seq2pathway Vignette

Bin Wang, Xinan (Holly) Yang

October 17, 2016

Contents

1 Abstract 1
2 Package Installation 2
3 runseq2pathway 2
4 Two main functions 3
4.1 seq2gene 3
4.1.1 seq2gene flowchart 3
4.1.2 runseq2gene inputs/parameters 5
4.1.3 runseq2gene outputs 8
4.2 gene2pathway 10
4.2.1 gene2pathway flowchart 11
4.2.2 gene2pathway_test inputs/parameters 11
4.2.3 gene2pathway_test outputs 12
5 Examples 13
5.1 ChIP-seq data analysis 13
5.1.1 Map ChIP-seq enriched peaks to genes using runseq2gene 13
5.1.2 Discover enriched GO terms using gene2pathway_test with gene scores 15
5.1.3 Discover enriched GO terms using Fisher’s Exact test without gene scores 17
5.1.4 Add description for genes 20
5.2 RNA-seq data analysis 20
6 R environment session 23

Abstract

Seq2pathway is a novel computational tool to analyze functional gene-sets (including signaling pathways) using variable next-generation sequencing data\textsuperscript{[1]}. Integral to this tool are the “seq2gene” and “gene2pathway” components in series that infer a quantitative pathway-level profile for each sample. The seq2gene function assigns phenotype-associated significance of genomic regions to gene-level scores, where the significance could be p-values of SNPs or point mutations, protein-binding affinity, or transcriptional expression level. The seq2gene function has the feasibility to assign non-exon regions to a range of neighboring genes besides the nearest one, thus facilitating the study of functional non-coding elements\textsuperscript{[2]}. Then the gene2pathway summarizes gene-level measurements to pathway-level scores, comparing the quantity of significance for gene members within a pathway with those outside a pathway. It implements an improved FAIME algorithm together with other three conventional gene-set enrichment analysis methods\textsuperscript{[3]}. The output of seq2pathway is a general structured pathway scores, thus allowing one to functionally interpret phenotype-associated significance of genomic regions derived by next generational sequencing experiments.
Package Installation

Currently, seq2pathway works in both Linux and Windows. It has wrapped python scripts to annotate loci to genes, thus requires Python v2.7 running on the system. For Windows, the Python should be installed at "C:/python27/python" (default). Make sure supporting data package seq2pathway.data is installed with seq2pathway package.

If you don't have biocLite() you can get it like this:

source("http://bioconductor.org/biocLite.R")
biocLite("seq2pathway.data")
biocLite("seq2pathway")

Now install seq2pathway and the data package:

> library("seq2pathway.data")
> library("seq2pathway")

runseq2pathway

This function provides end-users a straightforward work-flow to implement the seq2pathway algorithms. It facilitates the screening of novel biological functions using just a few code lines, the main function to derive enriched pathways from genomic regions. It uses the Gene Ontology (GO)-defined gene-sets by default and can be run against either the MSigDB-defined[4] or customized gene-sets.

> head(runseq2pathway, n=8)

The inputs are almost the same as those introduced below for the the two main functions runseq2gene and gene2pathway_test. We therefore only introduce the new parameters here.

The inputs are almost the same as those introduced below for the the two main functions runseq2gene and gene2pathway_test. We therefore only introduce the new parameters here.

Note that the wrapped function runseq2pathway supports the “FAIME” method only and performs empirical test if the new parameter FAMETest equals to “TRUE”.

If setting FAMETest=TRUE and/or calculating the empirical p-values, an end-user should provide the formatted input file (see following example).

Column 1 the unique IDs (labels) of genomic regions of interest
Column 2 the chromosome IDs (eg. chr5 or 5)
Column 3 the start of genomic regions of interest
Column 4 the end of genomic regions (for SNP and point mutations, the difference of start and end is 1bp)
Column 5 the scores or values of the sample(s) along with the genomic regions
Column … other custom-defined information

Another new parameter collapsemethod is a character for determining which method to use when call the function collapseRows in package WGCNA[5].

These are the options provided by WGCNA for the parameter collapsemethod (directly from WGCNA Vignette):

“MaxMean” (default) or “MinMean” = choose the row with the highest or lowest mean value, respectively
“maxRowVariance” = choose the row with the highest variance (across the columns of data)

“absMaxMean” or “absMinMean” = choose the row with the highest or lowest mean absolute value

“ME” = choose the eigenrow (first principal component of the rows in each group)

“Average” for each column, take the average value of the rows in each group

“function” use this method for a user-input function (see the description of the argument "methodFunction")

Two main functions

The output of runseq2pathway can be achieved equally by running runseq2gene and gene2pathway_test functions in series. These two functions facilitate end-users to track details on the gene-level. End-users can also apply the gene2pathway_test function to analyze functional enrichment for customized gene lists independently.

Here we introduce these two main functions separately. For each function, we describe the significance, its features with a flowchart, the inputs and parameters, then the output in details.

“runseq2gene” The first components in series to map genomic regions to coding and non-coding genes[2].

“gene2pathway_test” The second components in series to run pathway enrichment analysis for coding genes. This function provides three alternative pathway estimating methods which are FAIME[3], Kolmogorov-Smirnov test[6], and cumulative rank test[6].

seq2gene

Nearly 99% of human genome are non-coding nucleotides[7]. Identifying and delineating the function of all coding genes and non-coding elements remains a considerable challenge. We developed the computational function runseq2gene to link genomic regions of interest to genes in a many-to-many mapping, by considering the possibility that genes within a search radius in both directions from intergenic regions may fall under control of cis-regulation[2]. Using the seq2gene strategy with a search radius of 100k-base, our recent study in vivo defined a transcription factor-mediated cis-regulatory element from both ChIP-seq and transcriptomic data[8]. We also identified an intronic locus of one gene regulates the transcript of its neighbor gene instead of its host gene, suggesting the need to associate a functional genomic locus to broader candidate targets[9]. We thus suggest a larger search radius for the seq2gene function, such as 100k -150k bases, given that the average enhancer-promoter loop size is 120 kb in mammalian genomes[10] and enhancers act independently of their orientation[11][12].

seq2gene flowchart

Figure 1: Seq2gene flowchart. The inputs are on the left, and the outputs are on the right.
Figure 1 gives the flowchart for the seq2gene process. Built on our previous publication[2], the current seq2gene uses the reference human genome annotation for the ENCODE project (GENCODE) [13] version 19 for human genome and version M4 for mouse genome (Ensembl version 78 in GRCm38). ENCODE is a re-merge between the Ensembl annotation and updates from HAVANA(http://www.gencodegenes.org/releases/). Table 1 lists the statistics of the gene annotations that are used by seq2pathway.

<table>
<thead>
<tr>
<th>Species</th>
<th>GENCODE Release</th>
<th>Corresponding Ensembl assembly</th>
<th># of coding genes</th>
<th># of Long non-coding RNAs</th>
<th># of Small non-coding RNAs</th>
<th># of Pseudogenes</th>
<th># of all genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>19(Dec.2013)</td>
<td>GRCh74/hg19</td>
<td>20345</td>
<td>13870</td>
<td>9013</td>
<td>14206</td>
<td>57820</td>
</tr>
<tr>
<td>Mouse</td>
<td>M4(Aug.2014)</td>
<td>GRCm38.p3/mm10</td>
<td>22032</td>
<td>6951</td>
<td>5853</td>
<td>7957</td>
<td>43346</td>
</tr>
</tbody>
</table>

The seq2gene algorithm uses a bisection strategy to search among exon and transcript annotations. Figure 2 is the pseudocode for the function[2]. To perform the basic bisect algorithm with respect to exon and transcript separately, we have prepared for end users the internal “exon.table” and “transcript.table” files based on the GENCODE general feature format. Both file use ENSEMBL IDs as the key index.
Algorithm: seq2gene
Input: peaks, exontable, transcriptable, search radius
Output: peak with annotated gene information

1. for i = 1 to length(peaks) do begin:
2. m = peakleft
3. n = peakright
4. middle = (m+n)/2
5. locate the nearest exon(J) for peak(i) by the basic bisect algorithm
6. if peak(i) resides inside exon(J)
7. report peak(i) with exon(J)
8. endif
9. while exon(x) intersecting with peak(i)
10. report peak(i) with exon(x)
11. exon(x) = the closest exons (left or right)
12. endwhile
13. locate the nearest transcript(H) by the basic bisect algorithm
14. if peak(i) resides outside transcript(H)
15. report peak(i) with transcript(H), intergenic region *
16. else
17. report peak(i) with transcript(H), intron region
18. endif
19. for transcripts(t) within the position of transcript(H)+ search radius
20. if peak(i) resides outside transcripts(t)
21. report peak(i) with transcripts(x), intergenic region *
22. else
23. report peak(i) with transcripts(x), intron region
24. endif
25. end
26. end

*: more details about distance, promoter and bidirectional region judgment

Figure 2: Pseudo-code of the seq2gene algorithm.

runseq2gene inputs/parameters

inputfile An R object input file that records genomic region information (coordinates). This object could be a data frame defined as:

column 1 the unique IDs of peaks/mutations/SNPs;
column 2 the chromosome ID (eg. chr5 or 5);
column 3 the start site of genomic regions;
column 4 the end site of genomic regions (for SNP and point mutations, the difference of start and end is 1bp);
column 5 ... custom defined.

There is one demo data in data.frame format in our package.

> data(Chipseq_Peak_demo)
> class(Chipseq_Peak_demo)
[1] "data.frame"

> head(Chipseq_Peak_demo)

<table>
<thead>
<tr>
<th>peakID</th>
<th>chrom</th>
<th>start</th>
<th>end</th>
<th>signalvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak_59951</td>
<td>chr14</td>
<td>19003706</td>
<td>19004370</td>
<td>6.611026</td>
</tr>
<tr>
<td>Peak_59952</td>
<td>chr14</td>
<td>19003800</td>
<td>19024138</td>
<td>3.450042</td>
</tr>
</tbody>
</table>
3 Peak_59953 chr14 19005068 19005305 10.997456
4 Peak_59954 chr14 19006372 19006587 21.055350
5 Peak_59955 chr14 19013301 19013534 8.242503

Or, the input format could be a GRanges object (from R package GenomicRanges). There is a demo data in GRanges format in our package as well.

```R
> data(GRanges_demo)
> class(GRanges_demo)

[1] "GRanges"
attr(,"package")
[1] "GenomicRanges"

> GRanges_demo[1:3,]

GRanges object with 3 ranges and 3 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>name</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
<td>&lt;character&gt;</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>a</td>
<td>chr1</td>
<td>[1, 7]</td>
<td>-</td>
<td>peak1</td>
</tr>
<tr>
<td>b</td>
<td>chr2</td>
<td>[2, 8]</td>
<td>+</td>
<td>peak2</td>
</tr>
<tr>
<td>c</td>
<td>chr2</td>
<td>[3, 9]</td>
<td>+</td>
<td>peak3</td>
</tr>
</tbody>
</table>

seqinfo: 3 sequences from an unspecified genome; no seqlengths

Note that for this particular GRanges object, the seqnames, ranges, strand, and name columns are necessary. And for a data frame object, the first four columns are orderly. Specifically, here are three more examples.

example 1:

<table>
<thead>
<tr>
<th>peakID</th>
<th>chrom</th>
<th>chromstart</th>
<th>chromend</th>
<th>name</th>
<th>score</th>
<th>strand</th>
<th>thickstart</th>
<th>thickend</th>
</tr>
</thead>
<tbody>
<tr>
<td>peak2</td>
<td>chr7</td>
<td>127477031</td>
<td>127478198</td>
<td>Neg2</td>
<td>0</td>
<td>-</td>
<td>127477031</td>
<td>127478198</td>
</tr>
<tr>
<td>peak3</td>
<td>chr7</td>
<td>127478198</td>
<td>127479365</td>
<td>Neg3</td>
<td>0</td>
<td>-</td>
<td>127478198</td>
<td>127479365</td>
</tr>
</tbody>
</table>

example 2:

<table>
<thead>
<tr>
<th>peakID</th>
<th>Chr</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACS_M_1210</td>
<td>chr9</td>
<td>21754771</td>
<td>21755152</td>
</tr>
<tr>
<td>MACS_M_1211</td>
<td>chr9</td>
<td>21753771</td>
<td>21754023</td>
</tr>
<tr>
<td>MACS_M_1212</td>
<td>chr9</td>
<td>21753901</td>
<td>21754023</td>
</tr>
</tbody>
</table>

example 3:

<table>
<thead>
<tr>
<th>SNP</th>
<th>chr</th>
<th>Physical_position</th>
<th>position_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs953509</td>
<td>9</td>
<td>81560347</td>
<td>81560348</td>
</tr>
<tr>
<td>rs719293</td>
<td>2</td>
<td>50516523</td>
<td>50516524</td>
</tr>
<tr>
<td>rs1394384</td>
<td>17</td>
<td>28813156</td>
<td>28813157</td>
</tr>
<tr>
<td>rs1609772</td>
<td>1</td>
<td>186820222</td>
<td>186820223</td>
</tr>
</tbody>
</table>

**search_radius(unit bp)** A non-negative integer, with which the input genomic regions can be assigned not only to the matched/nearest gene, but also with all genes within a search radius. Default is 150000. Figure 3 illustrates the definition of search_radius, being calculated from the middle of a genomic region to both sides.
promoter_radius (unit bp) A non-negative integer. Default is 200. Note that promoters are calculated from transcription start site (TSS) of genes (Figure 4). Promoters can be about 100-2000 base pairs upstream of their TSSs [14]. User can assign the promoter_radius to define promoter regions in the genome.

promoter_radius2 (unit bp) A non-negative integer. Default is 100. User can as well use this parameter to define downstream regions of the TSSs as promoter.

genome A character specifies the genome type. Currently, “hg38”, “hg19” (human), and “mm10”, “mm9” (mouse) are supported.

adjacent A Boolean. Default is FALSE to search all genes within the search_radius. Using “TRUE” to find the adjacent genes only and ignore parameters “SNP” and “search_radius”.

SNP A Boolean specifies the input object type. By default is FALSE to keep on searching for intron and neighboring genes. Otherwise, runseq2gene stops searching when the input genomic region is residing on a coding gene exon.

PromoterStop A Boolean, “FALSE” by default to keep on searching neighboring genes using the parameter “search_radius”. Otherwise, runseq2gene stops searching for neighboring genes. This parameter has function only if an input genomic region map to promoter of coding gene(s).

NearestTwoDirection A boolean, “TRUE” by default to output the closest left and closest right coding genes with directions. Otherwise, output only the nearest coding gene regardless of direction.

UTR3 A boolean, “FALSE” by default to calculate the distance from genes’ 5UTR. Otherwise, calculate the distance from genes’ 3UTR.
runseq2gene outputs

The function runseq2gene outputs a matrix structured below.

Columns 1-4 The same as the first four columns in the input file.

Columns 5 PeakLength An integer gives the length of the input genomic region. It is the number of base pairs between the start and end of the region.

Columns 6 PeakMtoStartOverlap An integer gives the distance from the TSS of mapped gene to the middle of the genomic region. A negative signal only shows TSS of the mapped gene is at the right of the peak (Figure 5 A-B). Otherwise, PeakMtoStartOverlap reports a numeric range showing the location of overlapped coordinates (Figure 5 C).

Figure 5: The calculation of output PeakMtoStartOverlap. Scenarios could be an intergenic region of interest resides at the upstream (A) or downstream (B) of a coding gene, or a genomic region overlaps with intron or exon of a coding gene (C).

Columns 7 type A character specifies the relationship between the genomic region and the mapped gene (Figure 6)

“Exon” any part of a genomic region overlaps the exon region of the mapped gene;

“Intron” any part of a genomic region overlaps an intron region but not at exon region of the mapped gene;

“cds” any part of a genomic region overlaps the CDS region;

“utr” any part of a genomic region overlaps a UTR region;

“promoter” any part of a genomic region overlaps the promoter region of the mapped gene based on an intergenic region of mapped gene covers the input genomic region;

“promoter_internal” any part of a genomic region overlaps the promoter region of the mapped gene when an adjacent TTS region of mapped gene covers the input genomic region;

“Neareast” the mapped gene is the nearest gene if the genomic region is located in an intergenic region. “L” and “R” show the relative location of mapped genes;

“Neighbor” any mapped genes within the search radius but belongs to none of the prior types.
Figure 6: Six output type values in several scenarios. In each scenario, we map the genomic region of interest in green to the following types of a coding gene: exon (1), intron (2), the nearest (3), promoter (4), Nearest_L and Nearest_R (5), or Promoter_R (6).

Columns 8 BidirectionalRegion A Boolean indicates whether or not the input genomic region is in bidirectional region (Figure 7).
A “bidirectional gene pair” refers to two adjacent genes coded on opposite strands, with their 5’ UTRs oriented toward one another. NA means the genomic region is at exon or intron region.
Figure 7: The definition of output BidirectionalRegion in several scenarios. (1) Two adjacent genes code on opposite strands, with their 5’ ends oriented toward one another: Bidirectional region=TRUE. (2) Both two adjacent genes code on reverse strands: Bidirectional region=FALSE. (3) Both two adjacent genes code on forward strands: Bidirectional region=FALSE. (4) Two adjacent genes code on opposite strands, with their 3’ ends oriented toward one another: Bidirectional region=FALSE.

Columns 9 Chr An integer gives chromosome number of mapped gene.
Columns 10 TSS An integer indicates transcription start site of mapped gene regardless of strand.
Columns 11 TTS An integer indicates transcription termination site of mapped gene regardless of strand.
Columns 12 strand a character indicates whether gene is in forward (+) or reverse (-) direction on chromosome.
Columns 13 gene\_name A character gives official gene name of mapped genes.
Columns 14 source a character gives gene source (Ensembl classification) of mapped genes.
Columns 15 transID A character gives Ensemble transcript ID of mapped genes.

gene2pathway

The gene2pathway step integrates several featured GSA (geneset analysis) algorithms, characterized by the improved FAIME method (Functional Analysis of Individual Microarray/RNAseq Expression)[3][19]. We initially developed FAIME for transcriptomic analysis, which compares the cumulative quantitative effects of genes inside an ontology (set of functional related genes) with those outside thus overcoming a number of difficulties in prior GSA methods[3]. However, sensitivity of the FAIME algorithm remains a challenge as, at a significance level of false discovery rate (FDR) of 0.05, FAIME could identify hundreds of gene-sets, an impractical number for wet-lab validation. Therefore, we introduce in this package a new weighting parameter into the FAIME algorithm to better control the type-I error, especially for large gene-sets. Additionally, we recently used gene2pathway to integrate microarray and RNA-seq data for gene-set analysis (manuscript submitted).

Here we develop the function gene2path\_test as an improved tool for functionally analyzing versatile next generation sequencing data by taking account of quantitative sequence measurements. This function implements the improved FAIME algorithm. This function can run the classical Fisher’s exact test or novel gene2pathway tests.
gene2pathway flowchart

Figure 8 gives the flowchart for the gene2pathway process. Hereafter we use “pathway” to refer functional gene-sets for simplification.

![gene2pathway flowchart]

**Figure 8: gene2pathway flowchart.**

gene2pathway_test inputs/parameters

dat A data frame of gene expression or a matrix of sequencing derived gene-level measurements. The rows of dat correspond to genes, and the columns correspond to sample profile (e.g. Chip-seq peak scores, somatic mutation p-values, RNS-seq or microarray gene expression values).

Note that official gene symbols must label the **dat** rows. The values contained in dat should be either finite or NA. For example:

```
<table>
<thead>
<tr>
<th>Peak.Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARHGEF10</td>
</tr>
<tr>
<td>ARHGAP31</td>
</tr>
<tr>
<td>B4GALT4</td>
</tr>
</tbody>
</table>
```

DataBase A character string assigns an R GSA.genesets object to define gene-set. User can call GSA.read.gmt function in R GSA package to load customized gene-sets with a .gmt format. If not specified, GO defined gene sets (BP, MF, CC) will be used. For example,

```r
> data(MsigDB_C5,package="seq2pathway.data")
> class(MsigDB_C5)
```

```
[1] "GSA.genesets"
```

FisherTest A Boolean value. By default is TRUE to execute the function of the Fisher’s exact test. Otherwise, only executes the function of gene2pathway test.
**EmpiricalTest** A Boolean value. By default is FALSE for multiple-sample dat. When true, gene2pathway_test calculates empirical p-values for gene-sets.

**method** A character string determines which method to calculate the pathway scores. Currently, “FAIME” (default), “KS-rank”, and “cumulative-rank” are supported.

**genome** A character specifies the genome type. Currently, choice of “hg38”, “hg19”, “mm10”, and “mm9” is supported.

**alpha** A positive integer, 5 by default. This is a FAIME-specific parameter. A higher value puts more weights on the most highly-expressed ranks than the lower expressed ranks[^1] [^15].

**logCheck** A Boolean value. By default is FALSE. When true, take the log-transformed values of all genes if the maximum value of sample profile is larger than 20.

**na.rm** A Boolean value indicates whether to keep missing values or not when method=“FAIME”. By default is FALSE.

**B** A positive integer assigns the total number of random sampling trials to calculate the empirical p values. By default is 100.

**min_Intersect_Count** A number decides the cutoff of the minimum number of intersected genes when reporting Fisher’s exact tested results.

**gene2pathway_test outputs**

A list or data frame. If the parameter FisherTest is true, the result is a list including both reports for Fisher’s exact test and the gene2pathway test. Otherwise, only reports the gen2pathway test results. For example, below Table 4.2.3 is the head of result of gene2pathway test.
Table 2: result of gene2pathway

### Examples

The most critical issue in functionally interpreting genomic loci is to bridge non-coding regions with gene function. Seq2pathway offers the capability to discover pathway enrichment caused by long-distance cis-regulation of functional non-coding loci. Here we demonstrate the application on ChIP-seq and RNA-seq data analysis respectively. For ChIP-seq data, we demonstrate a use of runseq2gene and gene2pathway_test in series. To facilitate the comparison with conventional Fisher’s exact test, we demonstrated the use of two additional functions below.

**“FisherTest_GO_BP_MF_CC”** The GO enrichment analysis for coding genes using Fisher’s exact test.

**“FisherTest_MsigDB”** The MSigDB\(^4\) defined functional gene-set enrichment analysis for coding genes using the Fisher’s exact test.

### ChIP-seq data analysis

Map ChIP-seq enriched peaks to genes using runseq2gene

runseq2gene() is one of the key functions in the seq2pathway package. The runseq2gene links sequence-level measurements of genomic regions (including ChIP-seq peaks, SNPs or point mutation coordinates) to gene-level scores. The function has the option to assign non-exon regions to a broader range of neighboring genes than the nearest one, thus facilitating the study of functional non-coding elements. Currently, Seq2pathway only works in Linux or windows.
with python2.7 environment, as it has wrapped python scripts to annotate loci to genes.
To execute runseq2gene, we need to assign input file. An example of input file, Chipseq_Peak_demo, is included in the package.

```r
> data(Chipseq_Peak_demo)
> head(Chipseq_Peak_demo)

        peakID chrom start end   signalvalue
1  Peak_59951 chr14 19003706 19004370 6.611026
2  Peak_59952 chr14 19003800 19024138 3.450042
3  Peak_59953 chr14 19005068 19005305 10.997456
4  Peak_59954 chr14 19006372 19006587 21.055350
5  Peak_59955 chr14 19013301 19013534 8.242503
```

Then user can run demo data below:

```r
> Chipseq_anno <- runseq2gene(
+     inputfile=Chipseq_Peak_demo,
+     genome="hg38", adjacent=FALSE, SNP=FALSE, search_radius=1000,
+     PromoterStop=FALSE, NearestTwoDirection=TRUE)

[2] "Load Reference"
[3] "Check Reference files"
[5] "Start Annotation"
[6] "Finish Annotation"

> class(Chipseq_anno)
[1] "list"

> head(Chipseq_anno[[1]])

        peakID chrom start end PeakLength peakMtoStart_Overlap
1  Peak_59951 chr14 19003706 19004370       664         -36604
2  Peak_59951 chr14 19003706 19004370       664        430303
3  Peak_59952 chr14 19003800 19024138      20338     [19024090, 19024286]
4  Peak_59952 chr14 19003800 19024138      20338          10121
5  Peak_59952 chr14 19003800 19024138      20338          -46535
6  Peak_59952 chr14 19003800 19024138      20338        420372

  type BidirectionalRegion Chr      TSS      TTS  strand
1  Nearest_L            N  chr14 18967434 18999012    +
2  Nearest_R            N  chr14 19402486 19434341    -
3     Exon               <NA> chr14 19024090 19055551    +
4 Promoter_internal     <NA> chr14 19024090 19055551    +
5  Nearest_L            N  chr14 18967434 18999012    +
6  Nearest_R            N  chr14 19402486 19434341    -

  gene_name source transID
1  POTEG protein_coding ENSG00000222036.4
2  POTEM protein_coding ENSG00000187537.10
3  CTD-2314B22.1 lincRNA ENSG00000258314.3
4  CTD-2314B22.1 lincRNA ENSG00000258314.3
5  POTEG protein_coding ENSG00000222036.4
6  POTEM protein_coding ENSG00000187537.10
```
Discover enriched GO terms using `gene2pathway_test` with gene scores

After mapping peaks to genes, we will practice `gene2pathway_test` function. This function summarizes gene scores to pathway-scores for each sample. The function `gene2pathway_test` includes `rungene2pathway` function, which summarizes gene scores to pathway-scores for each sample, and is another main function in our package. The `rungene2pathway` function provides different methods (“FAIME”, “KS-rank”, and “cumulative-rank”) to convert gene-level measurements to pathway-level scores. The function `gene2pathway_test` also includes `FisherTest` function to perform conventional Fisher’s exact test (FET). The `FisherTest` function uses the corrected, common gene background for selected pathways. Hereafter we use ‘pathway’ to refer functional gene-sets including GO for simplification. Following are R example codes.

# Example 1:
Running FAIME and FET against MSigDB defined gene-sets with empirical p-values

```r
> ## give the previously defined gene-sets
data(MsigDB_C5, package="seq2pathway.data")
class(MsigDB_C5)

[1] "GSA.genesets"

> ## load the gene-level measurements, here is an example of ChIP-seq scores
data(dat_chip)
> head(dat_chip)

   peakscore
ABCD4  8.433123
ABHD12B 9.526305
ABHD4  9.988747
AC004817.1 10.086676
AC005477.1 10.086676
AC007375.1 10.186544

> result_FAIME<-gene2pathway_test(dat= dat_chip, DataBase= MsigDB_C5,
     FisherTest=TRUE, EmpiricalTest=TRUE, method="FAIME",
     alpha=5, logCheckALSE, na.rm=FALSE)
```

The output will be a list, which include two data frame. One data set is the result of Fisher’s exact test, with the geneset from MSigDB, the other is the result of `rungene2pathway` function with method ‘FAIME’. We calculated empirical p-values for a single sample.

# Example 2:
Running FAIME and FET against GO defined gene-sets with empirical p-values

```r
> result_FAIME<-gene2pathway_test(dat= dat_chip,
     FisherTest=TRUE, EmpiricalTest=TRUE, method="FAIME",
     alpha=5, logCheck=FALSE, na.rm=FALSE)
```

In our package, there is an R resultant object `dat_gene2path_chip` as demo of `result_FAIME`.

```r
> data(dat_gene2path_chip, package="seq2pathway.data")
> names(dat_gene2path_chip)

[1] "gene2pathway_result.2" "gene2pathway_result.FET"

> class(dat_gene2path_chip$gene2pathway_result.2)
```
GO:0000082 The mitotic cell cycle transition by which a cell in G1 commits to S phase. The process begins with the build...
GO:0000086 The mitotic cell cycle transition by which a cell in G2 commits to M phase. The process begins when the kinase...
GO:0000122 Any process that stops, prevents, or reduces the frequency, rate or extent of transcription from an RNA polymerase II promoter.

<table>
<thead>
<tr>
<th>GO ID</th>
<th>Fisher_Pvalue</th>
<th>Fisher_odds</th>
<th>FDR</th>
<th>Intersect_Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO:0000082</td>
<td>0.3201774</td>
<td>11.80262</td>
<td>0.12</td>
<td>10</td>
</tr>
<tr>
<td>GO:0000086</td>
<td>-0.3358601</td>
<td>15.12201</td>
<td>0.49</td>
<td>6</td>
</tr>
<tr>
<td>GO:0000122</td>
<td>-0.1153585</td>
<td>6.11356</td>
<td>0.16</td>
<td>20</td>
</tr>
</tbody>
</table>

GO:0000082 CDKN3 GPR132 MNAT1 POLE2 PSMA3 PSMA6 PSMB5 PSMB9 PSMA7 PSMB11
GO:0000086 AJUBA DYNC1H1
GO:0000122 AJUBA BMP4 DACT1 DICER1 ESR2 FOXA1 GSC JDP2 NKK2-1 PPM1A PRMT5 PSEN1 ROR1 SALL2 SIX1 SNW1 ...

> class(dat_gene2path_chip$gene2pathway_result.FET)
[1] "list"

> names(dat_gene2path_chip$gene2pathway_result.FET)
[1] "GO_BP" "GO_CC" "GO_MF"

> colnames(dat_gene2path_chip$gene2pathway_result.FET$GO_BP)
[1] "GOID"           "Description"          "Fisher_Pvalue"
[4] "Fisher_odds"   "FDR"           "Intersect_Count"
[7] "GO_gene_inBackground" "GO_gene_raw_Count" "Intersect_gene"

#Example 3:
Running FAIME and FET against GO defined gene-sets without empirical p-values
> result_FAIME<-gene2pathway_test(dat= dat_chip, FisherTest=TRUE, EmpiricalTestALSE, method="FAIME", alpha=5, logCheck=FALSE, na.rm=FALSE)

#Example 4:
Running FAIME only against GO defined gene-sets with empirical p-values
> result_FAIME<-gene2pathway_test(dat= dat_chip, FisherTest=FALSE,EmpiricalTest=TRUE, method="FAIME", alpha=5, logCheck=FALSE, na.rm=FALSE)

Discover enriched GO terms using Fisher’s Exact test without gene scores

There are two functions to run FET in the package seq2pathway. Both perform conditional FET with modified gene background that is the common genes between genome and the gene-set database, e.g., MSigDB (Figure 9)[2]. The FisherTest.GO_BP_MF_CC function uses GO (GO.db.2.14.0) defined gene-sets, and the FisherTest_MsigDB function requires MsidDB defined gene-sets as input.

![Figure 9](image.png)

Figure 9: **Conditional Fisher’s exact test with corrected common background.** The common background between genome and the gene-set database, e.g., MSigDB, is illustrated as a grey region, which contains around 22,000 human coding genes or 15,546 mouse coding genes.

FisherTest_MsigDB function:

- Inputs/parameters:
  - gsmap An R GSA.genesets object defined by the package “GSA” for functional gene-set (or termed as pathway for simplification). For example,
    > data(MsigDB_C5,package="seq2pathway.data")
    > class(MsigDB_C5)
    [1] "GSA.genesets"
  - gs A characteristic vector of gene symbols of interest.
**Genome** A character specifies the genome type. Currently, choice of “hg38”, “hg19”, “mm10”, and “mm9” is supported.

**min.Intersect.Count** A number decides the cutoff of the minimum number of intersected genes when reporting Fisher’s exact tested results.

- **Output:** A data frame of Fisher's exact tested result with the following columns:
  - **GeneSet** MSigDB gene-set names (ID)
  - **Description** MSigDB definition and description for the gene-sets
  - **Fisher_Pvalue** the raw P values
  - **Fisher_odds** estimate of the odds ratios
  - **FDR** the multi-test adjusted P values using the Benjamini and Hochberg method[16]
  - **Intersect_Count** the sizes of the overlap between gene-set genes and the input gene list
  - **MsigDB_gene_inBackground** the counts of genes among each MSigDB gene-set that are also within the given genome background
  - **MsigDB_gene_raw_Count** the original counts of genes in each MSigDB geneset
  - **Intersect_gene** the intersecting genes' symbols

- **An example:**

```r
> data(dat_chip)
> head(dat_chip)
```

<table>
<thead>
<tr>
<th>peakscore</th>
<th>ABCD4</th>
<th>8.433123</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABHD12B</td>
<td>9.526305</td>
<td></td>
</tr>
<tr>
<td>ABHD4</td>
<td>9.988747</td>
<td></td>
</tr>
<tr>
<td>AC004817.1</td>
<td>10.08676</td>
<td></td>
</tr>
<tr>
<td>AC005477.1</td>
<td>10.08676</td>
<td></td>
</tr>
<tr>
<td>AC007375.1</td>
<td>10.186544</td>
<td></td>
</tr>
</tbody>
</table>

```r
> FS_test<-FisherTest_MsigDB(gsmap=MsigDB_C5, gs=as.vector(rownames(dat_chip)))
> head(FS_test)
```

|---------------------|---------------|-------------|-----|-----------------|----------------------------|------------------------|---------------------|---------------------|---------------------|---------------|-----|---------------|----------------------------|------------------------|---------------------|------------------|
FisherTest_GO_BP_MF_CC function:

- Inputs/parameters:
  - **gs**: A characteristic vector of gene symbols, the input genelist.
    Note that the seq2pathway package has prepared an internal R object
    `GO_MF_CC_BP_term_gene_lists_Fromorg.Hs.egGO2EG.rData`, which is formatted from biomaRt_2.20.0
    and org.Hs.eg.db_2.14.0 gene symbols and GO.db_2.14.0 gene ontologies.
  - **genome**: A character specifies the genome type. Currently, choice of “hg38”, “hg19”, “mm10”, and “mm9” is
    supported.
  - **min_Intersect_Count**: A number decides the cutoff of the minimum number of intersected genes when reporting
    Fisher’s exact test results.
  - **Ontology**: A character specifies the Gene Ontology, choice of “GOterm”, “BP”, “MF”, “CC” and “newOntology” is
    supported.
  - **newOntology**: A list of two lists with the same ontology IDs. For each ontology ID, the 1st list is the lists of
defined genes and the 2nd list is the description.

- Outputs:
A list of 3 data frames, each is a result of Fisher’s exact test, using GO CC, BP, MF respectively. Each data
frame reports FET results with the following columns.

  - **GOID**: GO term ID
  - **Description**: GO definition and description for the gene-sets based on the R object GO.db_2.14.0
  - **Fisher_Pvalue**: the raw P values
  - **Fisher_odds**: estimate of the odds ratios
  - **FDR**: the multi-test adjusted P values using the Benjamini and Hochberg method[16]
  - **Intersect_Count**: the sizes of the overlap between GO gene members and the input gene list
  - **GO_gene_inBackground**: the counts of genes among each GO term that are also within a given genome back-
ground
  - **GO_gene_raw_Count**: the original counts of genes in each GO term
  - **Intersect_gene**: the intersecting genes’ symbols

- An example:
  ```r
  > data(dat_chip)
  > head(dat_chip)
  
  peakscore
  ABCD4  8.433123
  ABHD12B  9.526305
  ABHD4  9.988747
  AC004817.1  10.086676
  AC005477.1  10.086676
  AC007375.1  10.186544
  
  > FS_test<- FisherTest_GO_BP_MF_CC(gs=as.vector(rownames(dat_chip)),Ontology="BP")
  [1] "Fisher's exact test done"
  > head(FS_test$GO_BP)
  ```
Add description for genes

The function `addDescription` is wrapped from R package “biomaRt” \[17\][18] to converts gene hgnc_symbol to gene description.

- **Inputs/parameters:**
  - **genome** A character specifies the genome type. Currently, choice “hg19”, “mm10”, and “mm9” is supported.
  - **genevector** A characteristic vector or list of gene symbols.

- **Output:**
  A data frame with two columns, first is the input genelist and second is the biomaRt gene description in details.

- **An example:**

```r
> gene_description<-addDescription(genome="hg19",genevector=as.vector(rownames(dat_chip)))
> head(gene_description)
```

<table>
<thead>
<tr>
<th>hgnc_symbol</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCD4</td>
<td>ATP-binding cassette, sub-family D (ALD), member 4 [Source:HGNC Symbol;Acc:68]</td>
</tr>
<tr>
<td>ABHD12B</td>
<td>abhydrolase domain containing 12B [Source:HGNC Symbol;Acc:19837]</td>
</tr>
<tr>
<td>ABHD4</td>
<td>abhydrolase domain containing 4 [Source:HGNC Symbol;Acc:20154]</td>
</tr>
<tr>
<td>ACIN1</td>
<td>apoptotic chromatin condensation inducer 1 [Source:HGNC Symbol;Acc:17066]</td>
</tr>
<tr>
<td>ACOT1</td>
<td>acyl-CoA thioesterase 1 [Source:HGNC Symbol;Acc:33128]</td>
</tr>
<tr>
<td>ACOT2</td>
<td>acyl-CoA thioesterase 2 [Source:HGNC Symbol;Acc:18431]</td>
</tr>
</tbody>
</table>

RNA-seq data analysis

RNA-seq is increasingly used for measuring gene expression levels. Normally, RNA-seq measures multiple samples from more than one sample-groups. Base on expressions on the gene-level, user can run the `gene2pathway_test` function and skip the `runseq2gene()` function.

Here is an example to run `gene2pathway_test` function for RNA-seq data, using an example data in the package.

```r
> data(dat_RNA)
> head(dat_RNA)
```
Using the inputs similar to the example coding for ChIPseq data, the output of the gene2pathway_test function running RNAseq data will be a matrix of pathway scores for multiple samples.

```r
> dat_gene2path_RNA <- gene2pathway_test(dat=dat_RNA, DataBase=MsigDB_C5, EmpiricalTest=FALSE, alpha=5, logCheck=FALSE, method="FAIME", na.rm=TRUE)
> head(dat_gene2path_RNA$gene2pathway_result.2)
```

<table>
<thead>
<tr>
<th>Des</th>
<th>TCGA_2841</th>
<th>TCGA_2840</th>
<th>TCGA_2843</th>
<th>TCGA_2842</th>
<th>TCGA_2845</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6.3606</td>
<td>10.2275</td>
<td>2.5394</td>
<td>2.8203</td>
<td>7.8670</td>
</tr>
<tr>
<td>A1CF</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>A2LD1</td>
<td>1.2489</td>
<td>1.3508</td>
<td>2.1397</td>
<td>1.9969</td>
<td>1.0495</td>
</tr>
<tr>
<td>A2M</td>
<td>0.2507</td>
<td>2.4767</td>
<td>3.8131</td>
<td>0.9692</td>
<td>1.7197</td>
</tr>
<tr>
<td>A2ML1</td>
<td>0.0710</td>
<td>0.0473</td>
<td>0.2541</td>
<td>0.0538</td>
<td>0.1098</td>
</tr>
</tbody>
</table>

```r
> head(dat_gene2path_RNA$gene2pathway_result.FET)
```
<table>
<thead>
<tr>
<th>GeneSet Description</th>
<th>Fisher P-value</th>
<th>Fisher #qds</th>
<th>FDR Intersect</th>
<th>MagDB #gene</th>
<th>MagDB Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>HYDROLASE ACTIVITY _ACTING_ON_ACID</td>
<td>1.608744e-20</td>
<td>56.6790665</td>
<td>1.584249e-17</td>
<td>37</td>
<td>39</td>
</tr>
<tr>
<td>CHEMOKINE RECEPTOR BINDING</td>
<td>9.352298e-20</td>
<td>29.8938093</td>
<td>4.016531e-17</td>
<td>39</td>
<td>43</td>
</tr>
<tr>
<td>PRIMARY_ACTIVE TRANSMEM-BRANE TRANSPORTER ACTIVITY</td>
<td>1.629424e-19</td>
<td>37.7811394</td>
<td>4.016531e-17</td>
<td>37</td>
<td>40</td>
</tr>
<tr>
<td>ATPASE_ACTIVITY COUPLED_TO_MOVE-MENT_OF_SUBSTANCES</td>
<td>1.629424e-19</td>
<td>37.7811394</td>
<td>4.016531e-17</td>
<td>37</td>
<td>40</td>
</tr>
<tr>
<td>CHEMOKINE ACTIVITY</td>
<td>3.45165e-19</td>
<td>29.1130402</td>
<td>6.808585e-19</td>
<td>38</td>
<td>42</td>
</tr>
<tr>
<td>BIOPOLYMER_METABOLIC_PROCESS</td>
<td>1.206876e-15</td>
<td>0.5819836</td>
<td>1.983299e-13</td>
<td>294</td>
<td>1673</td>
</tr>
</tbody>
</table>

Gene Set Details:

- **HYDROLASE ACTIVITY**: This gene set includes genes involved in hydrolytic enzymes, such as HMG-CoA reductase, which is involved in lipid metabolism.
- **CHEMOKINE RECEPTOR BINDING**: This set includes genes involved in chemokine receptor signaling pathways, crucial for immune response and inflammation.
- **PRIMARY_ACTIVE TRANSMEM-BRANE TRANSPORTER ACTIVITY**: Genes in this set are responsible for primary active transport across membranes, essential for cellular processes.
- **ATPASE_ACTIVITY COUPLED_TO_MOVE-MENT_OF_SUBSTANCES**: Includes ATPases essential for coupled transport of substances across membranes.
- **CHEMOKINE ACTIVITY**: Key roles in immune response and inflammation.
- **BIOPOLYMER_METABOLIC_PROCESS**: Encompasses genes involved in metabolic processes related to biopolymers, such as proteins and carbohydrates.
R environment session

> require(seq2pathway)
> sessionInfo();

R version 3.3.1 (2016-06-21)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.1 LTS

locale:
[1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C
[3] LC_TIME=en_US.UTF-8 LC_COLLATE=C
[5] LC_MONETARY=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_US.UTF-8 LC_NAME=C
[9] LC_ADDRESS=C LC_TELEPHONE=C

attached base packages:
[1] stats graphics grDevices utils datasets methods base

other attached packages:
[1] seq2pathway_1.6.0 seq2pathway.data_1.5.0

loaded via a namespace (and not attached):
[1] Rcpp_0.12.7 XVector_0.14.0 GenomeInfoDb_1.10.0
[4] RColorBrewer_1.1-2 plyr_1.8.4 zlibbioc_1.20.0
[7] bitops_1.0-6 iterators_1.0.8 tools_3.3.1
[10] biomarT_2.30.0 rpart_4.1-10 preProcessCore_1.36.0
[13] RSQLite_1.0.0 gtable_0.2.0 lattice_0.20-34
[16] Matrix_1.2-7.1 foreach_1.4.3 DBI_0.5-1
[19] parallel_3.3.1 WGCNA_1.51 gridExtra_2.2.1
[22] cluster_2.0.5 S4Vectors_0.12.0 IRanges_2.8.0
[25] stats4_3.3.1 grid_3.3.1 nnet_7.3-12
[28] impute_1.48.0 data.table_1.9.6 Biobase_2.34.0
[31] AnnotationDbi_1.36.0 XML_3.98-1.4 survival_2.39-5
[34]外国0.8-67 GSA_1.03 latticeExtra_0.6-28
[37] Formula_1.2-1 GO.db_3.4.0 ggplot2_2.1.0
[40] fastcluster_1.1.21 GenomicRanges_1.26.0 Hmisc_3.17-4
[43] scales_0.4.0 codetools_0.2-15 matrixStats_0.51.0
[46] splines_3.3.1 BiocGenerics_0.20.0 dynamicTreeCut_1.63-1
[49] colorspace_1.2-7 acepack_1.3-3.3 RCurl_1.95-4.8
[52] doParallel_1.0.10 munsell_0.4.3 chron_2.3-47

References


