TSRchitect User's Guide

R. Taylor Raborn and Volker P. Brendel

Department of Biology, Indiana University

First edition 25 January 2017

Updated 2 February 2017

TSRchitect is an R package for analyzing diverse types of high-throughput transcription start site (TSS) profiling datasets. In recent years, large-scale TSS profiling data has characterized the landscape of transcription initiation at high resolution, identifying promoter architecture in a number of eukaryotic model systems, including human, mouse, fruit fly and worm. TSRchitect can handle TSS profiling experiments that contain either single-end or paired-end sequence reads.

Examples of TSS profiling data types that TSRchitect is capable of handling are:

- CAGE (Cap Analysis of Gene Expression) [Single-end]
- PEAT (Paired-end Analysis of Transcription) [Paired-end]
- RAMPAGE (RNA Annotation and Mapping of Promoters for Analysis of Gene Expression) [Paired-end]
- TSS-seq [Single-end]

TSRchitect provides the capability to efficiently identify putative promoters—which we call transcription start regions (TSRs)—from TSS profiling experiments. TSRchitect can accomodate multipel datasets, including biological replicates and multiple tissues/conditions, that were generated in a variety of model organisms and genome assemblies, requiring only aligned TSS profiling information (in BAM format) as the initial input. To aid the downstream analysis of identified promoters, TSRchitect calcualtes a variety of TSR properties that have been shown to be associated with promoter architecture, including TSR activity,

width and the Shape Index (SI). Finally, TSRchitect's output is compatible with popular differential expression software such as edgeR, assisting in downstream analysis to identify TSRs that are differentially active in one sample versus another. In addition to this vignette, the TSRchitect Reference Manual is available as part of the package's online documentation.

Getting started

In your current directory, create subdirectories as follows on the command line.

```
mkdir downloads
cd downloads
mkdir HsRAMPAGEbam
```

Now that this is complete, we proceed with the first of the three examples contained in this vignette.

Example 1: Identifying promoters from RAMPAGE data derived from two human cell lines.

RAMPAGE is a TSS profiling method that identifies promoters at large-scale using a cap-based library construction method that is adapted for paired-end sequencing. Developed recently by Batut and Gingeras (2013), it has become a popular method for promoter identification and is currently part of the data compredium in the latest edition of the ENCODE project.

In this example we will process RAMPAGE data derived from two immortalized human cell lines with TSRchitect. The experiments selected for this vignette are part of the ENCODE project and are publically available online at the ENCODE Experiment matrix. The two samples come from HT1080 cells, which is a well-characterized fibrosarcoma cell line, and NCI-H460 cells, which are derived from a large cell lung carcinoma in a male patient.

To begin, we must first download the RAMPAGE datasets (which were aligned to GRCh38 and are in BAM format) to our local system. To accomplish this we will utilize the "ENCODExplorer" package, which is part of the Bioconductor suite. More information on ENCODExplorer package can be found at the following link: https://www.bioconductor.org/packages/release/bioc/html/ENCODExplorer.html.

Now we can proceed (in the R console) with downloading the data:

```
#Downloading the files:
library(ENCODExplorer)
data(encode_df, package = "ENCODExplorer")
datasets <- fuzzySearch(searchTerm = c("ENCFF214GWH", "ENCFF265SGZ", "ENCFF242UWH", "ENCFF34
downloadEncode(datasets, df=encode_df, format="bam")
```

Once the above steps are complete, we will move the files into the subdirectory `downloads/` to keep a record of data provenance. To use TSRchitect in this example, we coveniently use symbolic links to give more intuitive names to the datasets (as well as ordering them in proper order; see below):

```
#The following are to be executed in your shell terminal:
mv *.bam downloads
cd downloads
cd HsRAMPAGEbam
ln -s ../downloads/ENCFF214GWH.bam H460-rep1.bam
ln -s ../downloads/ENCFF265SGZ.bam H460-rep2.bam
ln -s ../downloads/ENCFF242UWH.bam HT1080-rep1.bam
ln -s ../downloads/ENCFF348EKW.bam HT1080-rep2.bam
cd ..
```

We'll also need to download a human gene annotation, which will be important a little later.

Please download and uncompress the annotation file and move it into the downloads/ folder.

The file is found here: ftp://ftp.sanger.ac.uk/pub/gencode/Gencode_human/release_19/gencode.v19.annotation.gff3.gz

Now that we have our input and annotation files prepared, we can load TSRchitect into our R workspace, along with another library required for parallelization:

```
#loading TSRchitect
library(TSRchitect)
```

To implement parallelization, we'll need to set up a cluster of nodes first (here we use 4 nodes as we have 4 .bam files). Using parallelization will substantially speed up processing.

```
#loading the doParallel package, which is requried for parallel processing
library(doParallel)

mycl <- makeCluster(4,type="FORK",outfile="")
registerDoParallel(mycl)
```

Next we'll create the dedicated S4 object—called the tssObject—on which TSRchitect's functions will be performed.

```
initializeExp(expTitle ="Human RAMPAGE", experimentName="Hs_RAMPAGE", /
isPairedEnd=TRUE)
ls()
```

This will create an object called `Hs_RAMPAGE` in your workspace. Before we continue, let's take a look look at the object in our workspace using `ls()` to make sure it appears.

Next we need to provide the sample names and specify which samples are biological replicates. In this case we are working with 4 total datasets and 2 samples in duplicate. Please note that, because the alignments on our `bamData` slot are organized in ascending alphabetical order (as are the file names on the `fileNames` slot), we must provide our identifiers in `sample.names` and `replicateIDs` to directly correspond to this. To check this on the tssObject S4 object you have created, simply check the list of .bam files as follows using one of TSRchitect's accessor methods.

```
#obtaining a list of bam files loaded to the tssObject S4 object
getFileNames(experimentName=Hs_RAMPAGE)
```

```
setSampleID(experimentName=Hs_RAMPAGE, sample.names=c("H460-rep1", /
"H460-rep2", "HT1080-rep1","HT1080-rep2"), replicate.IDs=c(1,1,2,2))
```

Now that we have completed the above commands, we can proceed with importing the .bam files using the function `importBam`. This will attach GenomicAlignments objects (representing the four bam files in this example) to your tssObject.

```
importBam(experimentName=Hs_RAMPAGE, expDir="HsRAMPAGEbam/"
```

You will receive a message on your console when all four files have been imported:

Now that the alignment files have been imported and attached to our tssObject S4 object, we continue by computing the TSSs from the .bam files.

```
bamToTSS(experimentName=Hs_RAMPAGE)
```

Next we will calculate the abundance of each tag in our TSS datasets.

```
processTSS(experimentName=Hs_RAMPAGE, parallel=TRUE, tssSet="all", writeTable=TRUE)
```

Since we specified 'writeTable=TRUE', files (entitled "TSSset-1.txt" to "TSSset-4.txt") containing TSS abundance will be written into your working directory.

Now that we have calculated the abundance for each TSS in the previous step we can calculate TSRs (promoters) on each of the 4 separate datasets. We select a `tagCount` threshold of 25 tags in order for a TSS to be considered. The option `clustDist` is critical for the identification of TSRs and refers to the minimum distance between distinct TSRs (in other words, adjacent TSSs seperated by more than `clustDist` nucleotides will be in different TSRs). As with the previous step, we select 'writeTable=TRUE', and therefore we will find the output files ("TSRset-1" to "TSRset-4") written to the working directory.

```
determineTSR(experimentName = Hs_RAMPAGE, parallel = TRUE, tsrSetType = "replicates", /
tssSet="all", tagCountThreshold=25, clustDist=20, writeTable=TRUE)
```

To calculate TSRs from each sample (as opposed to each replicate) we need to combine our replicate data. This will be done using the identifiers we specified on our tssObject S4 object using the function initializeExp().

```
mergeSampleData(experimentName=Hs_RAMPAGE)
```

Having combined the TSS abundance of replicate data into samples, we next proceed with identifying TSRs for the two samples individually. We specify this with 'tsrSetType="merged"'.

```
#Generating the TSRs for the merged datasets:
determineTSR(experimentName=Hs_RAMPAGE, parallel=TRUE, tsrSetType="merged", /
tssSet="all", tagCountThreshold=40, clustDist=20, writeTable=TRUE)

#Ending the parallel session we set up earlier in the vignette:
stopCluster(mycl)
```

Now calculating the number of tags from each experiment within the combined set of TSRs:

```
addTagCountsToTSR(experimentName=Hs_RAMPAGE, tsrSetType="merged", /
tsrSet=3, tagCountThreshold=40, writeTable=TRUE)
```

Associating identified TSRs with gene annotations

Now that identifying TSRs are complete, an obvious and biologically useful step is to determine which TSRs are adjacent to annotated genes, and to retrieve the appropriate gene IDs. Before doing this, it is imperative to select an annotation file that was generated for the assembly to which the reads were aligned. For this example we will retrieve the appropriate annotation files from the Bioconductor package `AnnotationHub`.

Please install `AnnotationHub` if you haven't already done so.

```
source("https://bioconductor.org/biocLite.R")
biocLite("AnnotationHub")
```

In our case we need to download the Gencode annotation. We do this in the following manner:

```
library(AnnotationHub)
hub <- AnnotationHub()
query(hub, c("gencode", "gff", "human"))
```

This reveals nine gff3 annotations from Gencode that we can choose from. We will select the full annotation, which has the identifier "AH49555".

```
AnnotationHub with 9 records
# snapshotDate(): 2017-01-05
# $dataprovider: Gencode
# $species: Homo sapiens
# $rdataclass: GRanges
# additional mcols(): taxonomyid, genome, description,
#   coordinate_1_based, maintainer, rdatadateadded, preparerclass, tags,
#   sourceurl, sourcetype
# retrieve records with, e.g., 'object[["AH49554"]]'
```

```
          title
AH49554 | gencode.v23.2wayconspseudos.gff3.gz
AH49555 | gencode.v23.annotation.gff3.gz
AH49556 | gencode.v23.basic.annotation.gff3.gz
AH49557 | gencode.v23.chr_patch_hapl_scaff.annotation.gff3.gz
AH49558 | gencode.v23.chr_patch_hapl_scaff.basic.annotation.gff3.gz
AH49559 | gencode.v23.long_noncoding_RNAs.gff3.gz
AH49560 | gencode.v23.polyAs.gff3.gz
AH49561 | gencode.v23.primary_assembly.annotation.gff3.gz
AH49562 | gencode.v23.tRNAs.gff3.gz
```

Using TSRchitect, We can use the function `importAnnotationHub` to import our desired annotated record and attach it our `tssObject`. We accomplish this as follows:

```
importAnnotationHub(experimentName = Hs_RAMPAGE, provider = "gencode", annotType = "gff3", s
ID = "AH49555")
```

Next, we associate the gene annotation to the TSRs within our two merged samples. We selected the feature 'transcript' from the Gencode annotation.

```
addAnnotationToTSR(experimentName=Hs_RAMPAGE, tsrSetType="merged", /
tsrSet=1, upstreamDist=1000, downstreamDist=200, feature="transcript", /
featureColumnID="ID", writeTable=TRUE)

addAnnotationToTSR(experimentName=Hs_RAMPAGE, tsrSetType="merged", /
tsrSet=2, upstreamDist=1000, downstreamDist=200, feature="transcript", /
featureColumnID="ID", writeTable=TRUE)
```

Finally, we will repeat the two commands above, instead associating the gene annotation to the "combined" set of TSRs, which is found in the 3rd position on the `tsrDataMerged` slot.

```
addAnnotationToTSR(experimentName = Hs_RAMPAGE, tsrSetType="merged", tsrSet=3, /
upstreamDist=1000, downstreamDist=200, feature="transcript", featureColumnID="ID", /
writeTable=TRUE)
```

Let's briefly look at the sets of TSRchitect-identified TSRs.

Using the accessor methods we applied in earlier examples, let's take a quick glance at our set of identified TSRs.

```
getTSRdata(Hs_RAMPAGE, slotType="merged", slot=1) -> HT1080.tsrs
dim(HT1080.tsrs)

getTSRdata(Hs_RAMPAGE, slotType="merged", slot=2) -> H460.tsrs
dim(H460)

getTSRdata(Hs_RAMPAGE, slotType="merged", slot=3) -> combined.tsrs
dim(combined.tsrs)
```

Let's look at some of the tsrs we identified on our 'combined' set.

```
head(combined.tsrs)
```

We see that there are 22750 TSRs identified in the combined set, and 15904 and 18040 TSRs in the H460 and HT1080 samples, respectively. We also notice that there are 5 additional columns in the combined set. This is due to us previouly having added tag counts to the combined set of TSRs using 'addTagCountsToTSR', something we did not do in this vignette for the two individual samples.

You now have a complete set of TSS and TSR data attached to your tssObject S4 object, in addition the tables that were already written to your working directory.

This concludes Example 1. Should we wish to save our tssObject and return to our work later, we simply type the following, which will write an R binary to your working directory.

```
save(Hs_RAMPAGE, file="Hs_RAMPAGE.RData")
```

Important note: before you continue with another example, please move the output files generated in your working directory to a separate, dedicated folder. Otherwise some or all of the files you generate in subsequent examples will be overwritten.

Example 2: Identifying promoters in the model plant A. thaliana using a PEAT dataset.

For our second example will process TSS profiling data from Arabidopsis root tissue. These data come from the Megraw Lab at Oregon State University as reported in Morton et al., 2014. Link to paper:

As with the previous example, we first must download the raw data. In this case we have only a single alignment file to retrieve, which is found here: https://drive.google.com/open?id=0B5WxM8Ere19uZmlHM1F3ZS1iWWc.

The annotation file is available from the TAIR10 database: ftp://ftp.arabidopsis.org/home/tair/Genes/TAIR10_genome_release/TAIR10_gff3/TAIR10_GFF3_genes.gff. Please move it into the downloads/ folder you have created.

Once download of the file `peat.sorted.bam` is complete, please move it to subdirectory "PEATbam/". Since there is only a single experiment, setting the sample IDs is simple:

```
initializeExp(expTitle="Arabidopsis PEAT dataset", experimentName="At_PEAT",
isPairedEnd=TRUE)

setSampleID(experimentName=At_PEAT, sample.names=c("experiment1"),
replicate.IDs=c(1))
```

Now we import the alignment file to our tssObject and convert the data to TSS coordinates:

```
importBam(experimentName=At_PEAT, expDir="PEATbam/")

bamToTSS(At_PEAT)
```

As in the previous example, now we can calculate the tag abundance at each location using `processTSS` and the identify TSRs within the sample using `determineTSR`. Note that we do not need to use `mergeSampleData` because there is only a single sample.

```
processTSS(experimentName=At_PEAT, parallel=FALSE, tssSet="all", writeTable=TRUE)
determineTSR(experimentName=At_PEAT, parallel=FALSE, tsrSetType="replicates", tssSet="all",
tagCountThreshold=25, clustDist=20, writeTable=TRUE)
```

Associating identified TSRs with gene annotations

We continue by associating our newly-identified TSRs with genes from the TAIR10 annotation. Note that we use different parameters for upstreamDist and downstreamDist than we did in Example 1. This is due to the high degree of compactness in the A. thaliana genome.

```
importAnnotationExternal(experimentName=At_PEAT, fileType="gff3", /
annotFile="downloads/TAIR10_GFF3_genes.gff") /

addAnnotationToTSR(experimentName=At_PEAT, tsrSetType="replicates", tsrSet=1, upstreamDist=5
downstreamDist=200, feature="gene", featureColumnID="ID", writeTable=TRUE)
```

Now we have a complete set of TSRs on our tssObj object. Let's take a look at them using one of our accessor methods.

```
At.tsrs <- getTSRdata(AtPEAT, setType="replicates", set=1)
dim(At.tsrs)
head(At.tsrs)
```

We can optionally save the tssObject as we have previously.

```
save(At_PEAT, file="At_PEAT_vignette.RData")
```

Example 3: Analysis of CAGE datasets from the ENCODE project

As we stated in the introduction of this vignette, TSRchitect is capable of handling diverse forms of TSS profiling data. In the first two examples, we analyze two distinct paired-end datasets: RAMPAGE and PEAT, respectively. In this example we will process data from CAGE, which is the most widely-used TSS profiling method to date. We will analyze CAGE data generated in two

well-characterized immortalized cell lines, MCF-7 and A549. MCF-7 cells are derived from a breast cancer tumor, and A549 originates from an adenocarinoma isolated from lung tissue. Both datasets are part of the ENCODE project, and therefore we can make use of the ENCODExplorer package that we originally introduced in Example 1.

```
#Downloading the files:
library(ENCODExplorer)
data(encode_df, package = "ENCODExplorer")
cage_data <- fuzzySearch(searchTerm = c("ENCFF552BXH","ENCFF288VTZ","ENCFF265RSX", /
"ENCFF944PCJ"), database = encode_df, /
filterVector = c("file_accession"), multipleTerm = TRUE)
downloadEncode(cage_data, df=encode_df, format="bam")
```

Now that the files have been downloaded, we will create symbolic links with the appropriate sample names. Please run the following commands from a linux command line:

```
mkdir downloads # ignore if the directory already exists
mv *.bam downloads
mkdir HsCAGEbam
cd HsCAGEbam
ln -s ../downloads/ENCFF265RSX.bam A549-rep1.bam
ln -s ../downloads/ENCFF944PCJ.bam A549-rep2.bam
ln -s ../downloads/ENCFF552BXH.bam MCF7-rep1.bam
ln -s ../downloads/ENCFF288VTZ.bam MCF7-rep2.bam
cd ..
```

As in Example 1, we also need to download a human gene annotation. Please download and uncompress the annoation file [Note: you may ignore this if you have already downloaded the file from Example 1].

The annotation file is found at the following location: ftp://ftp.sanger.ac.uk/pub/gencode/Gencode_human/release_19/gencode.v19.annotation.gff3.gz. Please move it into the downloads/ directory if it does not already exist there.

Now we can set up the tssObject S4 object. Note that we must specify `isPairedEnd=FALSE` because this is single-end CAGE data.

```
# initializing the tssObject
initializeExp(expTitle ="Human CAGE experiment", experimentName="CAGEhuman", expDir="HsCAGEb
```

Now we set the sample IDs. As before, it is vital to provide `sample.names` and `replicate.IDs` in the order of the files on the `fileNames` slot to they exactly correspond to the .bam files that we imported.

```
# setting the sample IDs and names for the datasets within the CAGE library
setSampleID(experimentName=CAGEhuman, sample.names=c("A549-rep1","A549-rep2",/
"MCF7-rep1","MCF7-rep2"), replicate.IDs=c(1,1,2,2))
```

As in the prior two examples, we then import the .bam files and obtain TSSs from these data.

```
#Loading the .bam files from the CAGE experiment:
importBam(experimentName=CAGEhuman)

#Converting the alignment data into TSS information and attaching it to the tssObject:
bamToTSS(experimentName=CAGEhuman)
```

As we did in our first example, we need to set up a parallel instance as follows:

```
#Setting up a parallel instance:
library(doParallel)
mycl <- makeCluster(4,type="FORK",outfile="")
registerDoParallel(mycl)
```

Next we must calculate the CAGE tag abundance at each TSS position, followed by identification of TSRs within our 4 replicate datasets.

```
#Constructing the tag count per TSS data matrix:
processTSS(experimentName=CAGEhuman, parallel=TRUE, tssSet="all", writeTable=TRUE)

#Finding TSRs for the replicate datasets:
determineTSR(experimentName=CAGEhuman, parallel=TRUE, tsrSetType="replicates", /
tssSet="all", tagCountThreshold=25, clustDist=20, writeTable=TRUE)
```

Now we merge data from replicates into their two corresponding samples.

```
#Merging TSS data from the replicates:
mergeSampleData(experimentName=CAGEhuman)
```

Once this is complete, we can complete TSR identification on the merged samples.

```
#Finding TSRs for the merged samples and adding tag counts:
determineTSR(experimentName=CAGEhuman, parallel=TRUE, tsrSetType="merged", /
tssSet="all", tagCountThreshold=40, clustDist=20, writeTable=TRUE)
addTagCountsToTSR(experimentName=CAGEhuman, tsrSetType="merged", tsrSet=3, /
tagCountThreshold=40, writeTable=TRUE)

#Ending the parallel session we set up earlier in the vignette:
stopCluster(mycl)
```

Now we need to import the annotation file and attach it to our tssObj S4 object. To do this, we will to use the same record (Gencode v. 23) that we referred to in Example 1.

```
importAnnotationHub(experimentName = CAGEhuman, provider = "gencode", annotType = "gff3",
species = "human", ID = "AH49555")
```

Next we associate the gene annotation to the TSRs within our two merged samples i) MCF7 cells and ii) A549 cells. As we did in Example 1, we select the feature 'transcript' from the Gencode annotation.

```
addAnnotationToTSR(experimentName = CAGEhuman, tsrSetType="merged", tsrSet=1,/
upstreamDist=1000, downstreamDist=200, feature="transcript", featureColumnID="ID",/
writeTable=TRUE) #A549 cells
addAnnotationToTSR(experimentName = CAGEhuman, tsrSetType="merged", tsrSet=2, /
upstreamDist=1000, downstreamDist=200, feature="transcript", featureColumnID="ID",/
writeTable=TRUE) #MCF7 cells
```

Associating the selected annotation features with the TSRs on the 'combined' slot:

```
addAnnotationToTSR(experimentName=CAGEhuman, tsrSetType="merged", tsrSet=3,/
upstreamDist=1000, downstreamDist=200, feature="transcript", featureColumnID="ID",/
writeTable=TRUE)
```

Using the accessor methods we applied in earlier examples, let's take a quick glance at our set of identified TSRs.

```
getTSRdata(CAGEhuman, setType="merged", slot=1) -> MCF7.tsrs

dim(MCF7.tsrs)

getTSRdata(CAGEhuman, setType="merged", slot=2) -> A549.tsrs

dim(A549.tsrs)

getTSRdata(CAGEhuman, setType="merged", slot=3) -> CAGEhuman.tsrs

dim(CAGEhuman.tsrs)
```

Let's look at some of the tsrs we identified on our 'combined' set.

```
head(CAGEhuman.tsrs)
```

We can optionally save the tssObject for future use:

```
save(CAGEhuman, file="CAGEhuman-vignette.RData")
```