background.plot$plot.minus
```



We see slight deviations from the ideally expected, equal distribution but nothing that mirros the profiles dervid from the DHS data.

Workflow 2: Split and Analyse a Longer Sequence

Sasquatch works on a k-mer basis. Currently, it can handle k-mers of length 5 to 7 bp. To analyse longer sequences we can scan the sequence with a sliding window approach. We use the "QueryLongSequence" function. The input sequence is split into k-mers of length "kl" and the SFR ratio of the corresponding average profile is calculated. The results are reported in a data frame. If the plot flag is set "plot=TRUE" the function also retrieves a list of average profile plots, one for each splitted k-mer. Options for the plots can be set in the function call as well.

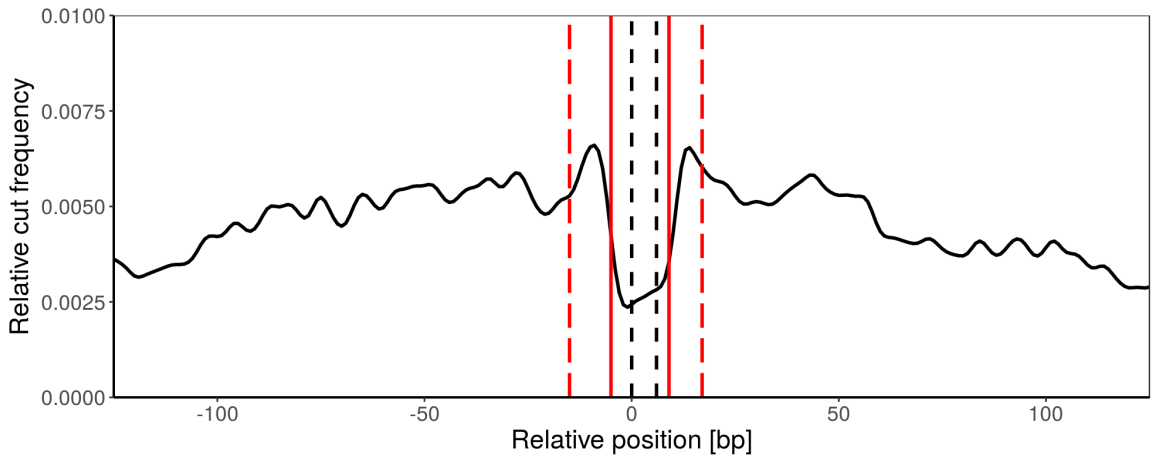
```
dissect.list <- QueryLongSequence(
  sequence="AGCACGTGTTC",
  kl=7,
  tissue=tissue,
  data.dir=data.dir,
  pnorm.tag=pnorm.tag,
  vocab.flag=TRUE,
  frag.type=frag.type,
  plots=TRUE,
  smooth=TRUE,
  plot.shoulders=TRUE
  )

dissect.list$df
```

```
##      kmer     sfr
## 1 AGCACGT 1.56084
## 2 GCACGTG 2.10825
## 3 CACGTGT 2.04224
## 4 ACGTGTT 1.59803
## 5 CGTGTTC 1.62318
```

```
dissect.list$plots[[2]] <- dissect.list$plots[[2]] +
  theme(text = element_text(size=16))
```

We see that position 2 scores highest and contains common E-box motif consensus sequence ("CACGTG"). To investiagte, we access the corresponding plot from the list.

```
dissect.list$plots[[2]]
```



Workflow 3: Compare Reference against Variance Sequences

We can use Sasquatch to estimate the impact of sequence variation e.g. from SNPs. This is predicted based on changes in the footprinting potential as proxy. To compare two sequences we use the "CompareSequences" function. Both input sequences are split up into k-mer windows of length "kl". The k-mers are then compared pair-wise by means of their SFRs. The damage is calculated as the difference between the reference SFR and the variant SFR ($SFR_ref - SFR_var = dmg$). A positive damage is thus associated with reducing the footprint characteristic, while a negative damage is associated with introducing or strengthening a footprint. The total damage is than calculated as the sum of all k-mer comparisons along the sequence pair [default: damage.mode = "exhaustive"] or

as the highest damage from a single k-mer pair [damage.mode = "local"]. The function retrieves a summary, a detailed data frame listing the pair-wise comparison and overlay profile plots. The default [plots="highest"] only reports the plot of the highest scoring k-mer pair. [plots="all"] retrieves a list containg one plot per pair.

For illustration, we estimate the impact of a SNP in an artificial GATA-site in a tissue specific manner.

```
compare.list <- CompareSequences(
  sequence1="ATAGATAATCGCT",  #reference sequence
  sequence2="ATAGATCATCGCT",  #variant sequence
  kl=6,
  damage.mode="exhaustive",  #mode to calcualte the overall damage
  tissue=tissue,
  data.dir=data.dir,
  pnorm.tag=pnorm.tag,
  vocab.flag=TRUE,
  frag.type=frag.type,
  plots="highest"  #only plot the highest damage scoring pair
  )
```

```
## Warning:  'panel.margin' is deprecated.  Please use 'panel.spacing' property instead
```

```
compare.list$summary
```

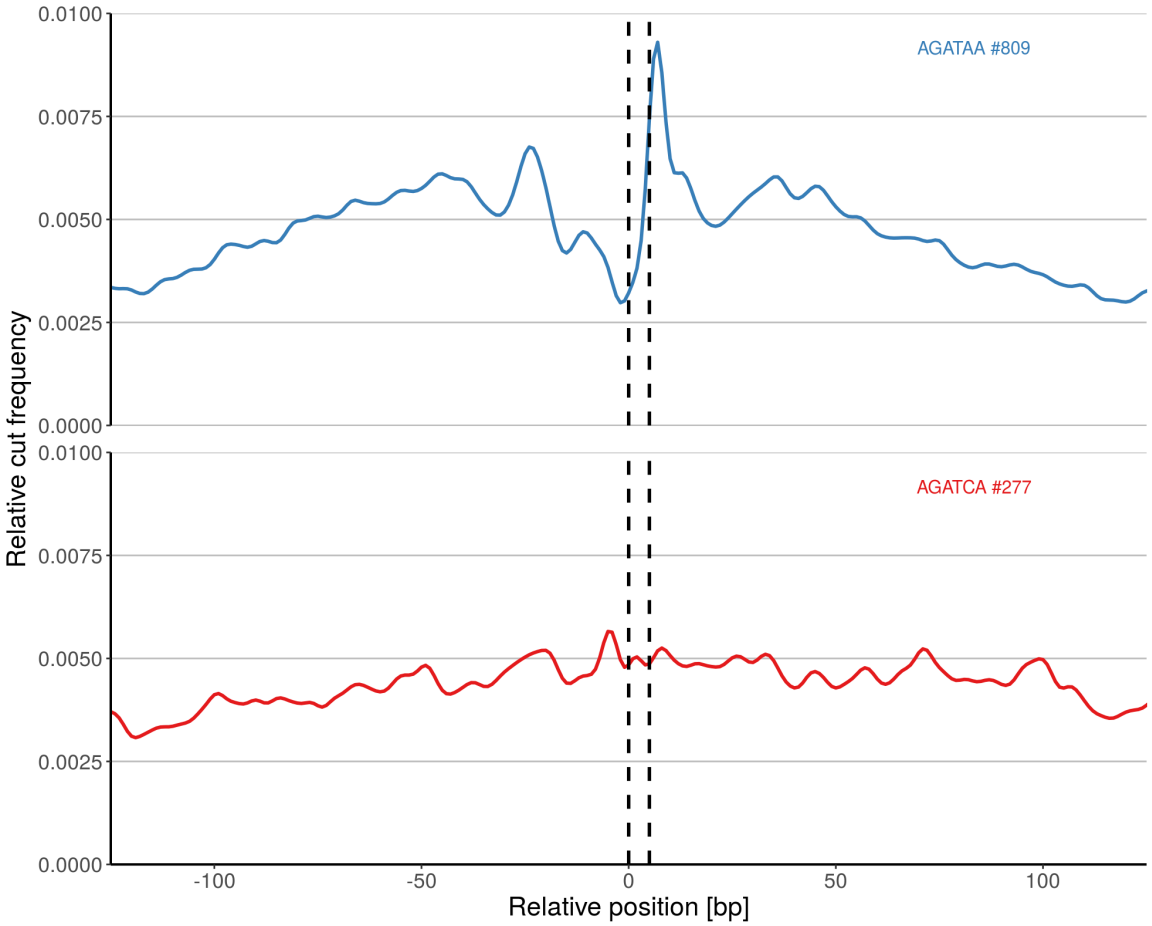
```
##    sequence.ref  sequence.var kmer.ref kmer.var SFR.ref SFR.var
## 1 ATAGATAATCGCT ATAGATCATCGCT   AGATAA   AGATCA   1.725   1.074
##   total.damage perc.change
## 1        0.903       0.898
```

```
compare.list$df
```

```
##   kmer.ref kmer.var sfr.ref sfr.var   damage
## 1   ATAGAT   ATAGAT 1.21169 1.21169  0.00000
## 2   TAGATA   TAGATC 1.45731 1.23765  0.21966
## 3   AGATAA   AGATCA 1.72539 1.07413  0.65126
## 4   GATAAT   GATCAT 1.29163 1.12140  0.17023
## 5   ATAATC   ATCATC 1.20423 1.36310 -0.15887
## 6   TAATCG   TCATCG 1.28574 1.37481 -0.08907
## 7   AATCGC   CATCGC 1.30640 1.19668  0.10972
## 8   ATCGCT   ATCGCT 1.17521 1.17521  0.00000
```

To visualize the difference in the average footprints, we access the overlay plot of the highest k-mer pair.

```
compare.list$plots
```

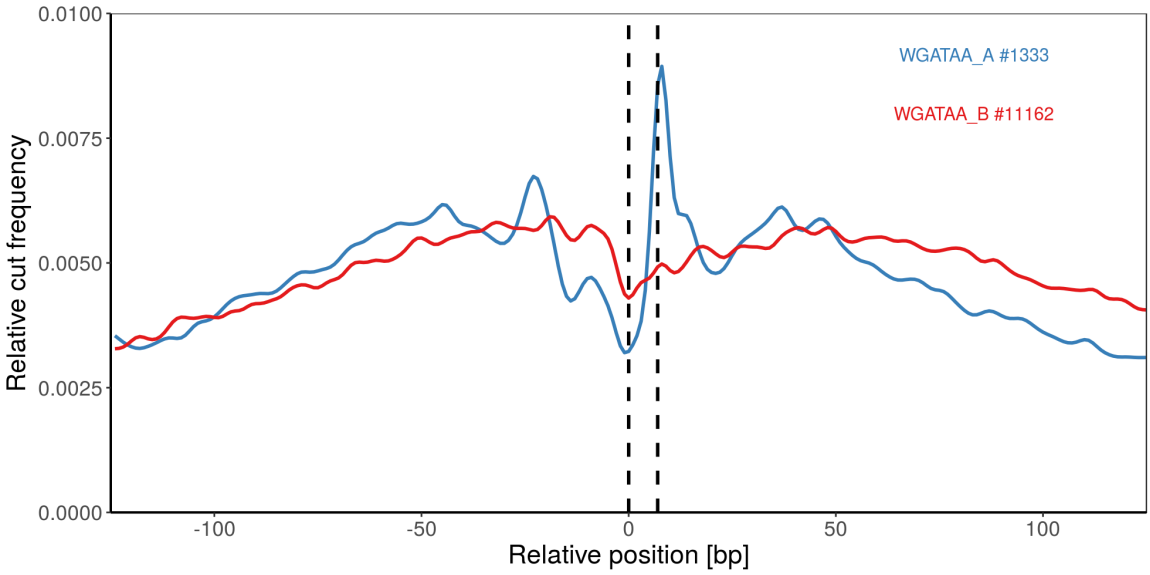


If we directly want to plot the comparison of two k-mers and bypass the rest, we can produce the same plot using the "PlotOverlapKmers" wrapper function. This function can also be used to quickly compare the footprinting potential of a single k-mer across different tissues. For example, we can compare the GATA-core motif in primary erythroid against and ENCODE MCF7 dataset.
Note: That average footprint (profiles) depend on the underlying data. Factors like differences in the DNase-seq protocol determine average footprint shapes. Therefore, average footprint profiles are not straight-forward to comapre across tissues. Such comparisons should be made only in a qualtitative fashion [like footrpint "present" or "not present"] and while keeping these factors in mind.

```
compare.plot <- PlotOverlapKmers(
  kmer1="WGATAA",
  kmer2="WGATAA",
  tissue1="WIMM_primary_erythroid_Fibach_Fade8",
  tissue2="ENCODE_UW_MCF7_merged",
  ymode="merged",  # ---> get a merged plot
  data.dir=data.dir,
  pnorm.tag=pnorm.tag,
  frag.type="DNase",
  smooth=TRUE
  )
```

Please note: To keep the plots tidy, we do not print the full tissue ID to the plots.

```
compare.plot
```



Workflow 4: Query Batches of Reference and Variant Sequence Pairs

To scan the damaging potential of multiple SNPs, we can can query pairs of reference and variant sequences as batch using the "RefVarBatch" function.

The input is a three column data frame "id ref var".
First we produce an examplary data frame.

```
batch.in <- data.frame(
        id=c("1", "2", "3"),
        ref=c("ATAGATAATCGCT", "ATAGATAATCGCT", "ATAGATAATCGCT"),
        var=c("ATAGATCATCGCT", "ATAGATTATCGCT", "ATAGATGATCGCT")
        )
```

Now we run the function. The result is an 9 column data frame listing the id and sequences, the highest scoring k-mers and the reference and variant SFR with the resulting total damage and percentange change. Note that this function specifically profits from a pre-calulated vocabulary file.

```
batch.results <- RefVarBatch(
  ref.var.df=batch.in,
  kl=7,
  damage.mode="exhaustive",
  tissue=tissue,
  data.dir=data.dir,
  pnorm.tag=pnorm.tag,
  vocab.flag=TRUE,
  frag.type=frag.type
  )

batch.results

##   id  sequence.ref  sequence.var kmer.ref kmer.var SFR.ref SFR.var
## 1  1 ATAGATAATCGCT ATAGATCATCGCT  ATAATCG  ATCATCG   1.894   1.406
## 2  2 ATAGATAATCGCT ATAGATTATCGCT  TAATCGC  TTATCGC   1.350   2.427
## 3  3 ATAGATAATCGCT ATAGATGATCGCT  ATAGATA  ATAGATG   1.728   1.351
##   total.damage perc.change
## 1        1.104       0.546
## 2       -0.153       0.755
```

```
## 3        0.580       0.518
```

To get a quick impression of potentially relevant transcription factors we use the JASPAR PWM data base to query the short sequences against transcription factor binding motifs. We first require the relevant packages.

```
library(Biostrings)
library(TFBSTools)
```

The relevant JASPAR motifs are stored as PWMs in the JASPAR2014 R package. We stored the human and mouse PWMS (all versions) into .RData objects to save preprocessing time. We load the relevant human object from the distribution.

```
#will load human.pwm
load(
"/home/ron/fusessh/database_assembly/jaspar/jaspar2014.human.9606.all.versions"
)
```

The "QueryJaspar" function scans an input sequence against all loaded PWMs and reports the highest scoring factors with their relative matching score. For querying a whole Workflow 4 resulting data frame we utilize the wrapper for the batch query "QueryJasparBatch". Per default the sequence (reference or variant) with the highest, single kmer SFR is queried.

```
batch.results.jaspar <- QueryJasparBatch(
  df=batch.results,  #data frame as result from Workflow 4
  damage.threshold=0.3,  #absolute damage threshold to query a SNP
  match.threshold=0.8,  #relative matching score to report a match
  pwm.data=human.pwm  #presaved JASPAR PWM object
  )

batch.results.jaspar
```

```
##   id  sequence.ref  sequence.var kmer.ref kmer.var SFR.ref SFR.var
## 1  1 ATAGATAATCGCT ATAGATCATCGCT  ATAATCG  ATCATCG   1.894   1.406
## 2  2 ATAGATAATCGCT ATAGATTATCGCT  TAATCGC  TTATCGC   1.350   2.427
## 3  3 ATAGATAATCGCT ATAGATGATCGCT  ATAGATA  ATAGATG   1.728   1.351
##   total.damage perc.change                          jaspar
## 1        1.104       0.546 GATA3=0.86;SRY=0.85;FOXL1=0.81;
## 2       -0.153       0.755                                .
## 3        0.580       0.518 GATA3=0.86;SRY=0.85;FOXL1=0.81;
```

Workflow 4 extended: Exhaustive *in silico* Mutation

A popular task for variant prediction methods is to perform an exhaustive *in silico* mutation for a genomic sequence of interest by querying each possible base substitution at every base position. We can use that to find potential motifs that can be disrupted by various mutations or mutations that would introduce a k-mer with a strong footprint potential.

First, we have to load genome data and get the genomic sequence of nterest as a character string.
Note: for mutating on a sliding k-mer window basis we have to extract "kl-1" bases surrounding the sequence of interest. (e.g. $\pm$ 6 bp for 7-mer basis)

```
#load genome
library(BSgenome)
library(BSgenome.Hsapiens.UCSC.hg18)
genome <- BSgenome.Hsapiens.UCSC.hg18
```

```
#set the sequence coordinates of the desired genomic location
chr <- "chr16"
start.pos <- 145852
end.pos <- start.pos + 30

# Get the sequence sequence
seq <- as.character(getSeq(genome, "chr16", start=start.pos-6, end=end.pos+6))

is(seq)

##  [1] "character"             "vector"
##  [3] "data.frameRowLabels"   "SuperClassMethod"
##  [5] "characterORconnection" "characterORNULL"
##  [7] "atomic"                "characterORMIAME"
##  [9] "EnumerationValue"      "vectorORfactor"

seq

## [1] "GTGCCCGCATGTGCTTATTTCTGCAAAAATAAACCATGGCAGG"
```

Now we can query the "in silico" mutation function "InSilicoMutation". The functions splits the sequence into windows of size $kl * 2 - 1$. For each window, the center base is mutated into all three possible bases. Then the reference as well as the variant windows are analysed on a sliding k-mer bases. Per default [damage.mode="exhaustive"], like in Workflow 3, the single SFRs ar summed up. The damage is calculated for each variant and reported according to the selected *report* mode. With the [default report="all"], all three possible mutations are reported for each position. The resulting dataframe is the input for ploting the InSilicoMutationplots. Alternatively, we can select to only report one possible mutation per position to retrieve a data frame that is straightforward to convert to bedGraph oder BigWig format for visualization in a genome browser. We can set report="max" to only report the highes possible damage per posiion or report="maxabs" to only report the highest absolute damage per position. Note that, like for the batch analysis, using the vocabluary file dramatiaćally speeds up the analysis here.

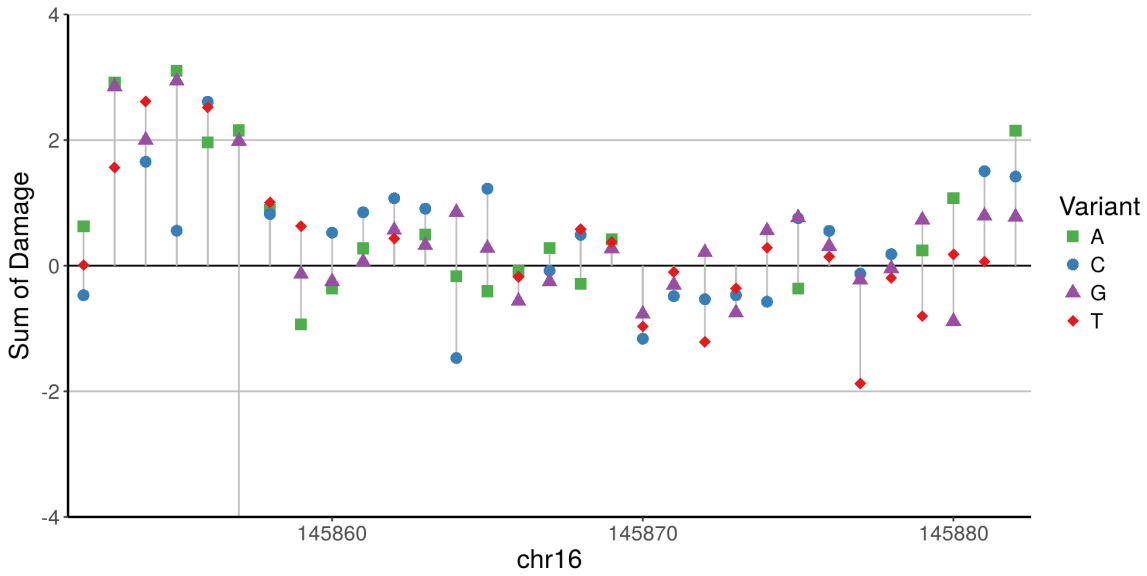
```
df.insilico <- InSilicoMutation(sequence=seq,
  kl=7,
  chr="chr16",
  position=start.pos,
  report="all",
  damage.mode="exhaustive",
  tissue=tissue,
  data.dir=data.dir,
  pnorm.tag=pnorm.tag,
  vocab.flag=TRUE,
  frag.type=frag.type,
  progress.bar = FALSE
)

## [1] "Processing 93 sequence windows:"

head(df.insilico)

##     chr    pos ref.base var.base       ref.seq       var.seq damage
## 1 chr16 145852        G        A GTGCCCGCATGTG GTGCCCACATGTG  0.624
## 2 chr16 145852        G        C GTGCCCGCATGTG GTGCCCCCATGTG -0.470
## 3 chr16 145852        G        T GTGCCCGCATGTG GTGCCCTCATGTG  0.011
## 4 chr16 145853        C        A TGCCCGCATGTGC TGCCCGAATGTGC  2.919
## 5 chr16 145853        C        G TGCCCGCATGTGC TGCCCGGATGTGC  2.850
## 6 chr16 145853        C        T TGCCCGCATGTGC TGCCCGTATGTGC  1.565
```

Note that the mutation is implemented as apply function and can easily be parallelized. To monitor progress we recommend the "pbapply" package and setting "progress.bar=TRUE" otehr wise set to FALSE.

To visualize the results in a InSilicoMutation ball plot, we parse the data frame into the "InSilicoMutationPlot" function.

```
InSilicoMutation.plot <- InSilicoMutationPlot(df.insilico, ylim=c(-4,4))
```

```
plot(InSilicoMutation.plot)
```



Speeding things up with preloading data

Many tasks can be speeded up by preloading either the preprocessed vocabulary file which contains the SFR for every possible kmer given a tissue of interest or the processed average profiles per kmer or both. By default Sasquatch greps the SFR values or profiles from the raw txt files. When querying multiple kmers, especially when performing *in silico* mutations (!!!), it is faster to load the raw files into memory first and parse them to the functions.

We can preload the profiles and the vocabulary file, given a tissue of interest from our database copy. Which of them to load and provide to the functions depends on the task of the function. Please refer to the reference manual for this options but in general, everything that onlz needs the SFRs and can be set with *vocab.file = TRUE* can run from preloading the vocabulary file only (which is very fast). Functions that work on the profiles for estimating shoulders, or quantify the SFR again or produce any kind of plots would need the preloaded profiles to be provided.

Preloading the vocabulary gives you a two cloumn data frame with all possible kmers and the respective SFR in the tissue of interest.

```
# Preloading the vocabulary file
vocabulary <- PreLoadVocab(data.dir, tissue)

head(vocabulary)

##        V1      V2
## 1 AAAAAAA 1.44029
## 2 AAAAAAC 1.18363
## 3 AAAAAAG 1.13017
```

```
## 4 AAAAAAT 1.33640
## 5 AAAAACA 1.07518
## 6 AAAAACC 1.16832
```

Preloading the profiles gives you a list of two data frames, one for the plus and one for the minus strand profiles. Each data frame has all kmers of a respective length in the first column, the respective kmer occurrences in the second column and a profile per kmer of length 300 bp + kmer length (305 - 307 bp). Please note that we store the surrounding 300 bp windows wen preprocessing but currently only use the surrounding 250 bp windwows for post-processing, quantification and visualisation.

```
# Preloading the profiles
profiles <- PreLoadKmerProfiles(kl=5, data.dir, tissue, pnorm.tag)

dim(profiles)

## NULL

dim(profiles$plus)

## [1] 1024  307

dim(profiles$minus)

## [1] 1024  307
```

Indivual Workflows from Basic Functions

The wrapper functions discussed above are constructed from more basic functions. For a more flexible usage/analysis we can construct our own workflow from the base level functions.