Package 'ReferenceEnhancer'

September 8, 2023

Title: Package for optimizing and assembling genome annotations for 3' single-cell RNA-sequencing analysis.

Description: ReferenceEnhancer contains a set of tools for optimizing genome annotations for droplet based 3' single-cell RNA- sequencing (10x Genomics, Dropseq etc.) data analysis. Regular genome annotations and transcriptomic references generated based on them come with several problems causing discarded sequencing data from final gene expression estimates (outlined in detail in https://www.biorxiv.org/content/10.1101/2022.04.26.489449v1). These include read loss stemming from gene overlaps, sequencing reads mapping to 3' unannoated exons as well as introns. ReferenceEnhancer enables fixing these issues and assembling optimized genome annotations that circumvent these problems and recover the discarded gene expression data.

URL: https://github.com/PoolLab/ReferenceEnhancer

Version: 1.0

Depends: R(>=4.0.0, GenomicRanges(>=1.50.2), GenomicAlignments (>=1.34.1), stringr(>=1.5.0), GenomicFeatures (>=1.50.4), gdata(>=2.18.01), rtracklayer, BEDOPS(>v 2.0), bedtools(v.2.26.0).

License: Artistic-2.0

Maintainer: Helen Poldsam (helen.poldsam@utsouthwestern.edu), Allan-Hermann Pool (allan-hermann.pool@utsouthwestern.edu)

1. Installation of ReferenceEnhancer

```r
install.packages("devtools")
require(devtools)
install_github("PoolLab/ReferenceEnhancer")
```

2. Basic workflow

This is the basic workflow for optimizing a genome annotation for single-cell RNA-seq work using ReferenceEnhancer:

1. Load ReferenceEnhancer and import ENSEMBL/10x Genomics default genome annotation file in General Transfer Format (GTF).

   This file can be downloaded from 10x Genomics provided reference transcriptome "gene" folder at https://support.10xgenomics.com/single-cell-gene-expression/software/downloads/latest or Ensembl.org if wish to customize more.

   ```r
   library(ReferenceEnhancer)
   ```

```
genome_annotation <- LoadGtf(unoptimized_annotation_path)
```

Example:

```
genome_annotation <- LoadGtf(unoptimized_annotation_path =
"test_genes.gtf")
```

2. Identify all overlapping genes based on the ENSEMBL/10x Genomics default genome annotation file (GTF), rank-order them according to the number of gene overlaps.

   Prioritize this gene list for manual curation focusing on exonically overlapping genes. The function saves the list of overlapping genes in working directory as overlapping_gene_list.csv.

   ```
   gene_overlaps <- IdentifyOverlappers(genome_annotation)
   ```

   Example:

   ```
   gene_overlaps <- IdentifyOverlappers(genome_annotation =
   genome_annotation)
   ```

3. Generate recommended actions for overlapping genes based on original genome annotation GTF file and a list of overlapping genes. gene_pattern specifies which genes to treat as pseudo- or low value genes. *This step is optional.*

   The function updates overlapping_gene_list.csv file with added recommendations.

   ```
   OverlapResolutions(genome_annotation, overlap_data, gene_pattern)
   ```

   Example:

   ```
   OverlapResolutions(genome_annotation = genome_annotation, overlap_data
   = gene_overlaps, gene_pattern = c("Rik$", "^Gm"))
   ```

4. Manual curation of overlapping gene list. Results in "overlapping_gene_list.csv" specifying which genes and transcripts need deleting. *This step is optional.*

   a. For gene deletion, list "Delete" in the "final_classification" column.

   b. For transcript deletion, list all transcripts that need deleting under "transcripts_for_deletion" column separated by ", ".

   c. Consult the specific respective genome annotation in the Ensembl genome browser or equivalent.

   d. This step can result in “rename_genes.csv” specifying which genes require renaming. If two distinct genes have completely overlapping final exons that cannot be otherwise disentangled, consider listing one of the genes for deletion and rename the remaining gene for clarity (add old gene name under "old_name" column in the "rename_genes.csv" and specify new name under "new_name" column in the same file).

5. Extract intergenic reads from Cell Ranger aligned bam file. The function saves extracted intergenic reads in working directory as intergenic_reads.bed.

```
IsolateIntergenicReads(bam_file_name, index_file_name, barcode_length)
```

Example:

```
IsolateIntergenicReads(bam_file_name = "test_bam.bam", index_file_name
= "test_index.bam.bai", barcode_length = 26)
```

6. Generate gene boundaries in order to assign intergenic reads to a specific gene. The function saves results in working directory as gene_ranges.bed.

   Note: This step runs partially in bash/linux terminal. Before this step, make sure that bedops (https://bedops.readthedocs.io/en/latest/) has been installed to your computer.

   ```
   GenerateGeneLocationBed(genome_annotation, bedops_loc)
   ```

   Example:

   ```
   GenerateGeneLocationBed(genome_annotation = genome_annotation,
   bedops_loc = NULL)
   ```

7. Identify candidate genes for extension with excess 3' intergenic reads and create a rank ordered list of genes as a function of 3' intergenic read mapping within 10kb of known gene end. A rank ordered list of gene extension candidates is saved in working directory as gene_extension_candidates.csv.

   Note: This step runs partially in Bash/Linux terminal. Before this step, make sure that bedtools (https://bedtools.readthedocs.io/en/latest/content/installation.html) has been installed to your computer and that it has been added to the path in your R environment.

   ```
   GenerateExtensionCandidates(bedtools_loc)
   ```

   Example:

   ```
   GenerateExtensionCandidates(bedtools_loc = NULL)
   ```

8. Create the final optimized annotation file. The function saves the result in working directory as optimized_reference in GTF.

   ```
   OptimizedAnnotationAssembler(unoptimized_annotation_path,
   gene_overlaps, gene_extension, gene_replacement)
   ```

   Example:

   ```
   OptimizedAnnotationAssembler(unoptimized_annotation_path =
   "test_genes.gtf", gene_overlaps = "test_overlapping_gene_list.csv",
   gene_extension = "gene_extension_candidates.csv", gene_replacement =
   "test_gene_replacement.csv")
   ```

3. LoadGTF

Description

Use to import the Ensembl/10x Genomics default genome annotation or other desired genome annotation file in GTF format for optimization for scRNA-seq analysis. Note: This file can be downloaded from 10x Genomics provided reference transcriptome "gene" folder at https://support.10xgenomics.com/single-cell-gene-expression/software/downloads/latest or Ensembl.org if wish to customize more.

Usage
```
LoadGtf(unoptimized_annotation_path)
```

Arguments

`unoptimized_annotation_path`	Path to the unoptimized genome annotation GTF file.

Value
Resulting object contains the genome annotation entries from the genome annotation GTF file.

Examples
```
LoadGtf(unoptimized_annotation_path = "test_genes.gtf")
```

4. IdentifyOverlappers

Description
Identifies all same-strand overlapping genes based on the unoptimized genome annotation file in GTF, rank-orders them according to the number of gene overlaps. Saves the list of overlapping genes in working directory as “overlapping_gene_list.csv”.

Usage
```
IdentifyOverlappers(genome_annotation)
```

Arguments

`genome_annotation`	Unoptimized genome annotation file in GTF. Could be obtained from Ensembl, Refseq, 10x Genomics or elsewhere.

Value
Rank-ordered gene list of same-strand overlapping genes (“overlapping_gene_list.csv”).

Examples
```
genome_annotation <- LoadGtf(unoptimized_annotation_path = "test_genes.gtf")
IdentifyOverlappers(genome_annotation = genome_annotation)
```

5. OverlapResolutions

Description
Based on original genome annotation GTF file and a list of overlapping genes, generates recommended actions for overlapping genes. This is an optional step that can help with decision making during the manual curation step.
Gene overlaps can be resolved by one of several strategies including (i) leaving overlapping gene annotations unchanged if their exons don’t directly overlap, (ii) deleting offending readthrough transcripts from upstream genes, (iii) deleting offending premature gene transcripts from downstream genes, (iv) deleting pseudogenes and non-protein coding genes with poor support and no read mapping that obscure well established protein coding genes or (v) for extensively overlapping genes deleting one and renaming the other to capture otherwise discarded reads. As well annotated genomes contain several thousand same-strand overlapping genes and properly resolving gene overlaps often requires manual inspection of the locus to determine best course of action, prioritization of genes for manual curation is often desirable. To this end, *OverlapResolutions* function classifies genes to prioritize for direct inspection. The following algorithm is used to classify genes for appropriate curation:

1. If gene overlaps with multiple genes:
    a. If gene's exons overlap with another gene's exons ➔ classify for "Manual inspection"
    b. If gene's exons do not overlap with any other genes' exons ➔ classify as "Keep as is"
    c. Assign recommended action for overlapping genes
        i. If nested gene does not overlap with any other gene ➔ classify as "Keep as is"
        ii. If nested gene overlaps with more than one gene ➔ classify for "Manual inspection"
2. If gene overlaps with only one other gene, test whether gene is non-protein coding/pseudogene ("Gm" and "…Rik" gene models in mice; "AC…" and "AL…" gene models in humans)
    a. If both overlapping genes are non-protein coding/pseudogenes ➔ classify for "Manual inspection"
    b. If only one gene in the overlapping gene pair is non-protein coding/pseudogene, test if genes have overlapping exons:
        i. In case no overlapping exons ➔ classify both genes as "Keep as is"
        ii. In case exons overlap ➔ mark non-protein coding/pseudogene for deletion ("Delete")
    c. If both genes are well supported genes:
        i. If their exons don't overlap ➔ mark both genes as "Keep as is"
        ii. If their exons do overlap, determine the number of opposing gene's exonic overlap for each exon of each gene and find the exon with most overlaps for both upstream and downstream gene to determine appropriate course of action:
            1. If downstream gene's exon has more overlaps than its upstream counterpart, classify downstream gene as "Premature transcript deletion" and upstream gene as "Keep as is"
            2. If upstream gene's exon has more overlaps than its downstream counterpart, classify upstream gene as "Readthrough transcript deletion" and downstream gene as "Keep as is"
            3. Otherwise classify both for "Manual inspection"

The resulting recommendations can be used in the manual curation step, where all genes that are not classified in the "Keep as is" category should directly be scrutinized in the Ensembl genome browser (ensemble.org, with the correct genome builds) and/or cross-referenced to the respective Refseq genome annotation within the Integrated Genome Browser (IGV 2.11.9).

Usage

```
OverlapResolutions(genome_annotation, overlap_data, gene_pattern)
```

Arguments

`genome_annotation`	Unoptimized genome annotation (e.g. Ensembl/10x Genomics) default genome annotation GTF file. This should be a dataframe created with the *LoadGtf()* function in this package.
`overlap_data`	A list of overlapping genes generated by *IdentifyOverlappers*.
`gene_pattern`	The pattern in gene names, that is unique for pseudo- or other low quality or low interest genes. Patterns for recognizing candidate pseudo- or low quality genes can be defined with regular expressions for matching gene names with a given pattern. See vignette (regular-expressions) in the *stringr* package for details or examples below.

Value

Generates "overlapping_gene_list.csv" with added recommendations for resolving gene overlaps in the "automatic_classification" column.

Examples

```
genome_annotation <- LoadGtf(unoptimized_annotation_path = "test_genes.gtf")
gene_overlaps <- IdentifyOverlappers(genome_annotation = genome_annotation)
```

```
OverlapResolutions(genome_annotation = genome_annotation, overlap_data =
gene_overlaps, gene_pattern = c("^Gm", "Rik$"))
```

Note: The example treats genes starting with “Gm…” and ending with “…Rik” as pseudogenes. Additional patterns for recognizing candidate pseudo- or low-quality genes can be defined with regular expressions for matching gene names with a given pattern. See vignette(regular-expressions) in the stringr package for details.

6. Manual curation of overlapping gene list

Inspect gene overlaps and transcript structure in Ensembl (https://nov2020.archive.ensembl.org/Mus_musculus/Info/Index for mice and https://useast.ensembl.org/Homo_sapiens/Info/Index for humans) to determine which read-through and premature start transcripts to eliminate (mark under "transcripts_for_deletion" column in "overlapping_gene_list.csv"). Make sure you look up the correct genome build in Ensembl.
For pseudogenes/poorly supported gene models that obscure protein coding genes, examine evidence for its existence (are they reported by other annotation consortia - e.g. Refseq). You can also examine if any reads map to it with regular exonic reference and determine whether to keep or eliminate it from the annotation. To mark a gene for deletion, mark its status as "Delete" in the "final_classification" column in the “overlapping_gene_list.csv”.
For genes that cannot be unentangled due to overlap of terminal exons, delete one of them and rename the other to reflect that reads could originate from both. To this end, mark one of the genes for deletion (add "Delete" under "overlapping_gene_list.csv" file’s “final_classification” column). Also, create a new .CSV file "rename_genes.csv" where under column "old_names" you can list gene names that need to be renamed and under column "new_names" list the new hybrid gene name. See https://github.com/PoolLab/generecovery/tree/main/mouse_mm10_input_files for a formatting example.

7. IsolateIntergenicReads

Description
Intergenic reads are extracted from Cell Ranger aligned bam file. Use a scRNA-seq dataset of interest that has been aligned to the unoptimized genome reference with the Cell Ranger count pipeline. Intergenic reads can be identified by two features: their read identity tag RE = "I" (for intergenic) OR their RE=E (for exonic) with AN = <some gene>. The latter reads are in fact intergenic reads since Cell Ranger wrongly classifies reads mapping antisense to an exon as exonic (i.e. RE="E"). The false exonic reads can be recognized and captured as proper intergenic reads by extracting two kinds of reads (RE=I and RE=E & AN=<something else than NA). Also, removing duplicates command in GenomicAlignments package does not work for intergenic (nor for intronic) reads. Duplicate and corrupt read removal has to be done manually (i.e. make sure cellular and molecular barcodes have specified lengths and duplicate barcodes removed).

Note that bam files can often be many tens of gigabytes and thus this step is highly memory intensive.

Usage
```
IsolateIntergenicReads(bam_file_name, index_file_name, barcode_length)
```

Arguments

`bam_file_name`	Path to Cell Ranger generated bam file (run Cell Ranger count pipeline on sequencing data of interest and aligning it to the unoptimized transcriptomic reference).
`index_file_name`	Path to Cell Ranger generated bam.bai file (run Cell Ranger count pipeline on sequencing data of interest and aligning it to the unoptimized transcriptomic reference).
`barcode_length`	Optional. Specifies the length of barcode needed. If not specified, defaults to 26.

Value
Saves extracted intergenic reads as a separate file (“intergenic_reads.bed”)

Examples
```
IsolateIntergenicReads(bam_file_name = "test_bam.bam", index_file_name  =
"test_index.bam.bai", barcode_length = 26)
```

8. GenerateGeneLocationBed

Description
Makes a bed file with gene boundaries, which is required for assigning intergenic reads to a specific gene and discovering genes with large amounts of intergenic reads near its 3’ gene end.

Note 1: This step is partially run in Linux Terminal in Bash and requires BEDOPS (https://bedops.readthedocs.io/en/latest/). Make sure BEDOPS is installed and provide a path to BEDOPS in the function if you get an error message.

Note 2: In Linux terminal, navigate to folder with the genome annotation of interest. The annotation file should be named "genes.gtf" per 10x Genomics convention.

Usage
```
GenerateGeneLocationBed(genome_annotation, bedops_loc)
```

Arguments

`genome_annotation`	Genome annotation DataFrame loaded with LoadGtf() function in this package.
`bedops_loc`	Optional. Location of BEDOPS in file system.

Value
Saves “gene_ranges.bed” in working directory.

Examples
```
genome_annotation <- LoadGtf(unoptimized_annotation_path = "test_genes.gtf")
GenerateGeneLocationBed(genome_annotation = genome_annotation, bedops_loc  =
NULL)
```

9. GenerateExtensionCandidates

Description
Identifies candidate genes for extension with excess 3' intergenic reads and creates a rank ordered list of genes as a function of 3' intergenic read mapping within 10kb of known gene end. You can use this as a prioritized gene list for gene extension to examine in Integrated Genomics Viewer.
Note: It runs partially in Bash/Linux terminal. Make sure bedtools is installed and in PATH variable in Linux or MacOS.

Usage
```
GenerateExtensionCandidates(bedtools_loc)
```

Arguments

`bedtools_loc`	Optional. Location of bedtools in file system.

Value
Rank ordered list of gene extension candidates saved to working directory as “gene_extension_candidates.csv”.

Examples

```
GenerateExtensionCandidates(bedtools_loc = NULL)
```

10. Manual curation 3' gene ends

Download and install https://software.broadinstitute.org/software/igv/. Load the transcriptome aligned sequencing read data ("xxx.bam" file) to the Integrative Genomics Viewer (IGV) and choose appropriate species/genome. Go through the rank ordered candidate gene list in "gene_extension_candidates.csv" and examine evidence for unannotated 3' exons and UTRs:

a) Extensive splicing between annotated and candidate unannotated regions at the genetic locus as visualized by IGV.
b) Continuous read mapping from known gene end (delayed cutoff of sequencing reads posterior from known gene end).
c) Examine external evidence (e.g. Allen *In Situ* Brain Atlas, Human Protein Atlas etc.).

Based on the latter, provide new gene boundaries under two new columns: "update_start" and "update_end" depending on whether gene is on "-" or "+" strands, respectively. Use Excel formulas, if necessary, to scale curation, and save file as "gene_extension_candidates.csv". The resulting manually curated "gene_extension_candidates.csv" will be used as an input for the OptimizedAnnotationAssembler function.

11. OptimizedAnnotationAssembler

Description

OptimizedAnnotationAssembler generates the scRNA-seq optimized genome annotation. The resulting optimized genome annotation can be used to generate the transcriptomic reference for mapping single-cell sequencing data (e.g. with cellranger mkref or STAR --runMode genomeGenerate). Note that completing this step is time intensive and can sometimes take 12-24 hours depending on the length of the annotation to be optimized. This function goes through the following steps:

0. Loads data and libraries:
    - Genome annotation file to be optimized in GTF.
    - "overlapping_gene_list.csv" file specifying how to resolve gene overlap derived issues. "Delete" entries in $final_classification field mark genes for deletion. Transcript names in $transcripts_for_deletion mark specific transcripts for deletion.
    - "gene_extension_candidates.csv" specifying updated gene boundaries for incorporating intergenic reads.
    - "rename_genes.csv" specifying gene names to be replaced and new names (under $old_names and $new_names fields, respectively).

1. Resolves "self-overlapping" gene (duplicate gene_ids) derived issues. Required for making references compatible with multiome workflows.

2. Creates pre-mRNA genome annotation from input genome annotation. This step extracts all transcript entries from the genome annotation and defines them as full-length exons with new transcript IDs and corresponding transcripts. This allows to capture many intronically mapped reads that otherwise get discarded.

3. Gene deletion step: Deletes all annotation entries for genes destined for deletion (has "Delete" entry in $final_classification field of "overlapping_gene_list.csv".

4. Transcript deletion step: Deletes all transcripts destined for deletion (transcript names listed in the "transcripts_for_deletion" column in "overlapping_gene_list.csv".

5. Gene coordinate adjustment step: Replaces the left most or right most coordinate of the first exon of a gene in genome annotation if there is a coordinate in columns $new_left or $new_right in the "gene_extension_candidates.csv".

6. Adds pre-mRNA reads to all genes not in the gene overlap list.

7. Renames genes to avoid discarding expression data with near perfect terminal exon overlap.

8. Saves the optimized genome annotation in a new GTF file.

Usage

```
OptimizedAnnotationAssembler(
  unoptimized_gtf,
  gene_overlaps,
  gene_extension,
  gene_replacement
)
```

Arguments

`unoptimized_annotation_path`	Path to unoptimized genome annotation file in GTF.
`gene_overlaps`	Overlapping genes list generated with *IdentifyOverlappers* function.
`gene_extension`	List of gene extension candidates generated with *GenerateExtensionCandidates* function.
`gene_replacement`	Optional. Manually generated list of gene names to be replaced in .csv format. Column names: old_name, new_name.

Value

Single-cell RNA-seq optimized genome annotation that can be used to generate the transcriptomic reference (e.g. with cellranger mkref or STAR --runMode genomeGenerate pipelines) for mapping single-cell sequencing data.

Examples

```
OptimizedAnnotationAssembler(unoptimized_annotation_path = "test_genes.gtf",
gene_overlaps = "test_overlapping_gene_list.csv", gene_extension =
"./gene_extension_candidates.csv", gene_replacement =
"test_gene_replacement.csv")
```