1 Introduction

graphite (GRAPH Interaction from pathway Topological Environment) was designed to:
- provide networks derived from eight public pathway databases,
- automate the conversion of node identifiers (e.g. from Entrez IDs to gene symbols),
facilitate the execution of topological pathway analyses on the provided networks.

The pathway databases available in this version are:

- KEGG [1]
- Panther [10]
- PathBank [13]
- PharmGKB [11]
- Reactome [2]
- SMPDB [12]
- WikiPathways [14]

All provided pathways are annotated in human and, since version 1.14, other 13 species are available.

graphite pathways are collected and pre-processed at every BioConductor release (roughly every 6 months). This guarantees the synchronization of the provided data with all other BioConductor annotation packages.

The topological pathway analyses directly supported are:

- SPIA [5, 6, 4]
- topologyGSA [7]

Since version 1.24, every graphite pathway can be accessed through three separate views:

- **gene-only network** where metabolite nodes have been removed and edges have been propagated through them;
- **metabolite-only network** where protein nodes have been removed and edges have been propagated through them;
- **mixed network** containing both proteins and metabolites, useful for metabolomic and transcriptomic data integration.

## 2 Pathway topology conversion to gene network

In order to gather curated information about pathways, we have collected data from different public databases that have emerged as reference points for the systems biology community. The KEGG database has been in development by Kanehisa Laboratories since 1995, and is now a prominent knowledge base for integration and interpretation of large-scale molecular data sets generated by genome sequencing and other high-throughput experimental technologies. KEGG is the only pathway database not in BioPax format, as it stores the information using the KGML format. KEGG pathways (KGML format) provides maps for both signaling and metabolic pathways [1]. Reactome, backed by the EBI, is one of the most complete repository; it is frequently updated and provides a semantically rich description of each pathway [2]. Finally, Panther [10] is data are a comprehensive, curated database of pathways, protein families, trees, subfamilies and functions available at http://pantherdb.org backed by the University of Southern California. Since version 1.24, graphite contains also SMPDB [12] and PharmGKB [11], important resources for metabolomic data analyses.
2.1 Pathway definition

`graphite` pathways are derived by conversion of KGML and BioPax data formats. KEGG database provides a separate XML file, one for each pathway. Thus, a pathway is defined by all the reactions defined within each file. For the other databases, we define a pathway for each element of type `pathway` in the BioPax document.

2.2 Nodes with multiple elements

Within a pathway, a node can correspond to multiple gene products. These nodes with multiple elements can be divided into protein complexes (proteins linked by protein-protein interactions) and the groups containing alternative members (genes generally with similar biochemical functions). When considering signal propagation these groups should be considered differently; the first (hereafter group AND) should be expanded into a clique (all proteins connected to the others), while the second (hereafter group OR) should be expanded without connection among the contained elements.

In the KGML format there are two ways of defining nodes with multiple elements: protein complexes (group AND defined by entry `type="group"`) and group with alternative members (group OR defined by entry `type="gene"`). In the BioPax format only one type of group is allowed: protein complexes (group AND) with type `complex`. However, it often happens that protein tag contains multiple `xref` pointing to alternative elements of the process (group OR).

2.3 Metabolite and protein-mediated interactions

Metabolites mediated interactions are interactions for which a metabolite acts as a bridge between two proteins. Since metabolites are not measured during gene expression analysis (using microarray or RNA-seq), their removal from the original network is useful. However, the trivial elimination of the metabolites, without signal propagation, will strongly bias the topology, interrupting the signals that pass through them. If element $A$ is linked to metabolite $c$ and metabolite $c$ is linked to element $B$, thus elements $A$ should be linked to elements $B$.

Within the KGML format, there are two different ways of describing a metabolite mediated interaction: i) direct interaction `type="PPrel"` ($A$ interacts with $B$ through metabolite $c$) and ii) indirect one `type="PCrel"` ($A$ interacts with metabolite $c$ and $c$ interacts with $B$).

Since proper signal propagation is crucial for topological gene set analysis we decided to include additional rules for the propagation reconstructing a connection between two genes connected through a series of metabolites. Not all metabolites are considered for the propagation because some of them, such as Hydrogen, H2O, ATP, ADP etc., are highly frequent in map descriptions and the signal propagation through them would lead to degenerate too long chains of metabolites. The metabolites not considered for propagation are not characteristic of a specific reaction but secondary substrates/products widely shared among different processes.

Since version 1.24, `graphite` can be used also for metabolomics pathway analyses. We applied the same procedure described above to provide network with metabolites only propagating the signal through proteins.

Finally, `graphite` includes pathways containing both proteins and metabolites (without edges propagation) allowing also integrated pathway analyses of gene expression and metabolomic data.
2.4 Edge attributes

`graphite` allows the user to see the single/multiple relation types that characterize an edge. The type of edges have been preserved to remain as close as possible to the original annotations. Some new types have been introduced due to the needs of the topological conversion. For instance a `Process(indirect)` is introduced when the edge is generated propagating the signal from a gene to another gene passing through metabolites, or from a metabolite to another metabolite passing through proteins.

2.5 Loading pathways

Human pathways are natively distributed with the package. Since the version 1.14.0, non-human pathway data are also available and can be downloaded automatically using the `pathways` function.

`pathways` requires the name of the specie of interest and the name of the pathway database name as follows:

```r
> humanReactome <- pathways("hsapiens", "reactome")
> names(humanReactome)[1:10]
> p <- humanReactome["ABC-family proteins mediated transport"]
> p
"ABC-family proteins mediated transport" pathway
Native ID = R-HSA-382556
Database = Reactome
Species = hsapiens
Number of nodes = 123
Number of edges = 1886
Retrieved on = 28-04-2024
URL = http://reactome.org/PathwayBrowser/#/R-HSA-382556
```

A pathway database is a list of pathways. We can access a pathway through its name, as above, or through its position in the list, as follow:

```r
> p <- humanReactome[[21]]
> pathwayTitle(p)
[1] "Activation of BAD and translocation to mitochondria"
```

In the pathway, nodes represent genes/proteins:
Edges can be characterized by multiple functional relationships:

> head(edges(p))

<table>
<thead>
<tr>
<th>src_type</th>
<th>src</th>
<th>dest</th>
<th>_type</th>
<th>dest</th>
<th>direction</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UNIPROT P10415</td>
<td>UNIPROT P55957</td>
<td>undirected</td>
<td>Binding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>UNIPROT P10415</td>
<td>UNIPROT Q92934</td>
<td>undirected</td>
<td>Binding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>UNIPROT P27348</td>
<td>UNIPROT Q92934</td>
<td>undirected</td>
<td>Binding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>UNIPROT P31749</td>
<td>UNIPROT Q92934</td>
<td>directed</td>
<td>Control(Out: ACTIVATION of BiochemicalReaction)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>UNIPROT P31751</td>
<td>UNIPROT Q92934</td>
<td>directed</td>
<td>Control(Out: ACTIVATION of BiochemicalReaction)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>UNIPROT P31946</td>
<td>UNIPROT Q92934</td>
<td>undirected</td>
<td>Binding</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

By default, the function `edges` and `nodes` provide the edges or the nodes of the pathway containing only proteins with signals propagated through metabolites. Since version 1.24, using the option `which` we can access the pathway with proteins and metabolites (`which` = "mixed") or the pathway with only metabolites with edges propagated through proteins (`which` = "metabolites").

> head(nodes(p), which = "mixed")

> head(edges(p), which = "mixed")

<table>
<thead>
<tr>
<th>src_type</th>
<th>src</th>
<th>dest</th>
<th>_type</th>
<th>dest</th>
<th>direction</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UNIPROT P10415</td>
<td>UNIPROT P55957</td>
<td>undirected</td>
<td>Binding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>UNIPROT P10415</td>
<td>UNIPROT Q92934</td>
<td>undirected</td>
<td>Binding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>UNIPROT P27348</td>
<td>UNIPROT Q92934</td>
<td>undirected</td>
<td>Binding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>UNIPROT P31749</td>
<td>UNIPROT Q92934</td>
<td>directed</td>
<td>Control(Out: ACTIVATION of BiochemicalReaction)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>UNIPROT P31751</td>
<td>UNIPROT Q92934</td>
<td>directed</td>
<td>Control(Out: ACTIVATION of BiochemicalReaction)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>UNIPROT P31946</td>
<td>UNIPROT Q92934</td>
<td>undirected</td>
<td>Binding</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
These same steps can be used to access the KEGG, Panther, PathBank, PathBank, PharmGKB, Reactome, SMPDB and WikiPathways databases for Homo sapiens, but not all such databases are available for all other species. To know the pathway data available the user can use the `pathwayDatabases()` function:

```r
> pathwayDatabases()

species   database
1  athaliana  kegg
2  athaliana  pathbank
3  athaliana  wikipathways
4    btaurus  kegg
5    btaurus  pathbank
6    btaurus  reactome
7    btaurus  wikipathways
8  celegans   kegg
9  celegans   pathbank
10  celegans  reactome
11  celegans  wikipathways
12  cfamiliaris kegg
13  cfamiliaris reactome
14  cfamiliaris wikipathways
15  dmelanogaster kegg
16  dmelanogaster pathbank
17  dmelanogaster reactome
18  dmelanogaster wikipathways
19     drerio  kegg
20     drerio  reactome
21     drerio  wikipathways
22      ecoli  kegg
23      ecoli  pathbank
24      ecoli  wikipathways
25    ggallus   kegg
26    ggallus  reactome
27    ggallus  wikipathways
28    hsapiens kegg
29    hsapiens panther
30    hsapiens pathbank
31    hsapiens pharmgkb
32    hsapiens reactome
33    hsapiens smpdb
34    hsapiens wikipathways
35     mmusculus kegg
36     mmusculus pathbank
37     mmusculus reactome
38     mmusculus wikipathways
39     rnorvegicus kegg
40     rnorvegicus pathbank
41     rnorvegicus reactome
42     rnorvegicus wikipathways
43     scerevisiae kegg
44     scerevisiae pathbank
```
3 Pathway graph

The function `pathwayGraph` builds a `graphNEL` object from a pathway `p`:

```r
> g <- pathwayGraph(p)
> g
A graphNEL graph with directed edges
Number of Nodes = 15
Number of Edges = 39
```

```r
> edgeData(g)[1]
$`UNIPROT:P10415|UNIPROT:P55957`'
$`UNIPROT:P10415|UNIPROT:P55957`$weight
[1] 1

$`UNIPROT:P10415|UNIPROT:P55957`$edgeType
[1] "Binding"
```

Similarly to the `edges` and `nodes`, by default, the function `pathwayGraph` provide the pathway with only proteins and edges propagated through metabolites. Since version 1.24, using the option `which` we can access the pathway with proteins and metabolites (`which = "mixed"`) or the pathway with only metabolites with edges propagated through proteins (`which = "metabolites"`).

```r
> g <- pathwayGraph(p, which = "mixed")
> g
A graphNEL graph with directed edges
Number of Nodes = 19
Number of Edges = 69
```

```r
> edgeData(g)[1]
$`CHEBI:15377|CHEBI:43474`'
$`CHEBI:15377|CHEBI:43474`$weight
[1] 1

$`CHEBI:15377|CHEBI:43474`$edgeType
[1] "Process(BiochemicalReaction)"
```
4 Identifiers

Gene annotations databases are widely used as public repositories of biological information. Our current knowledge on biological elements is spread out over a number of databases (such as: Entrez Gene, RefSeq, backed by the NCBI http://www.ncbi.nlm.nih.gov/, UniProt, ENSEMBL backed by the EBI http://www.ebi.ac.uk/ to name just a few), specialised on a subset of specific biological entities (for instance, UniProt focuses on proteins while Entrez Gene focuses on genes). Key identifiers (IDs) in the internal structure of each database uniquely represent biological entities, thus biological entities can be identified by homogeneous IDs according to the selected database they refer to. Due to their different origins and specificity, switching from an ID to another is possible but not trivial: there could be either no correspondence among them or many-to-many relations. For detailed information about IDs, their structures and differences please consult those resources.

The function `converterIdentifiers` allows the user to convert the pathway IDs into different types, to fit the user needs. This mapping process, however, may lead to the loss of some nodes (not all identifiers may be recognized) and has an impact on the topology of the network (one ID may correspond to multiple IDs in another annotation or vice versa). We based the function of ID conversion through the species-specific Bioconductor `AnnotationDbi`, such as `org.Hs.eg.db`. Since the version 1.14, all the conversion supported in the annotation packages can be used. `converterIdentifiers` needs a list of pathways or a single pathway and a string describing the type of the identifier as provided by an Annotation package (for example, "ENTREZID"), while the values "entrez", "symbol" remains for backward compatibility.

```r
> pSymbol <- convertIdentifiers(p, "SYMBOL")
> pSymbol

"Activation of BAD and translocation to mitochondria" pathway
Native ID  = R-HSA-111447
Database   = Reactome
Species    = hsapiens
Number of nodes = 19
Number of edges = 59
Retrieved on = 28-04-2024
URL = http://reactome.org/PathwayBrowser/#/R-HSA-111447

> head(nodes(pSymbol))
[1] "SYMBOL:BCL2" "SYMBOL:YWHAQ" "SYMBOL:AKTI1" "SYMBOL:AKT2" "SYMBOL:YWHAB"
[6] "SYMBOL:SFN"

> reactomeSymbol <- convertIdentifiers(humanReactome[1:5], "SYMBOL")
```

Since the introduction of metabolites in `graphite` pathways (version 1.24), also the metabolite ID conversion is possible.
5 Cytoscape Plot

Several pathways have a huge number of nodes and edges, thus there is the need of an efficient system of visualization. To this end, `graphite` uses the `Rcy3` package to export the network to Cytoscape 3, through the function `cytoscapePlot`. Cytoscape is a Java based software specifically built to manage biological network complexity and for this reason it is widely used by the biological community. Since version 1.24, using the option `which` we can access the pathway with proteins and metabolites (`which = "mixed"`) or the pathway with only metabolites and edges propagated through proteins (`which = "metabolites"`).

```r
> cytoscapePlot(convertIdentifiers(reactome$'Unwinding of DNA', "symbol"), which = "mixed")
```

6 Topological pathway analysis

`graphite` gives access to two types of topological pathway analyses. For more details on the results obtained by these analyses see the corresponding R packages.

Note that, since version 1.24, given the introduction of nodes with mixed types, all the ID in the pathway have been prefixed with the type of identifier (e.g. ENTREZID:7157 or SYMBOL:TP53). Thus, also the gene expression or metabolite data, in order to match with the pathway nodes, should be in the same format (e.g. ENTREZID:7157 or SYMBOL:TP53).

6.1 SPIA

The analysis with `SPIA` requires the conversion of gene-gene networks in a suitable format. This conversion is performed by the function `prepareSPIA` that must be executed before the analysis command `runSPIA`. The `SPIA` data will be saved in the current working directory; every time you change it, you should also re-run `prepareSPIA`. Edges not included in SPIA
have been coerced into the admitted SPIA types. Metabolite mediated interactions (edges of propagation through metabolites) annotated in graphite with the "indirect" type are mapped into the SPIA edge type "indirect effect". For more details see the SPIA package [5, 6, 4].

```r
> library(SPIA)
> data(colorectalcancer)
> library(hgu133plus2.db)
> top$ENTREZ <- mapIds(hgu133plus2.db, 
+   top$ID, "ENTREZID", "PROBEID", multiVals = "first")
> top <- top[!is.na(top$ENTREZ) & !duplicated(top$ENTREZ), ]
> top$ENTREZ <- paste("ENTREZID", top$ENTREZ, sep = ":")
> tg1 <- top[top$adj.P.Val < 0.05, ]
> DE_Colorectal = tg1$logFC
> names(DE_Colorectal) <- tg1$ENTREZ
> ALL_Colorectal <- top$ENTREZ
> kegg <- pathways("hsapiens", "kegg")
> kegg <- convertIdentifiers(kegg, "ENTREZID")
> kegg <- kegg[c("Bladder cancer", "Cytosolic DNA-sensing pathway")]
> prepareSPIA(kegg, "keggEx")
> res <- runSPIA(de = DE_Colorectal, all = ALL_Colorectal, "keggEx")

Done pathway 1 : Bladder cancer..
Done pathway 2 : Cytosolic DNA-sensing pathway..
Done pathway 3 : <graphite_placeholder>..

> res

<table>
<thead>
<tr>
<th>Name</th>
<th>pSize</th>
<th>NDE</th>
<th>pNDE</th>
<th>tA</th>
<th>pPERT</th>
<th>pG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cytosolic DNA-sensing pathway</td>
<td>58</td>
<td>15</td>
<td>0.1986424</td>
<td>-2.0308063</td>
<td>0.753</td>
<td>0.4337664</td>
</tr>
<tr>
<td>Bladder cancer</td>
<td>28</td>
<td>4</td>
<td>0.8568536</td>
<td>-0.8630436</td>
<td>0.709</td>
<td>0.9102845</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pGFdr</th>
<th>pGFWER</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6506495</td>
<td>1</td>
<td>Inhibited</td>
</tr>
<tr>
<td>0.9102845</td>
<td>1</td>
<td>Inhibited</td>
</tr>
</tbody>
</table>

6.2 topologyGSA

topologyGSA uses graphical models to test the pathway components and to highlight those involved in its deregulation.

In graphite, topologyGSA has a dedicated function, runTopologyGSA, which performs the analysis on a single pathway or on a pathway list.

```r
> library(topologyGSA)
> data(examples)
> colnames(y1) <- paste("SYMBOL", colnames(y1), sep = ":")
> colnames(y2) <- paste("SYMBOL", colnames(y2), sep = ":")
> kegg <- pathways("hsapiens", "kegg")
> p <- convertIdentifiers(kegg["Fc epsilon RI signaling pathway"], "SYMBOL")
> runTopologyGSA(p, "var", y1, y2, 0.05)

Pathway Variance Test

data: expl, exp2 and g
The function `runTopoLogyGSA`, which easily performs the analysis on the entire pathway database, provides as result a list of two elements: a list with the results of the pathway analyses and the list of generated errors.

For more details see the `topologyGSA` package [7].

7 Build pathway

In `graphite`, it is also possible build a pathway object using `buildPathway`. This function creates a new object of type `Pathway` given a data frame describing its edges. The edges should be divided in three dataframes containing edges between proteins, edges between proteins and metabolites, and edges between metabolites, and the should be included in the new pathway using `proteinEdges`, `mixedEdges` and `metaboliteEdges` respectively. This practice will guarantee the compatibility with `convertIdentifiers`, this only works with the types of identifiers that are provided in the Annotation package (for example using the string, “ENTREZID”).

```r
> edges <- data.frame(src_type = "ENTREZID", src="672", 
+                     dest_type = "ENTREZID", dest="7157", 
+                     direction="undirected", type="binding")
> pathway <- buildPathway("#1", "example", "hsapiens", "database", 
+                        proteinEdges = edges)

> edges <- data.frame(src_type = "ENTREZID", src="672", 
+                     dest_type = "ENTREZID", dest="7157", 
+                     direction="undirected", type="binding")
> edgemix <- data.frame(src_type = "CHEBI", src="77750", 
+                       dest_type = "ENTREZID", dest="7157", 
+                       direction="undirected", type="biochemicalReaction")
> edgemet <- data.frame(src_type = "CHEBI", src="15351", 
+                       dest_type = "CHEBI", dest="77750", 
+                       direction="undirected", type="biochemicalReaction")
> pathway <- buildPathway("#1", "example", "hsapiens", "database", 
+                        proteinEdges = edges, 
+                        mixedEdges = edgemix, 
+                        metaboliteEdges = edgemet)
```

8 Parallelism

Some of `graphite` operations can be made significantly faster by exploiting the parallelism offered by recent hardware. Here is a list of the functions providing this option:

- `convertIdentifiers`
- `runTopoLogyGSA`

To exploit parallel processing you are going to need two ingredients. First, you have to specify the maximum number of cores that the package can use. For instance, if you have 6 cores on your computer you can set:
Your code, then, should try to call **graphite** functions passing entire lists of pathways; you should not loop manually.

```r
> original <- pathways("hsapiens", "reactome")
> converted <- convertIdentifiers(original, "SYMBOL")
```

**Warning: Parallelism is not guaranteed to reduce run times.** Splitting a given operation over multiple cores requires some coordination, and that has its own costs. In certain situations (small number of pathways / small number of cores) the overhead may actually make run times worse. Our suggestion is thus to:

- always set `Ncpus` to a value smaller or equal to the actual number of hardware cores (see `parallel::detectCores` for an indication);
- in any case, start from a small number like 2 or 4, and then work your way up measuring the actual speedups on your own hardware.

### 9 Session Information

```r
> sessionInfo()
```

R version 4.4.0 beta (2024-04-15 r86425)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 22.04.4 LTS

Matrix products: default
BLAS: /home/biocbuild/bbs-3.19-bioc/R/lib/libRblas.so
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0

locale:
[1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C
[3] LC_TIME=en_US.UTF-8 LC_COLLATE=en_US.UTF-8
[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

time zone: America/New_York
tzcode source: system (glibc)

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

other attached packages:
[1] topologyGSA_1.5.0  hgu133plus2.db_3.13.0  org.Hs.eg.db_3.19.1
loaded via a namespace (and not attached):
[1] viridisLite_0.4.2  blob_1.2.4
[3] fields_15.2      R.utils_2.12.3
[5] Biostrings_2.72.0 bitops_1.0-7
[7] fastmap_1.1.1     RCurl_1.98-1.14
[9] GenomicAlignments_1.40.0 dotCall64_1.1-1
[11] XML_3.99-0.16.1  digest_0.6.35
[13] Lifecycle_1.0.4  KEGGREST_1.44.0
[15] RSQLite_2.3.6    magrittr_2.0.3
[17] compiler_4.4.0   rlang_1.1.3
[19] tools_4.4.0      igraph_2.0.3
[21] yaml_2.3.8       rtracklayer_1.64.0
[23] knitr_1.46       S4Arrays_1.4.0
[25] bit_4.0.5        curl_5.2.1
[27] DelayedArray_0.30.0 abind_1.4-5
[29] BiocParallel_1.38.0 R.cache_0.16.0
[31] purrr_1.0.2      R.oo_1.26.0
[33] qgraph_2.38.0    grid_4.4.0
[35] xtable_1.8-4     SummarizedExperiment_1.34.0
[37] cli_3.6.2        mvtnorm_1.2-4
[39] rmarkdown_2.26   crayon_1.5.2
[41] httr_1.4.7       rjson_0.2.21
[43] DBI_1.2.2        cachem_1.0.8
[45] maps_3.4.2       zlibbioc_1.50.0
[47] parallel_4.4.0   BiocManager_1.30.22
[49] XVector_0.44.0   restfulr_0.0.15
[51] MatrixStats_1.3.0 vctrs_0.6.5
[53] Matrix_1.7-0     jsonlite_1.8.8
[55] bit64_4.0.5      Rgraphviz_2.48.0
[57] GenomicFeatures_1.56.0 glue_1.7.0
[59] spam_2.10-0      annotate_1.82.0
[61] codetools_0.2-20  GenomeInfoDb_1.40.0
[63] GenomicRanges_1.56.0 BiocIO_1.14.0
[65] UCSC.utils_1.0.0  qtl_1.66
[67] rappdirs_0.3.3    htmltools_0.5.8.1
[69] GenomeInfoDbData_1.2.12 R6_2.5.1
[71] gRbase_2.0.1      evaluate_0.23
[73] lattice_0.22-6   R.methodsS3_1.8.2
[75] png_0.1-8         Rsamtools_2.20.0
[77] memoise_2.0.1     BiocStyle_2.32.0
[79] Rcpp_1.0.12       R.rsp_0.46.0
[81] SparseArray_1.4.0 xfun_0.43
[83] MatrixGenerics_1.16.0 pkgconfig_2.0.3
References


