ExplorATE - Explore Active Transposable Elements -

Martin M. Femenias

ExplorATE (Explore Active Transposable Elements) is an R package for the exploration and identification of active transposons in RNA-seq data. The package offers functions to manipulate the RepeatMasker output files, and allows to discriminate TEs-coding transcripts from those repeats that are co-transcribed with genes coding non-transposon proteins. Through a simple pipeline you can solve overlaps of the repetitions that RepeatMasker cannot solve based on either highest score (HS), longer length (LE) or lower Kimura's distances (LD). The transposons are finally annotated in the reference file. Additionally, the user can apply additional selection criteria based on a like-Wicker rule, taking into account the percentage of identity, the length of the transposon family, and the coverage of a transposon family in the transcript. A decoy file and a transcriptome salmon-formated are created to be used for indexing and quantification with Salmon. Finally, a function is incorporated to import the quantification estimates into the R environment for their subsequent differential expression analysis.

INDEX

1. INSTALLING ExplorATE

ExplorATE requires some previously installed packages. Make sure you have a recent R version (> 4.1.0) and select those packages that are not available in your environment:

```
#install complementary packages
install.packages(c("stringr","seqinr","foreach","doParallel"))
install.packages(c("BiocManager","devtools"))
#BiocManager is required to install the packages below and devtools
#is required if you want to install from GitHub

BiocManager::install(c("readr","GenomicRanges", "IRanges","csaw", "edgeR",
                       "SummarizedExperiment","DESeq2", "tximport"))
```

You can install ExplorATE locally by downloading the file `ExplorATE_0.1.tar.gz` or install via GitHub using `devtools::install_github()`

```
#install ExplorATE locally
install.packages("./ExplorATE_0.1.tar.gz", repos=NULL, type = "source")

#install ExplorATE from GitHub
devtools::install_github("FemeniasM/ExplorATEproject")
```

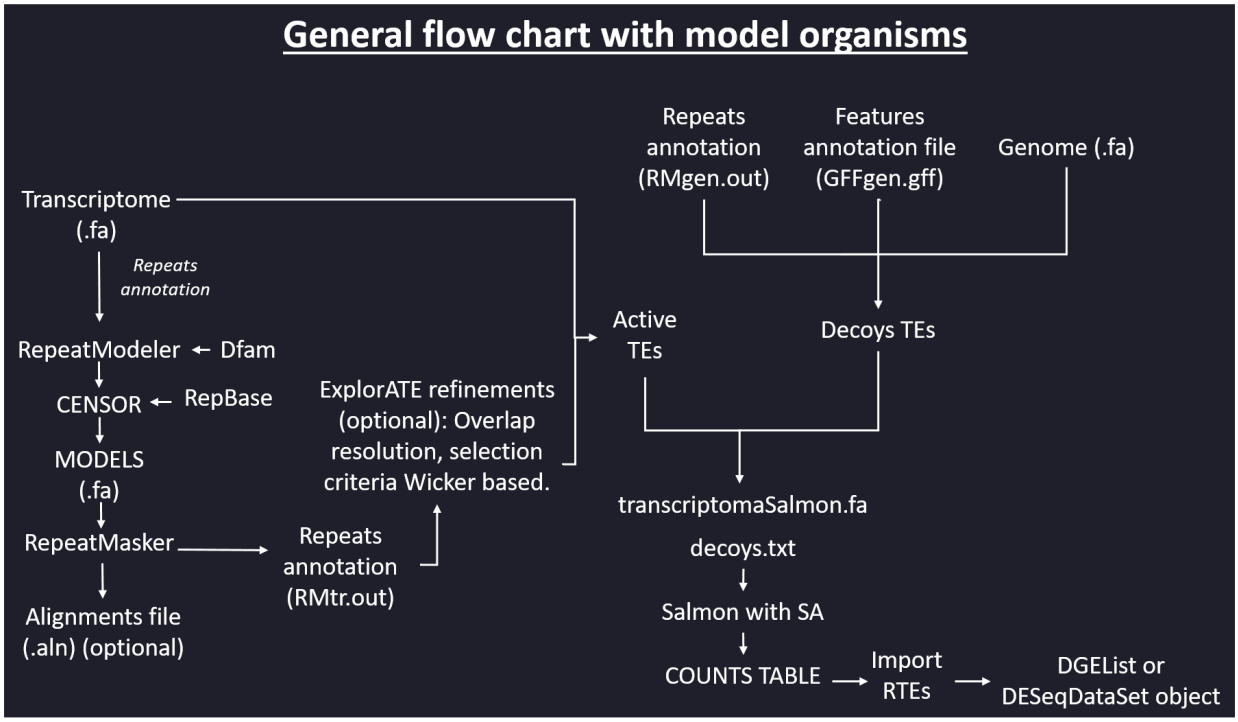
2. QUICK START

ExplorATE allows the exploration of transposable elements whether a reference genome exists or not. Below is a quick example for each case, however we encourage the user to explore the extended pipeline step by step.

2.1 Make TEs references in model organisms

When the reference genome is available the user must supply a *de novo* assembled transcriptome and a RepeatMasker output file originated from the transcriptome (`RMtr.out` and `transcriptome.fa` in the example below). In addition, you must have the genome in `.fasta` format, the GFF annotations file for the genome version, and a RepeatMasker file for the reference genome that can be obtained from the program's website.

ExplorATE takes the transcriptome and the RepeatMasker output file derived from the transcriptome to make a reference of active TEs in the samples. It also performs the overlaps resolution of repeats within each transcript. The user can perform additional settings on the RepeatMasker file as shown in sections 5 and 6. The figure below shows a summary of the workflow when the genome is available.



You can get the decoys files to run salmon from the bash script provided by ExplorATE, or run the `mk.reference_mo()` function from the ExplorATE package. Both examples are shown below. To run the shell script, first download the script and place it in a directory on your system:

```
wget https://github.com/FemeniasM/ExplorATEtools/mk.references_mo.sh
```

Then run the script assigning the appropriate arguments:

```
bash mk.references_mo.sh -p <threads> -b <bedtools binary path> -a dm_genannot.gff -g dm_genome.fa
-t dm_transcriptome.fa -o . -r dm_RMtr.out -s dm_RMgen.out -j 'HS' -c 'namRep'
```

By default `awk` and `bedtools` are assumed to be available in your `$PATH` environment variable. The `-j` argument defines the overlap resolution criteria that can be based on high scores (`'HS'`), longer element (`'LE'`), or low divergence (`'LD'`). The `-c` argument defines how the repeats found will be classified. if

'namRep' is defined it will take the names of each matching repetition (tenth column in the RepeatMasker output file). If 'classRep' is defined, the class of the repetition will be taken (the eleventh column in the RepeatMasker output file).

Linux users can also use the mk.reference_mo() function to run the same script from R:

```
RM.reference <- mk.reference_mo(bedtools="bedtools/binary/path",
                GFFgen="gtf/file/to/reference/genome/path",
                genome="reference/genome/in/fasta/format/path",
                trme="transcriptome/in/fasta/format/path",
                RMgen="RepeatMasker/output/file/from/transcriptome/path",
                RMtr="RepeatMasker/output/file/from/transcriptome/path",
                overlapping=T,
                over.res="HS",
                by="namRep",
                threads=15,
                outdir="output/directory/path")
```

Once the decoys files (trmeSalmon.fasta and decoys.txt) are obtained you are ready to run salmon as described in section 2.3.

2.2 Make TEs references in non-model organisms

When there is no reference genome, ExplorATE takes a TransDecoder-generated gene model file and a blast-generated annotation file (TransDecoder.gff3 and BLAST.outfmt6 in the example below) to identify those repeats overlapping with protein-coding genes. Once the transcripts with co-transcribed repeats are identified, they are excluded from the RepeatMasker output file. The user can further refine his RepeatMasker file to resolve overlaps or apply selection criteria as shown in sections 5 and 6. A general flow chart of the pipeline is shown in the figure below.

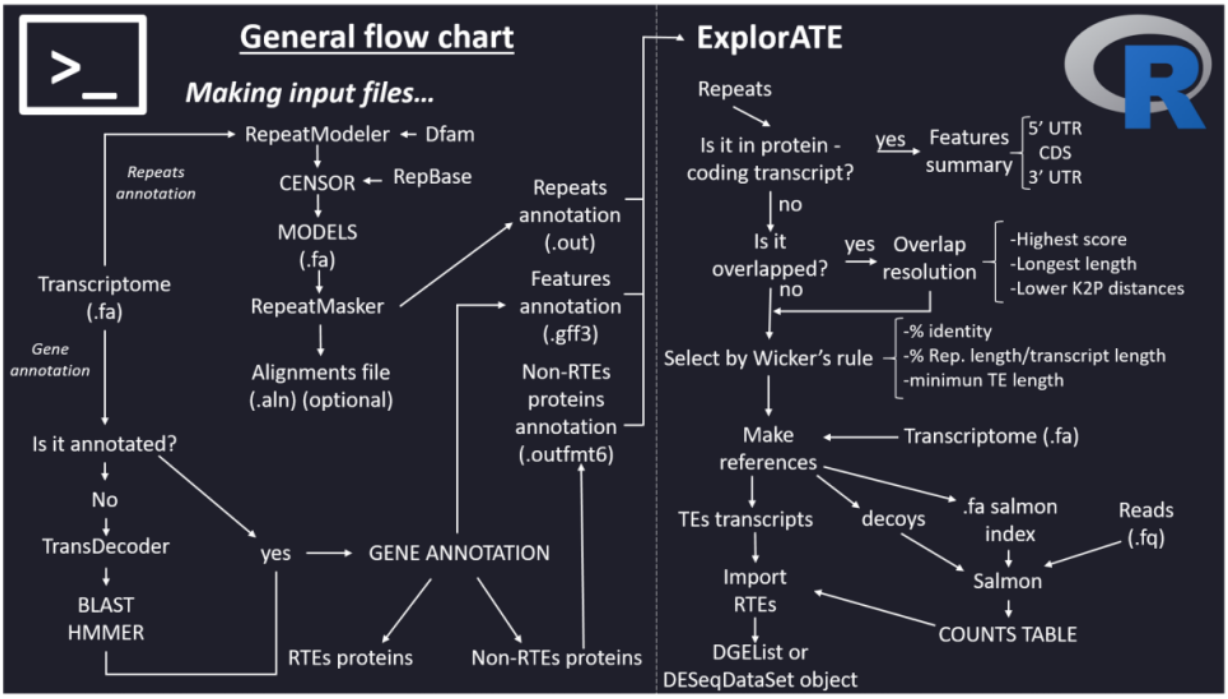


Figure 1: alt text

The user can run these steps separately or do them all together with the mk.reference() function as shown

below.

```
RM.reference <- ExplorATE::mk.reference(RepMask = "path/to/RepeatMasker.out",
                                        gff3 = "path/to/TransDecoder.gff3",
                                        anot = "path/to/BLAST.outfmt6",
                                        stranded = T,
                                        cleanTEsProt = F,
                                        featureSum = T,
                                        outdir = "./outdir",
                                        rm.cotrans = T,
                                        overlapping = T,
                                        align = "path/to/RepeatMasker.aln",
                                        over.res = "LD",
                                        ignore.aln.pos = T,
                                        threads = 18,
                                        trme = "path/to/transcriptome.fa",
                                        by = "classRep",
                                        rule = c(80,80,80)
                                    )
```

With this function a reference file will be created containing the class/family of transposon for each transcript, initially excluding the co-transcribed repetitions and applying a selection criteria based on like-Wicker's rule. In addition, the files `trmeSalmon.fasta` and `decoys.txt` will be created to run Salmon later.

When executing the `mk.reference()` function, it will initially ask us if the assigned classes/families are correct. You should check that there are no ambiguities in the labeling before executing the function. If the assigned classes/families are correct, you need to type `'y'+Enter` in the console.

2.3 Run Salmon and import count estimates

After creating the reference file, the next step is to perform the quantification estimate with Salmon. Linux users can use the `run.salmon()` function in R or run Salmon with the appropriate arguments.

```
ExplorATE::run.salmon(index = "sampleIndex",
                      decoys = "decoys.txt",
                      salmon_path = "path/to/salmon-latest_linux_x86_64/bin/salmon",
                      kmer = 31,
                      threads = 18,
                      trme = "path/to/trmeSalmon.fasta",
                      lib_dir = "path/to/reads",
                      pe_se = "pe"
                      )
```

The above function uses the `decoys.txt` and `trmeSalmon.fasta` files generated by `mk.reference()` or `mk.reference_mo()` functions. If you decide to run Salmon in another way, be sure to use these files and the –gcBias, –validateMappings, –useVBOpt flags. You will find the details in Patro et al. (2017), Love et al. (2016) and Srivastava et al. (2020).

If you have any questions about how to run the program you can consult the documentation of Salmon.

Finally the estimates are imported from Salmon. The `ExplorATE::import.RTEs()` function allows the estimates to be imported into R and creates a DGEList or DESeqDataSet object with the estimates corrected for changes in the average length of the transcripts across the samples.

```
y <- ExplorATE::import.RTEs(
  path.sal = "path/to/salmon/output",
  conditions = rep(c("S1", "S2", "S3"), each = 3),
  ref.sal = RM.reference,
```

```
  import_to = "edgeR"
)
```

The y object is ready to be used for differential expression analysis in edgeR. If you will use DESeq2 for later differential expression analysis, you must use the `import_to = "DESeq2"` argument. If the `import_to =` argument is omitted, the transcript-level estimates will be imported.

3. MAKING INPUT FILES

Before using ExplorATE, you must generate/download files that contain detailed annotations of the repeats present in the transcriptome and/or genome, and the coding-proteins transcripts. Also, you'll want to generate some files that help refine the TEs annotation. If you are working with a model organism you must download its reference genome and the appropriate GFF file for the genome version. RepeatMasker annotations to the model organisms can be found on the program's website. You will then need to do a de novo assembly of your transcriptome and run RepeatMasker with the appropriate library for your organism. When working with a non-model organism you must first annotate the transcriptome. The mk.inputfiles.sh script runs identify candidate coding regions within transcript sequences with TransDecoder and performs ORF search by homology against the Pfam (using HMMER) and UniProt/SwissProt (using BLAST) databases. Finally, it runs RepeatMasker against a user-specified library. The transposon library should be as close as possible to the species of interest. Some users may want to perform *de novo* identification of elements with RepeatModeler and subsequently re-annotate "unknown" sequences using CENSOR to generate libraries specific to their organism of interest. You can find useful information on how to generate *de novo* libraries and combine them with RepeatMasker libraries on the RepeatModeler web page and at this link. In other cases, curated libraries of elements such as Dfam and RepBase could directly provide reliable information on your studied organism. The more specific the library used, the lower the bias related to the age of the transposable element, which is why we recommend carefully selecting the library to use. Once the library is defined, the user is ready to run the script as shown below: First, download the script mk.inputfiles.sh

```
wget https://github.com/FemeniasM/ExplorATEtools/mk.inputfiles.sh
```

Then, run the script assigning the appropriate arguments:

```
bash mk.inputfiles.sh -p threads -b <blastp binary path> -h <hmmer binary path>
-r <RepeatMasker binary path> -d <TransDecoder directory path> -s <swissprot database>
-f <Pfam database> -l <Repeats library> -t <transcriptome file> -o <output directory>
```

By default `blastp`,`hmmscan` and `RepeatMasker` are assumed to be available in your `$PATH` environment variable.

3.1 RepeatMasker repeats annotations

To create a reference file, you must first enter a file that contains detailed annotations of the repeats present in the transcriptome. As mentioned before you must identify repeats in transcriptome using RepeatMasker against selected libraries, either Dfam (profiles HMM derived from Repbase), Repbase, or user-defined. You can also perform TE family identification and modeling with RepeatModeler and build specific libraries for your studied organism. RepeatMasker also allows you to generate an alignment file `.aln` along with the output. This file can be used later in the resolution of overlapping repeats within the same transcript. You can use the `read.RepMask()` function to read the RepeatMasker output file and verify it. The function `read.alignfile()` allows reading the alignment file and returns a `data.frame` with the identification of the sequence, the Family of the assigned repetition. You can average the distances per sequence or per family of TEs (you can run `?read.alignfile()` for more details).

Some examples are shown below:

```
RM <- ExplorATE::read.RepMask("RM.out")
ALN <- ExplorATE::read.alignfile("RM.aln", average = T, by="classRep")
```

3.1.1 Avoid ambiguities in repeats annotations It is possible that if different libraries are consulted there will be differences in the labeling of some repetitions. These ambiguities will cause problems later in the workflow so we must eliminate them. When we execute the function `rm.cotransRep()` it will ask us if the names assigned to the repetitions are correct (see the section 4). The example below shows how we can unify the names. Suppose your RepeatMasker file contains ambiguities for SINEs elements and you want to unify according to Repbase groups.

First we check the repeats names, for this you can run these simple commands:

```
RM <- ExplorATE::read.RepMask("RM.out")
sort(unique(RM$classRep))
```

If we find elements identified as `"NonLTR/SINE/7SL"` or `"NonLTR/SINE/tRNA"`, we can simply replace them with the correct assignment as follows:

```
RM$classRep <- gsub("NonLTR/SINE/7SL", "NonLTR/SINE/SINE1", RM$classRep)
RM$classRep <- gsub("NonLTR/SINE/tRNA", "NonLTR/SINE/SINE2", RM$classRep)
```

3.2 Protein-coding genes annotations

You can enter a reference file in `.outfmt6` format that is obtained from the output of annotators such as BLAST or dammit!. You must remove all annotations that correspond to transposon proteins such as reverse transcriptases, endonucleases, transposases, tyrosine recombinases, etc. Only in exploratory analyses you can enter your `.outfmt6` file without removing proteins related to transposons and use a local database to remove them. In this case, the annotations must be in UniProt/Swiss-Prot ID format such as "CO1A2_MOUSE" or "sp|Q01149|CO1A2_MOUSE" or "tr|H9GLU4|H9GLU4_ANOCA". You can assign the annotations file to an object and explore it with basic R functions, for example:

```
GENE.ANOT <- read.outfmt6("BLAST.outfmt6")
head(GENE.ANOT)
```

When repeats are co-transcribed with coding genes, you may want to know where this repeat is. You can supply a `.gff` file with the features for each transcript. In this way you find a detailed output of the repeats found in 5'-UTR, 3'-UTR or CDS regions. You can generate a `.gff` file with TransDecoder. We recommend running it with BLASTP and Pfam searches, and retaining only the best ORF. In the same way as the annotations file, the `.gff` file can be assigned an object and explored with the basic R functions:

```
GFF3 <- read.gff3("TransDecoder.gff3")
head(GFF3)
```

4. FILTERING COTRANSCRIBED REPEATS

Many of the repeats that were annotated in our transcriptome did not truly correspond to transposons. Some repeats may be co-transcribed with genes, therefore the first step for correct transposon annotation is the removal of repeats associated with coding genes. The `rm.cotransRep()` function remove this repeats from our RepeatMasker file.

```
RM.cotransClean <- ExplorATE::rm.cotransRep(RepMask = "RM.out",
                                            gff3 = "TransDecoder.gff3",
                                            anot = "BLAST.outfmt6",
                                            stranded = T,
                                            cleanTEsProt = F,
                                            featureSum = T,
                                            outdir = "./outdir")
```

In the above function you can assign both the input files' path and the assigned objects within the R environment. Therefore if you made modifications of these files within the R environment, you can assign them as follows:

```
RM <- ExplorATE::read.RepMask("RM.out")
GENE.ANOT <- read.outfmt6("BLAST.outfmt6")
GFF3 <- read.gff3("TransDecoder.gff3")

RM$classRep <- gsub("NonLTR/SINE/7SL", "NonLTR/SINE/SINE1", RM$classRep)
RM$classRep <- gsub("NonLTR/SINE/tRNA", "NonLTR/SINE/SINE2", RM$classRep)

RM.cotransClean <- ExplorATE::rm.cotransRep(RepMask = RM,
                                            gff3 = GFF3,
                                            anot = GENE.ANOT,
                                            stranded = T,
                                            cleanTEsProt = F,
                                            featureSum = T,
                                            outdir = "./outdir")
```

The argument `cleanTEsProt` refers to whether the annotations file `*.outfmt6` contains TEs-related proteins. We suggest that the user previously carefully remove TEs-related proteins from the `*.outfmt6` file and by default this argument is `cleanTEsProt = FALSE` (*i.e.* by default the program will not remove such proteins, if you want to change the default parameter you must assign `cleanTEsProt = TRUE`, see section 3.2).

If the argument `featureSum = T` additional files are created containing a summary of the protein-coding transcripts carrying repeats (`features.summary.*`).

5. RESOLVING OVERLAPPING

The RepeatMasker output file may contain overlapping repetitions when the program cannot resolve them automatically. You can resolve these overlaps with the python script provided by the creators of the program, or you can work it within the R environment with the function `ovlp.res()` Like the python script, the `ovlp.res()` function allows to solve the overlaps considering the higher score ("HS"), a longer length ("LE") or lower Kimura's distances ("LD"), that are assigned with the argument `over.res =`. When two repeats partially overlap, the overlapping bases are assigned to the item with the best score. If one repeat is contained within another, the repeat with the lowest score is discarded. If two items have the same score, they are assigned based on the first item in the RepeatMasker file.

```
RM.ovlp.res <- ExplorATE::ovlp.res(RepMask = "RM.out",
                                   gff3 = "TransDecoder.gff3",
                                   anot = "BLAST.outfmt6",
                                   stranded = T,
                                   cleanTEsProt = F,
                                   featureSum = T,
                                   outdir = "./outdir",
                                   rm.cotrans = T,
                                   ignore.aln.pos = T,
                                   threads = 18,
                                   align = "RM.aln",
                                   over.res = "LD",
                                 )
```

You can enter a raw RepeatMasker file, in this case if you want to remove co-transcribed repeats you must select the `rm.cotrans = T` argument. As mentioned in the previous section, we recommend that you perform the removal of proteins associated with transposable elements in the `.outfmt6` file and therefore you should assign `cleanTEsProt = F`. Alternatively you can enter with the argument `RepMask =` a preprocessed RepeatMasker file in which the removal of co-transcribed repetitions has already been performed:

```
RM.ovlp.res <- ExplorATE::ovlp.res(RepMask = RM.cotransClean,
                                   align = "RM.aln",
```

```
                                    over.res = "LD",
                                    ignore.aln.pos = T,
                                    threads = 18,
                                  )
```

Notice that the output directory `outdir =` is only required if `featureSum = T` and the alignments file `align =` is only required if "low divergence" is used as the resolution parameter (`over.res = "LD"`).

When the resolution of overlaps is done by "low divergence" (LD), the `ignore.aln.pos` argument must be added. Since there may be discrepancies in the positions between the alignment file and RepeatMasker output file, you can select whether or not to take them into account. If `ignore.aln.pos = T` the positions are ignored and the mean of class/family of repeats per transcript is taken.

The `ovlp.res()` function allows parallel processing. Select the number of cores to use with the `threads =` argument.

6. TRANSPOSONS ANNOTATION AND MAKING REFERENCE FILES

After cleaning our RepeatMasker file of co-transcribed repeats and resolve overlaps, the user must annotate the transcripts that potentially correspond to active TEs. These annotated transposons are used to generate a reference file and all remaining transcripts will be used as decoys in the estimation of subsequent quantification. To create these files directly you can use the `mk.reference ()` function or use a criterion based on a like-Wicker rule Wicker et al. (2007). By default the rule 0-0-0 but the user can modify it to discretion with the `rule =` argument. For example, an 80-50-150 rule means that only those transcripts that have 80% identity in 50% of the transcript and at least 150bp in length will be annoted as TEs. Like the above functions you can supply a clean RepeatMasker file or a raw RepeatMasker file and assign the appropriate arguments to perform the cleaning as shown in the examples below:

```
RM.reference <- ExplorATE::mk.reference(RepMask = "RM.out",
                                        gff3 = "TransDecoder.gff3",
                                        anot = "BLAST.outfmt6",
                                        stranded = T,
                                        cleanTEsProt = F,
                                        featureSum = T,
                                        outdir = "./outdir",
                                        rm.cotrans = T,
                                        overlapping = T,
                                        align = "RM.aln",
                                        over.res = "LD",
                                        trme = "transcriptome.fa",
                                        ignore.aln.pos = T,
                                        threads = 18,
                                        by = "classRep",
                                        rule = c(90,80,100)
                                      )
```

In this example the argument `overlapping = T` indicates that our input file still needs to resolve the overlaps. Furthermore, the argument `rule = c(90,80,100)` indicates that transcripts that have 90% identity in 80% (or more) of the transcript with more than 100bp in length will be annotated. The `by =` argument indicates whether the classification should be done at the class, superfamily or family level (not recommended for exploratory analysis without reference genome). If you select `by = "classRep"` the column "classRep" from the RepeatMasker file will be used.

If you start from a processed file, *i.e.* without co-transcribed repeats or overlaps, you could run a code like the following:

```
RM.reference <- ExplorATE::mk.reference(RepMask = RM.ovlp.res,
                                        outdir = "./outdir",
                                        trme = "transcriptome.fa",
                                        by = "classRep",
                                        rule = c(90,80,100)
                                      )
```

The `mk.reference()` function returns a `data.frame` with annotated TEs transcripts and creates three files in the output directory: a `reference.csv` file with the annotations, a `decoy.txt` file and a `trmeSalmon.fasta` that will be used in quantification with Salmon.

7. SALMON QUANTIFICATION ESTIMATION

The Salmon program is widely used in the transcripts quantification because it is very fast and accurate. You can run Salmon locally or, if you are a Linux user, you can run it through the `run.salmon` function. If you have questions about how to run the program locally, please see the detailed Salmon's documentation. You must make sure to use the –gcBias and –validateMappings flags and the `decoy.txt` file generated by the `mk.reference ()` function. Details of these arguments can be found in Patro et al. (2017), Love et al. (2016) and Srivastava et al. (2020).

Linux users can run Salmon with the `run.salmon` function, however it requires that the program be already installed on their computer. An example to execute the function is shown below:

```
ExplorATE::run.salmon(index = "sampleIndex",
                      decoys = "decoy.txt",
                      salmon_path = "path/to/salmon-latest_linux_x86_64/bin/salmon",
                      kmer = 31,
                      threads = 18,
                      trme = "path/to/trmeSalmon.fasta",
                      lib_dir = "path/to/reads",
                      pe_se = "pe"
                      )
```

8. IMPORT ESTIMATES TO R FOR DIFFERENTIAL EXPRESSION ANALYSIS

The last step is to import the quantification estimates into the R environment and create a DGEList or DESeqDataSet object to perform the subsequent differential expression analysis. The `import.RTEs()` function imports the estimates using the `tximport` package (see Soneson et al. (2015) for more details). The `import.RTEs()` function directly creates an offset that corrects the estimates for changes in the average transcripts length across samples. The following code shows an example to create a DGEList object:

```
y <- ExplorATE::import.RTEs(
  path.sal = "path/to/salmon/output",
  conditions = rep(c("S1", "S2", "S3"), each = 3),
  ref.sal = RM.reference,
  import_to = "edgeR"
)
```

In the example above, the argument `import_to = "edgeR"` was selected to create a DGEList object. Additionally this function will add a CPMs matrix to the DGEList object. Now you can continue with the dispersion estimation functions in edgeR. For more details, see the edgeRUserGuide. More details of the edgeR package can be found in Robinson et al. (2010).

Alternatively, you can select `import_to = "DESeq2"` and continue with the `DESeq ()` function (see DESeq2 vignette and Love et al. (2014) for more details). If neither option is selected, the transcript-level estimates

will be imported.

9. BIBLIOGRAPHY

Haas, B., Papanicolaou, A., Yassour, M. et al. De novo transcript sequence reconstruction from RNA-seq using the Trinity platform for reference generation and analysis. Nat Protoc 8, 1494–1512 (2013). https://doi.org/10.1038/nprot.2013.084

Love, M. I., Huber, W. & Anders, S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biol 15, 550 (2014). https://doi.org/10.1186/s13059-014-0550-8

Love, M. I., Hogenesch, J. & Irizarry, R. Modeling of RNA-seq fragment sequence bias reduces systematic errors in transcript abundance estimation. Nat Biotechnol 34, 1287–1291 (2016). https://doi.org/10.1038/nbt.3682

Patro, R., Duggal, G., Love, M. I. et al. Salmon provides fast and bias-aware quantification of transcript expression. Nat Methods 14, 417-419 (2017). https://doi.org/10.1038/nmeth.4197

Robinson M. D., McCarthy D. J. , Smyth G. K. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26(1), 139–140 (2010). https://doi.org/10.1093/bioinformatics/btp616

Smit, A., Hubley, R & Green, P. RepeatMasker Open-4.0, (2013-2015). http://www.repeatmasker.org

Soneson C., Love M. I. and Robinson M. D. Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences. F1000Research 4(1521), (2015). https://doi.org/10.12688/f1000research.7563.1

Srivastava, A., Malik, L., Sarkar, H. et al. Alignment and mapping methodology influence transcript abundance estimation. Genome Biol 21, 239 (2020). https://doi.org/10.1186/s13059-020-02151-8

The UniProt Consortium. UniProt: a worldwide hub of protein knowledge, Nucleic Acids Research 47(D1), D506–D515 (2019). https://doi.org/10.1093/nar/gky1049

Wicker, T., Sabot, F., Hua-Van, A. et al. A unified classification system for eukaryotic transposable elements. Nat Rev Genet 8, 973–982 (2007). https://doi.org/10.1038/nrg2165