PRAM: Pooling RNA-seq and Assembling Models

Peng Liu, Colin N. Dewey, and Sündüz Keleş

2019-01-16

Contents

1 Introduction

Pooling RNA-seq and Assembling Models (**PRAM**) is an **R** package that utilizes multiple RNA-seq datasets to predict transcript models. The workflow of PRAM contains four steps. Figure 1 shows each step with function name and associated key parameters. In addition, we provide a function named `evalModel()` to evaluate prediction accuracy by comparing transcript models with true transcripts. In the later sections of this vignette, we will describe each function in details.

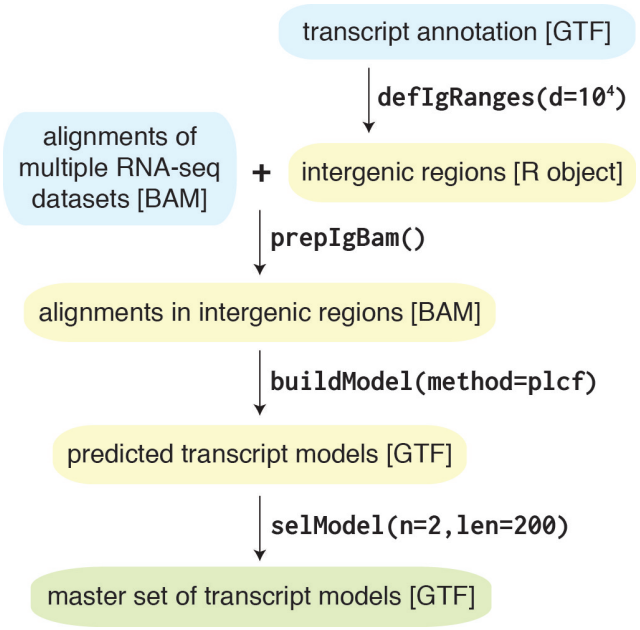

Figure 1: PRAM workflow

2 Installation

2.1 From GitHub

Use the following **R** command:

```
devtools::install_github('pliu55/pram')
```

2.2 From Bioconductor

To install this package, start R (version "3.6") and enter:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("GenomicRanges", version = "3.9")
```

For older versions of R, please refer to the appropriate Bioconductor release.

3 Quick start

3.1 Description

PRAM provides a function named `runPRAM()` to let you conveniently run through the whole workflow.

For a given gene annotation and RNA-seq alignments, you can predict transcript models in intergenic genomic regions:

```
##
## assuming the stringtie binary is in folder /usr/local/stringtie-1.3.3/
##
pram::runPRAM(  in_gtf, in_bamv, out_gtf, method='plst',
                stringtie='/usr/loca/stringtie-1.3.3/stringtie')
```

- `in_gtf`: an input GTF file defining genomic coordinates of existing genes. Required to have an attribute of **gene_id** in the ninth column.
- `in_bamv`: a vector of input BAM file(s) containing RNA-seq alignments. Currently, PRAM only supports strand-specific paired-end RNA-seq with the first mate on the right-most of transcript coordinate, i.e., 'fr-firststrand' by Cufflinks definition.
- `out_gtf`: an output GTF file of predicted transcript models.
- `method`: prediction method. For the command above, we were using pooling RNA-seq datasets and building models by **stringtie**. For a list of available PRAM methods, please check Table 2 below.
- `stringtie`: location of the **stringtie** binary. PRAM's model-building step depends on external software. For more information on this topic, please see Section 6.3 below.

3.2 Examples

PRAM has included input examples files in its `extdata/demo/` folder. The table below provides a quick summary of all the example files.

Table 1: `runPRAM()`'s input example files

input argument	file name(s)
`in_gtf`	in.gtf
`in_bamv`	SZP.bam, TLC.bam

You can access example files by `system.file()` in **R**, e.g. for the argument `in_gtf`, you can access its example file by

```
system.file('extdata/demo/in.gtf', package='pram')
#> [1] "/private/var/folders/t8/_sg3zwmj0rjb80s3swpktc_r0000gr/T/RtmpXYtubW/temp_libpathebf126e5c0b4/pram/ext
```

Below shows the usage of `runPRAM()` with example input files:

```
in_gtf = system.file('extdata/demo/in.gtf', package='pram')

in_bamv = c(system.file('extdata/demo/SZP.bam', package='pram'),
            system.file('extdata/demo/TLC.bam', package='pram') )
```

```
pred_out_gtf = tempfile(fileext='.gtf')

##
## assuming the stringtie binary is in folder /usr/local/stringtie-1.3.3/
##
pram::runPRAM(  in_gtf, in_bamv, pred_out_gtf, method='plst',
                stringtie='/usr/loca/stringtie-1.3.3/stringtie')
```

4 Define intergenic genomic ranges: `defIgRanges()`

4.1 Description

To predict intergenic transcripts, we must first define intergenic regions by `defIgRanges()`. This function requires a GTF file containing known gene annotation supplied for its `in_gtf` argument. This GTF file should contain an attribue of **gene_id** in its ninth column. We provided an example input GTF file in PRAM package: `extdata/gtf/defIGRanges_in.gtf`.

In addition to gene annotation, `defIgRanges()` also requires user to provide chromosome sizes so that it would know the maximum genomic ranges. You can provide one of the following arguments:

- `chromgrs`: a GRanges object, or
- `genome`: a genome name, currently supported ones are: **hg19**, **hg38**, **mm9**, and **mm10**, or
- `fchromsize`: a UCSC genome browser-style size file, e.g. hg19

By default, `defIgRanges()` will define intergenic ranges as regions 10 kb away from any known genes. You can change it by the `radius` argument.

4.2 Example

```
pram::defIgRanges(system.file('extdata/gtf/defIgRanges_in.gtf', package='pram'),
                  genome = 'hg38')
#> GRanges object with 456 ranges and 0 metadata columns:
#>                  seqnames          ranges strand
#>                     <Rle>       <IRanges>  <Rle>
#>     [1]             chr1 30001-248956422      *
#>     [2]             chr2     1-242193529      *
#>     [3]             chr3     1-198295559      *
#>     [4]             chr4     1-190214555      *
#>     [5]             chr5     1-181538259      *
#>     ...              ...             ...    ...
#>   [452] chrUn_KI270753v1         1-62944      *
#>   [453] chrUn_KI270754v1         1-40191      *
#>   [454] chrUn_KI270755v1         1-36723      *
#>   [455] chrUn_KI270756v1         1-79590      *
#>   [456] chrUn_KI270757v1         1-71251      *
#>   -------
#>   seqinfo: 455 sequences from an unspecified genome; no seqlengths
```

5 Prepare input RNA-seq alignments: `prepIgBam()`

5.1 Description

Once intergenic regions were defined, `prepIgBam()` will extract corresponding RNA-seq alignments from input BAM files. In this way, transcript models predicted at later stage will solely from intergenic regions. Also, with fewer RNA-seq alignments, model prediction will run faster.

Three input arguments are required by `prepIgBam()`:

- `finbam`: an input RNA-seq BAM file sorted by genomic coordinate. Currently, we only support strand-specific paired-end RNA-seq data with the first mate on the right-most of transcript coordinate, i.e. 'fr-firststrand' by Cufflinks's definition.
- `iggrs`: a GRanges object to define intergenic regions.
- `foutbam`: an output BAM file.

5.2 Example

```
finbam =system.file('extdata/bam/CMPRep2.sortedByCoord.raw.bam',
                    package='pram')

iggrs = GenomicRanges::GRanges('chr10:77236000-77247000:+')

foutbam = tempfile(fileext='.bam')

pram::prepIgBam(finbam, iggrs, foutbam)
#> RNA-seq alignments are saved in the following file:
#>  /var/folders/t8/_sg3zwmj0rjb80s3swpktc_r0000gr/T//RtmpXYtubW/fileebf148ea044c.bam
```

6 Build transcript models: `buildModel()`

6.1 Description

`buildModel()` predict transcript models from RNA-seq BAM file(s). This function requires two arguments:

- `in_bamv`: a vector of input BAM file(s)
- `out_gtf`: an output GTF file containing predicted transcript models

6.2 Transcript prediction methods

`buildModel()` has implemented seven transcript prediction methods. You can specify it by the `method` argument with one of the keywords: **plcf**, **plst**, **cfmg**, **cftc**, **stmg**, **cf**, and **st**. The first five denote meta-assembly methods that utilize multiple RNA-seq datasets to predict a single set of transcript models. The last two represent methods that predict transcript models from a single RNA-seq dataset.

The table below compares prediction steps for these seven methods. By default, `buildModel()` uses **plcf** to predict transcript models.

Table 2: Prediction steps of the seven `buildModel()` methods

method	meta-assembly	preparing RNA-seq input	building transcripts	assembling transcripts
plcf	yes	pooling alignments	Cufflinks	no
plst	yes	pooling alignments	StringTie	no
cfmg	yes	no	Cufflinks	Cuffmerge
cftc	yes	no	Cufflinks	TACO
stmg	yes	no	StringTie	StringTie-merge
cf	no	no	Cufflinks	no
st	no	no	StringTie	no

6.3 Required external software

Depending on your specified prediction method, `buildModel()` requires external software: Cufflinks, StringTie and/or TACO, to build and/or assemble transcript models. You can either specify the software location using the `cufflinks`, `stringtie`, and `taco` arguments in `buildModel()`, or simply leave these three arugments undefined and let PRAM download them for you automatically. The table below summarized software versions `buildModel()` would download when required software was not specified. Please note that, for **macOS**, pre-compiled Cufflinks binary versions 2.2.1 and 2.2.0 appear to have an issue on processing BAM files, therefore we recommend to use version 2.1.1 instead.

Table 3: `buildModel()`-required software and recommended version

software	Linux binary	macOS binary	required by
Cufflinks, Cuffmerge	v2.2.1	v2.1.1	**plcf**, **cfmg**, **cftc**, and **cf**
StringTie, StringTie-merge	v1.3.3b	v1.3.3b	**plst**, **stmg**, and **st**
TACO	v0.7.0	v0.7.0	**cftc**

6.4 Example

```
fbams = c(  system.file('extdata/bam/CMPRep1.sortedByCoord.clean.bam',
                        package='pram'),
            system.file('extdata/bam/CMPRep2.sortedByCoord.clean.bam',
                        package='pram') )

foutgtf = tempfile(fileext='.gtf')

##
## assuming the stringtie binary is in folder /usr/local/stringtie-1.3.3/
##
pram::buildModel(fbams, foutgtf, method='plst',
                 stringtie='/usr/loca/stringtie-1.3.3/stringtie')
```

7 Select transcript models: `selModel()`

7.1 Description

Once transcript models were built, you may want to select a subset of them by their genomic features. `selModel()` was developed for this purpose. It allows you to select transcript models by their total number of exons and total length of exons and introns.

`selModel()` requires two arguments:

- `fin_gtf`:input GTF file containing to-be-selected transcript models. This file is required to have **transcript_id** attribute in the ninth column.
- `fout_gtf`: output GTF file containing selected transcript models.

By default: `selModel()` will select transcript models with ≥ 2 exons and ≥ 200 bp total length of exons and introns. You can change the default using the `min_n_exon` and `min_tr_len` arguments.

7.2 Example

```
fin_gtf = system.file('extdata/gtf/selModel_in.gtf', package='pram')

fout_gtf = tempfile(fileext='.gtf')

pram::selModel(fin_gtf, fout_gtf)
#> Transcript models are saved in the following file:
#>  /var/folders/t8/_sg3zwmj0rjb80s3swpktc_r0000gr/T//RtmpXYtubW/fileebf151c57042.gtf
```

8 Evaluate transcript models: `evalModel()`

8.1 Motivation

After PRAM has predicted a number of transcript models, you may wonder how accurate these models are. To answer this question, you can compare PRAM models with real transcripts (i.e., positive controls) that you know should be predicted. PRAM's `evalModel()` function will help you to make such comparison. It will calculate precision and recall rates on three features of a transcript: exon nucleotides, individual splice junctions, and transcript structure (i.e., whether all splice junctions within a transcript were constructed in a model).

8.2 Input

`evalModel()` requires two arguments:

- `model_exons`: genomic coordinates of transcript model exons.
- `target_exons`: genomic coordinates of real transcript exons.

The two arguments can be in multiple formats:

- both are `GRanges` objects
- both are `character` objects denoting names of GTF files
- both are `data.table` objects containing the following five columns for each exon:

- **chrom**: chromosome name
- **start**: starting coordinate
- **end**: ending coordinate
- **strand**: strand information, either '+' or '-'
- **trid**: transcript ID

- `model_exons` is the name of a GTF file and `target_exons` is a `data.table` object.

8.3 Output

The output of `evalModel()` is a `data.table` object, where columns are evaluation results and each row is three transcript features.

Table 4: `evalModel()` output columns

column name	representation
feat	transcript feature
ntp	number of true positives (TP)
nfn	number of false negatives (FN)
nfp	number of false positives (FP)
precision	precision rate: $\frac{TP}{(TP+FP)}$
recall	recall rate: $\frac{TP}{(TP+FN)}$

Table 5: `evalModel()` output rows

feature name	representation
exon_nuc	exon nucleotide
indi_jnc	individual splice junction
tr_jnc	transcript structure

8.4 Example

```
fmdl = system.file('extdata/benchmark/plcf.tsv.gz', package='pram')
ftgt = system.file('extdata/benchmark/tgt.tsv.gz',  package='pram')

mdldt = data.table::data.table(read.table(fmdl, header=TRUE, sep="\t"))
tgtdt = data.table::data.table(read.table(ftgt, header=TRUE, sep="\t"))

pram::evalModel(mdldt, tgtdt)
#>        feat     ntp    nfn  nfp precision    recall
#> 1: exon_nuc 1723581 109337 8424 0.9951363 0.9403481
#> 2: indi_jnc    2889    162  192 0.9376826 0.9469027
#> 3:   tr_jnc    1138    118  252 0.8187050 0.9060510
```

9 Session Info

Below is the output of `sessionInfo()` on the system on which this document was compiled.

```
#> R version 3.5.1 (2018-07-02)
#> Platform: x86_64-apple-darwin15.6.0 (64-bit)
#> Running under: macOS  10.14.2
#>
#> Matrix products: default
#> BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
#>
#> locale:
#> [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#> [1] BiocStyle_2.10.0
#>
#> loaded via a namespace (and not attached):
#>  [1] SummarizedExperiment_1.10.1 xfun_0.3
#>  [3] remotes_2.0.2               lattice_0.20-35
#>  [5] testthat_2.0.1              usethis_1.4.0
#>  [7] htmltools_0.3.6             stats4_3.5.1
#>  [9] rtracklayer_1.42.1          yaml_2.2.0
#> [11] base64enc_0.1-3             XML_3.98-1.11
#> [13] rlang_0.3.0.1               pkgbuild_1.0.2
#> [15] glue_1.3.0                  withr_2.1.2
#> [17] BiocParallel_1.14.1         BiocGenerics_0.28.0
#> [19] sessioninfo_1.1.1           matrixStats_0.53.1
#> [21] GenomeInfoDbData_1.1.0      stringr_1.3.1
#> [23] zlibbioc_1.26.0             Biostrings_2.48.0
#> [25] devtools_2.0.1              evaluate_0.11
#> [27] memoise_1.1.0               pram_0.99.3
#> [29] Biobase_2.40.0              knitr_1.20
#> [31] callr_3.0.0                 IRanges_2.16.0
#> [33] ps_1.1.0                    GenomeInfoDb_1.18.1
#> [35] parallel_3.5.1              Rcpp_0.12.18
#> [37] backports_1.1.2             BiocManager_1.30.3
#> [39] DelayedArray_0.6.0          desc_1.2.0
#> [41] S4Vectors_0.20.1            pkgload_1.0.2
#> [43] XVector_0.22.0              fs_1.2.6
#> [45] Rsamtools_1.34.0            digest_0.6.18
#> [47] stringi_1.2.2               bookdown_0.7
#> [49] processx_3.2.0              grid_3.5.1
#> [51] GenomicRanges_1.34.0        rprojroot_1.3-2
#> [53] cli_1.0.1                   tools_3.5.1
#> [55] bitops_1.0-6                magrittr_1.5
#> [57] RCurl_1.95-4.10             crayon_1.3.4
#> [59] Matrix_1.2-14               data.table_1.11.8
#> [61] prettyunits_1.0.2           assertthat_0.2.0
#> [63] rmarkdown_1.10              R6_2.3.0
#> [65] GenomicAlignments_1.18.0    compiler_3.5.1
```