gmapR: Use the GMAP Suite of Tools in R

Michael Lawrence, Cory Barr

April 7, 2020

Contents

1 Introduction

The *gmapR* packages provides users with a way to access GSNAP, bam_tally, and other utilities from the GMAP suite of tools within an R session. In this vignette, we briefly look at the GMAP suite of tools available through the *gmapR* package and work through an example.

2 What is GMAP, GSNAP, and bam_tally?

The GMAP suite offers useful tools for the following:

- Genomic mapping: Given a cDNA, find where it best aligns to an entire genome
- Genomic alignment: Given a cDNA and a genomic segment, provide a nucleotide-level correspondence for the exons of the cDNA to the genomic segment
- Summarization via coverage plus reference and allele nucleotide polymorphism counts for an aligned set of sequencing reads over a given genomic location

GMAP (Genomic Mapping and Alignment Program) is particularly suited to relatively long mRNA and EST sequences such as those that are obtained from Roche 454 or Pacific Biosciences sequencing technologies. (At present, only GSNAP is available through the *gmapR*. GMAP integration is scheduled for the near future.)

GSNAP (Genomic Short-read Nucleotide Alignment Program) also provides users with genomic mapping and alignment capabilities, but is optimized to handle issues that arise when dealing with the alignment of short reads generated from sequencing technologies such as those from Illumina/Solexa or ABI/SOLiD. GSNAP offers the following functionality, as mentioned in Fast and SNP-tolerant detection of complex variants and splicing in short reads by Thomas D. Wu and Serban Nacu:

- fast detection of complex variants and splicing in short reads, based on a successively constrained search process of merging and filtering position lists from a genomic index
- alignment of both single- and paired-end reads as short as 14 nt and of arbitrarily long length
- detection of short- and long-distance splicing, including interchromosomal splicing, in individual reads, using probabilistic models or a database of known splice sites
- SNP-tolerant alignment to a reference space of all possible combinations of major and minor alleles
- alignment of reads from bisulfite-treated DNA for the study of methylation state

bam_tally provides users with the coverage as well as counts of both reference alleles, alternative alleles, and indels over a genomic region.

For more detailed information on the GMAP suite of tools including a detailed explication of algorithmic specifics, see `http://research-pub.gene.com/gmap/`.

3 Create a *GmapGenome* Object

To align reads with GMAP or GSNAP, or to use bam_tally, you will need to either obtain or create a *GmapGenome* object. *GmapGenome* objects can be created from FASTA files or *BSgenome* objects, as the following example demonstrates:

```
> library(gmapR)
> if (!suppressWarnings(require(BSgenome.Dmelanogaster.UCSC.dm3))) {
+   if (!requireNamespace("BiocManager", quietly=TRUE))
```

```
+       install.packages("BiocManager")
+   BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm3")
+   library(BSgenome.Dmelanogaster.UCSC.dm3)
+ }
> gmapGenomePath <- file.path(getwd(), "flyGenome")
> gmapGenomeDirectory <- GmapGenomeDirectory(gmapGenomePath, create = TRUE)
> ##> gmapGenomeDirectory
> ##GmapGenomeDirectory object
> ##path: /reshpcfs/home/coryba/projects/gmapR2/testGenome
>
> gmapGenome <- GmapGenome(genome=Dmelanogaster,
+                          directory=gmapGenomeDirectory,
+                          name="dm3",
+                          create=TRUE,
+                          k = 12L)
> ##> gmapGenome
> ##GmapGenome object
> ##genome: dm3
> ##directory: /reshpcfs/home/coryba/projects/gmapR2/testGenome
```

4 Aligning with GSNAP

The GSNAP algorithm incorporates biological knowledge to provide accurate alignments, particularly for RNA-seq data. In this section, we will align reads from an RNA-seq experiment provided in a fastq file to a selected region of the human genome. In this example, we will align to the region of the human genome containing TP53 plus an additional one megabase on each side of this gene.

First we need to obtain the desired region of interest from the genome:

```
> library("org.Hs.eg.db")
> library("TxDb.Hsapiens.UCSC.hg19.knownGene")
> eg <- org.Hs.eg.db::org.Hs.egSYMBOL2EG[["TP53"]]
> txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
> tx <- transcripts(txdb, filter = list(gene_id = eg))
> roi <- range(tx) + 1e6
> strand(roi) <- "*"
```

Next we get the genetic sequence and use it to create a GmapGenome object. (Please note that the TP53_demo GmapGenome object is used by many examples and tests in the gmapR and VariationTools packages. If the object has been created before, its subsequent creation will be instantaneous.)

```
> library("BSgenome.Hsapiens.UCSC.hg19")
> library("gmapR")
> p53Seq <- getSeq(BSgenome.Hsapiens.UCSC.hg19::Hsapiens, roi,
+                  as.character = FALSE)
> names(p53Seq) <- "TP53"
> gmapGenome <- GmapGenome(genome = p53Seq,
+                          name = paste0("TP53_demo_",
+                            packageVersion("TxDb.Hsapiens.UCSC.hg19.knownGene")),
+                          create = TRUE,
+                          k = 12L)
```

We add the known transcripts (splice sites) to the genome index:

```
> exons <- gmapR:::subsetRegion(exonsBy(txdb), roi, "TP53")
> spliceSites(gmapGenome, "knownGene") <- exons
```

The data package LungCancerLines contains fastqs of reads obtained from sequencing H1993 and H2073 cell lines. We will use these fastqs to demonstrate GSNAP alignment with gmapR.

```
> library("LungCancerLines")
> fastqs <- LungCancerFastqFiles()
```

GSNAP is highly configurable. Users create *GsnapParam* objects to specify desired GSNAP behavior.

```
> ##specify how GSNAP should behave using a GsnapParam object
> gsnapParam <- GsnapParam(genome=gmapGenome,
+                          unique_only=FALSE,
+                          suboptimal_levels=2L,
+                          npaths=10L,
+                          novelsplicing=TRUE,
+                          splicing="knownGene",
+                          indel_penalty=1L,
+                          distant_splice_penalty=1L,
+                          clip_overlap=TRUE)
```

Now we are ready to align.

```
> gsnapOutput <- gsnap(input_a=fastqs["H1993.first"],
+                      input_b=fastqs["H1993.last"],
+                      params=gsnapParam)
>
> ##gsnapOutput
> ##An object of class "GsnapOutput"
> ##Slot "path":
> ##[1] "/local/Rtmporwsvr"
> ##
> ##Slot "param":
> ##A GsnapParams object
> ##genome: dm3 (/reshpcfs/home/coryba/projects/gmapR2/testGenome)
> ##part: NULL
> ##batch: 2
> ##max_mismatches: NULL
> ##suboptimal_levels: 0
> ##snps: NULL
> ##mode: standard
> ##nthreads: 1
> ##novelsplicing: FALSE
> ##splicing: NULL
> ##npaths: 10
> ##quiet_if_excessive: FALSE
> ##nofails: FALSE
> ##split_output: TRUE
> ##extra: list()
> ##
> ##Slot "version":
> ## [1] NA NA NA NA NA NA NA NA NA NA NA NA
> ##
```

The gsnapOutput object will be of the class *GsnapOutput*. It will provide access to the BAM files of alignments that GSNAP outputs along with other utilities.

```
> ##> dir(path(gsnapOutput), full.names=TRUE, pattern="\\.bam$")
> ##[1] "/local/Rtmporwsvr/file1cbc73503e9.1.nomapping.bam"
> ##[2] "/local/Rtmporwsvr/file1cbc73503e9.1.unpaired_mult.bam"
> ##[3] "/local/Rtmporwsvr/file1cbc73503e9.1.unpaired_transloc.bam"
> ##[4] "/local/Rtmporwsvr/file1cbc73503e9.1.unpaired_uniq.bam"
```

5 Using bam_tally

Running the bam_tally method will return a *GRanges* of information per nucleotide. Below is an example demonstrating how to find variants in the TP53 gene of the human genome. See the documentation for the bam_tally method for more details.

gmapR provides access to a demo genome for examples. This genome encompasses the TP53 gene along with a 1-megabase flanking region on each side.

```
> genome <- TP53Genome()
```

The LungCancerLines R package contains a BAM file of reads aligned to the TP53 region of the human genome. We'll use this file to demonstrate the use of bam_tally through the gmapR package. The resulting data structure will contain the needed information such as number of alternative alleles, quality scores, and read position for each allele.

The call to bam_tally returns an opaque pointer to a C-level data structure. We anticipate having multiple means of summarizing these data into R data structures. For now, there is one: variantSummary, which returns a VRanges object describing putative genetic variants in the sample.

```
> bam_file <- system.file("extdata/H1993.analyzed.bam",
+                         package="LungCancerLines", mustWork=TRUE)
> breaks <- c(0L, 15L, 60L, 75L)
> bqual <- 56L
> mapq <- 13L
> param <- BamTallyParam(genome,
+                        minimum_mapq = mapq,
+                        concordant_only = FALSE, unique_only = FALSE,
+                        primary_only = FALSE,
+                        min_depth = 0L, variant_strand = 1L,
+                        ignore_query_Ns = TRUE,
+                        indels = FALSE, include_soft_clips = 1L, xs=TRUE,
+                        min_base_quality = 23L)
> tallies <-bam_tally(bam_file,
+                     param)
> variantSummary(tallies)
```

6 Creating a *GmapGenome* Package

After creating a *GmapGenome* object, you might want to distribute it for collaboration or version it for reproducible research. The function makeGmapGenomePackage allows you to do this. Continuing on with the D. melanogaster example from above, here is how to archive the D. melanogaster *GmapGenome* object in a *GmapGenome* package:

```
> makeGmapGenomePackage(gmapGenome=gmapGenome,
+                       version="1.0.0",
+                       maintainer="<your.name@somewhere.com>",
+                       author="Your Name",
+                       destDir="myDestDir",
+                       license="Artistic-2.0",
+                       pkgName="GmapGenome.Dmelanogaster.UCSC.dm3")
```

After creating the package, you can run `R CMD INSTALL myDestDir` to install it, or run `R CMD build myDestDir` to create a distribution of the package.

Many users with be working with the human genome. Many of the examples used in gmapR make use of a particular build of the human genome. As such, creating a *GmapGenome* of hg19 is recommended. Here is one way to create it, using a *BSgenome* object:

```
> if (!suppressWarnings(require(BSgenome.Hsapiens.UCSC.hg19))) {
+   if (!requireNamespace("BiocManager", quietly=TRUE))
+       install.packages("BiocManager")
+   BiocManager::install("BSgenome.Hsapiens.UCSC.hg19")
+   library(BSgenome.Hsapiens.UCSC.hg19)
+ }
> gmapGenome <- GmapGenome(genome=Hsapiens,
+                          directory = "Hsapiens",
+                          name = "hg19",
+                          create = TRUE)
> destDir <- "HsapiensGmapGenome"
> pkgName <- "GmapGenome.Hsapiens.UCSC.hg19"
> makeGmapGenomePackage(gmapGenome=gmapGenome,
+                       version="1.0.0",
+                       maintainer="<your.name@somewhere.com>",
+                       author="Your Name",
+                       destDir=destDir,
+                       license="Artistic-2.0",
+                       pkgName="GmapGenome.Hsapiens.UCSC.hg19")
```

After running the above code, you should be able to run `R CMD INSTALL GmapGenome.Hsapiens.UCSC.hg19` in the appropriate directory from the command line, submit GmapGenome.Hsapiens.UCSC.hg19 to a repository, etc. After installation, `library("GmapGenome.Hsapiens.UCSC.hg19")` will load a *GmapGenome* object that has the same name as the package.

```
> sessionInfo()

R version 4.0.0 alpha (2020-04-05 r78150)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Mojave 10.14.6

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
```

```
attached base packages:
[1] parallel  stats4    stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] LungCancerLines_0.25.0
 [2] gmapR_1.29.0
 [3] Rsamtools_2.3.7
 [4] BSgenome.Hsapiens.UCSC.hg19_1.4.3
 [5] BSgenome_1.55.4
 [6] rtracklayer_1.47.0
 [7] Biostrings_2.55.7
 [8] XVector_0.27.2
 [9] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
[10] GenomicFeatures_1.39.7
[11] GenomicRanges_1.39.3
[12] GenomeInfoDb_1.23.16
[13] org.Hs.eg.db_3.10.0
[14] AnnotationDbi_1.49.1
[15] IRanges_2.21.8
[16] S4Vectors_0.25.15
[17] Biobase_2.47.3
[18] BiocGenerics_0.33.3

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.4.6                lattice_0.20-41
 [3] prettyunits_1.1.1           assertthat_0.2.1
 [5] digest_0.6.25               BiocFileCache_1.11.4
 [7] R6_2.4.1                    RSQLite_2.2.0
 [9] httr_1.4.1                  pillar_1.4.3
[11] zlibbioc_1.33.1             rlang_0.4.5
[13] progress_1.2.2              curl_4.3
[15] blob_1.2.1                  Matrix_1.2-18
[17] BiocParallel_1.21.2         stringr_1.4.0
[19] RCurl_1.98-1.1              bit_1.1-15.2
[21] biomaRt_2.43.5              DelayedArray_0.13.10
[23] compiler_4.0.0              pkgconfig_2.0.3
[25] askpass_1.1                 openssl_1.4.1
[27] tidyselect_1.0.0            SummarizedExperiment_1.17.5
[29] tibble_3.0.0                GenomeInfoDbData_1.2.2
[31] matrixStats_0.56.0          XML_3.99-0.3
[33] fansi_0.4.1                 crayon_1.3.4
[35] dplyr_0.8.5                 dbplyr_1.4.2
[37] GenomicAlignments_1.23.2    bitops_1.0-6
[39] rappdirs_0.3.1              grid_4.0.0
[41] lifecycle_0.2.0             DBI_1.1.0
[43] magrittr_1.5                cli_2.0.2
[45] stringi_1.4.6               ellipsis_0.3.0
[47] vctrs_0.2.4                 tools_4.0.0
[49] bit64_0.9-7                 glue_1.4.0
[51] purrr_0.3.3                 hms_0.5.3
```

[53] memoise_1.1.0 VariantAnnotation_1.33.3