The OGRE user guide

Sven Berres

11/21/2022

Abstract

OGRE calculates overlap between user defined annotated genomic region datasets. Any regions can be supplied such as public annotations (genes), genetic variation (SNPs, mutations), regulatory elements (TFBS, promoters, CpG islands) and basically all types of NGS output from sequencing experiments. After overlap calculation, key numbers help analyse the extend of overlaps which can also be visualized at a genomic level. To start OGRE's GUI use function SHREC() in your R console. Find additional information and tutorials on github (https://github.com/svenbioinf/OGRE/). OGRE package version: 0.99.8

Installation

Install OGRE using Bioconductor's package installer.

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

Load the OGRE package:

```
library(OGRE)
```

Quick start- load datasets from hard drive

To start up OGRE you have to generate an `OGREDataSet` that is used to store your datasets and additional information about the analysis that you are conducting. Query and subjects files can be conveniently stored in their own folders as GenomicRanges objects in form of stored .rds / .RDS files. We point OGRE to the correct location by supplying a path for each folder with the character vectors `queryFolder` and `subjectFolder` . In this vignette we are using lightweight query and subject example data sets to show OGRE's functionality.

```
myQueryFolder <- file.path(system.file('extdata', package = 'OGRE'),"query")
mySubjectFolder <- file.path(system.file('extdata', package = 'OGRE'),"subject")

myOGRE <- OGREDataSetFromDir(queryFolder=myQueryFolder,
                             subjectFolder=mySubjectFolder)
```

```
## Initializing OGREDataSet...
```

By monitoring OGRE's metadata information you can make sure the input paths you supplied are stored correctly.

```
metadata(myOGRE)
```

```
## $queryFolder
## [1] "/home/bioinf/R/x86_64-pc-linux-gnu-library/4.2/OGRE/extdata/query"
##
## $subjectFolder
## [1] "/home/bioinf/R/x86_64-pc-linux-gnu-library/4.2/OGRE/extdata/subject"
##
## $outputFolder
## [1] "/home/bioinf/R/x86_64-pc-linux-gnu-library/4.2/OGRE/extdata/output"
##
## $gvizPlotsFolder
## [1] "/home/bioinf/R/x86_64-pc-linux-gnu-library/4.2/OGRE/extdata/gvizPlots"
##
## $summaryDT
## list()
##
## $itracks
## list()
```

Query and subject datasets are read by `loadAnnotations()` and stored in the `OGREDataSet` as `GRanges` objects. We are going to read in the following example datasets:

- query "genes" (242 Protein coding genes)
- subject "CGI" (365 CpG islands)
- subject "TFBS" (48761 Transcription factor binding sites)

```
myOGRE <- loadAnnotations(myOGRE)
```

```
## Reading query dataset...
```

```
## Reading subject datasets...
```

OGRE uses your dataset file names to label query and subjects internally, we can check these names by using the `names()` function since every `OGREDataSet` is a `GRangesList` .

```
names(myOGRE)
```

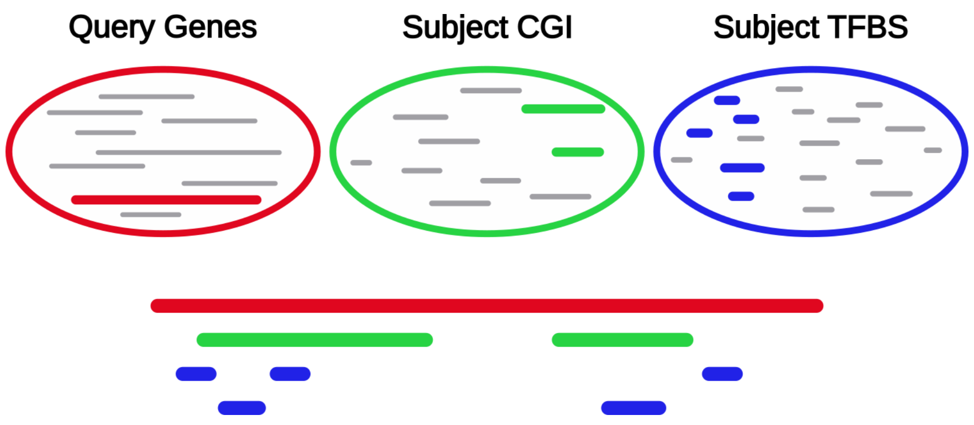
```
## [1] "genes" "CGI"   "TFBS"
```

Let’s have a look at the stored datasets:

```
myOGRE
```

```
## GRangesList object of length 3:
## $genes
## GRanges object with 242 ranges and 3 metadata columns:
##         seqnames            ranges strand |              ID        name
##            <Rle>         <IRanges>  <Rle> |     <character> <character>
##     [1]       21 10906201-11029719      - | ENSG00000166157        TPTE
##     [2]       21 14741931-14745386      - | ENSG00000256715  AL050302.1
##     [3]       21 14982498-15013906      + | ENSG00000166351       POTED
##     [4]       21 15051621-15053459      - | ENSG00000269011  AL050303.1
##     [5]       21 15481134-15583166      - | ENSG00000188992        LIPI
##     ...      ...               ...    ... .             ...         ...
##   [238]       21 47720095-47743789      - | ENSG00000160298    C21orf58
##   [239]       21 47744036-47865682      + | ENSG00000160299        PCNT
##   [240]       21 47878812-47989926      + | ENSG00000160305       DIP2A
##   [241]       21 48018875-48025121      - | ENSG00000160307       S100B
##   [242]       21 48055079-48085036      + | ENSG00000160310       PRMT2
##             score
##         <numeric>
##     [1]        NA
##     [2]        NA
##     [3]        NA
##     [4]        NA
##     [5]        NA
##     ...       ...
##   [238]        NA
##   [239]        NA
##   [240]        NA
##   [241]        NA
##   [242]        NA
##   -------
##   seqinfo: 25 sequences (1 circular) from hg19 genome
##
## $CGI
## GRanges object with 365 ranges and 3 metadata columns:
##         seqnames            ranges strand |          ID        name     score
##            <Rle>         <IRanges>  <Rle> | <character> <character> <numeric>
##     [1]       21   9437273-9439473      * |       26635    CpG:_285        NA
##     [2]       21   9483486-9484663      * |       26636    CpG:_165        NA
##     [3]       21   9647867-9648116      * |       26637     CpG:_18        NA
##     [4]       21   9708936-9709231      * |       26638     CpG:_31        NA
##     [5]       21   9825443-9826296      * |       26639    CpG:_120        NA
##     ...      ...               ...    ... .         ...         ...       ...
##   [361]       21 48018543-48018791      * |       26995     CpG:_21        NA
##   [362]       21 48055200-48056060      * |       26996     CpG:_88        NA
##   [363]       21 48068518-48068808      * |       26997     CpG:_24        NA
##   [364]       21 48081242-48081849      * |       26998     CpG:_55        NA
##   [365]       21 48087201-48088106      * |       26999     CpG:_93        NA
##   -------
##   seqinfo: 25 sequences (1 circular) from hg19 genome
##
## $TFBS
## GRanges object with 48761 ranges and 3 metadata columns:
##           seqnames            ranges strand |           ID        name
##              <Rle>         <IRanges>  <Rle> |  <character> <character>
##       [1]       21 29884415-29884427      + |  GATA1.85108    GATA1_04
##       [2]       21 46923766-46923780      + |    CDP.81529      CDP_02
##       [3]       21   9491627-9491638      - |   HFH1.46541     HFH1_01
##       [4]       21   9491706-9491725      - |  PPARA.24892    PPARA_01
##       [5]       21   9491792-9491815      + |   GFI1.35413     GFI1_01
##       ...      ...               ...    ... .          ...         ...
##   [48757]       21 48083381-48083404      + | STAT5A.43326   STAT5A_02
##   [48758]       21 48083400-48083419      + |   ARNT.19751     ARNT_02
##   [48759]       21 48084826-48084841      + |   BRN2.40426     BRN2_01
##   [48760]       21 48084830-48084847      + | FOXJ2.121681    FOXJ2_01
##   [48761]       21 48084834-48084845      + |  NKX3A.47953    NKX3A_01
##              score
##          <numeric>
##       [1]       891
##       [2]       831
##       [3]       865
##       [4]       757
##       [5]       817
##       ...       ...
##   [48757]       751
##   [48758]       792
##   [48759]       803
##   [48760]       889
##   [48761]       851
##   -------
##   seqinfo: 25 sequences (1 circular) from hg19 genome
```

To find overlaps between your query and subject datasets we call `fOverlaps()` . Internally OGRE makes use of the `GenomicRanges` package to calculate full and partial overlap as schematically shown.



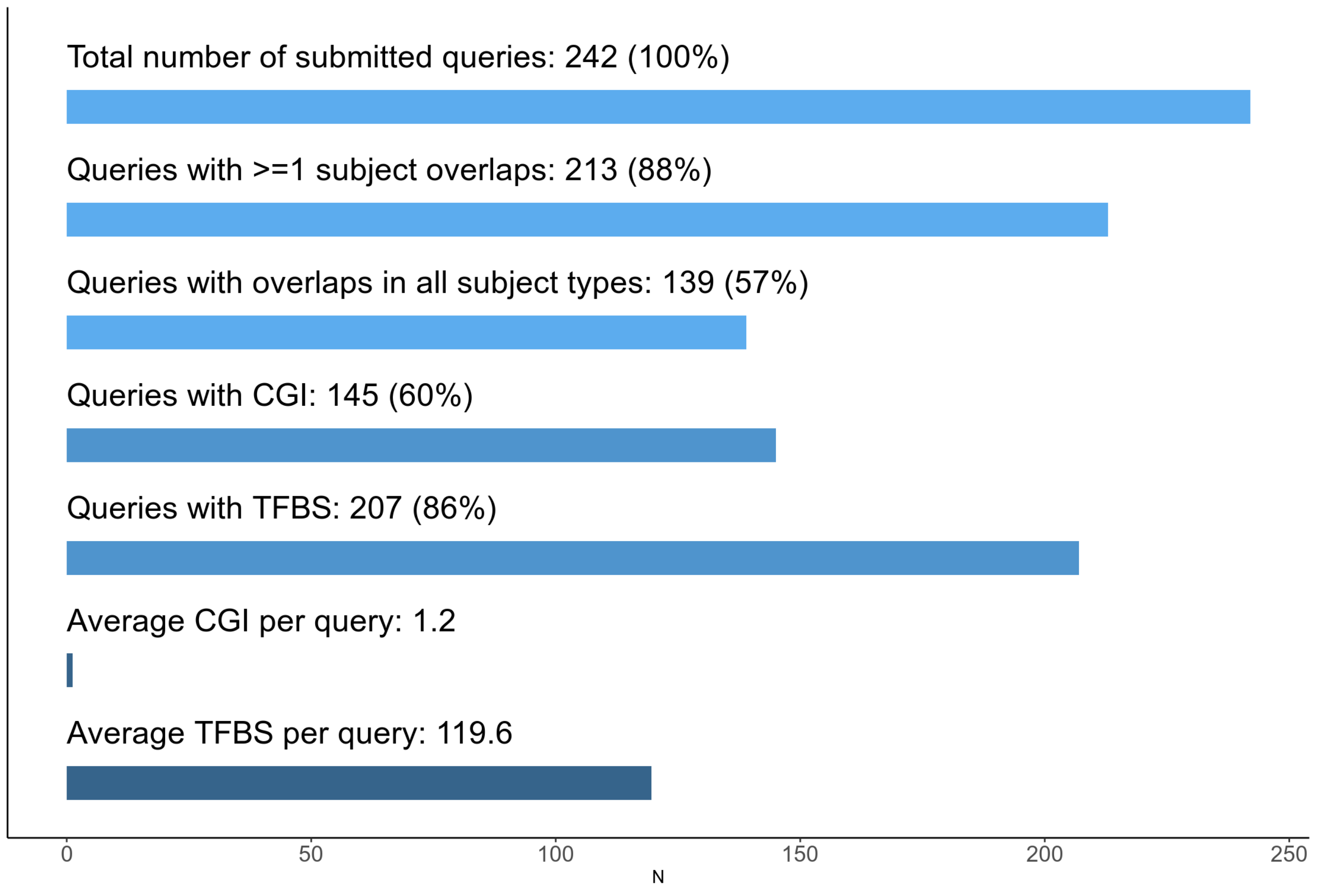
Any existing subject - query hits are then listed in `detailDT` and stored as a `data.table` .

```
myOGRE <- fOverlaps(myOGRE)
head(metadata(myOGRE)$detailDT,n=2)
```

```
##            queryID queryType subjID subjType queryChr queryStart queryEnd
## 1: ENSG00000166157     genes  26649      CGI       21   10906201 11029719
## 2: ENSG00000269011     genes  26654      CGI       21   15051621 15053459
##    queryStrand subjChr subjStart  subjEnd subjStrand overlapWidth overlapRatio
## 1:           -      21  10989914 10991413          *         1500   0.01214388
## 2:           -      21  15052411 15052644          *          234   0.12724307
```

The summary plot provides us with useful information about the number of overlaps between your datasets.

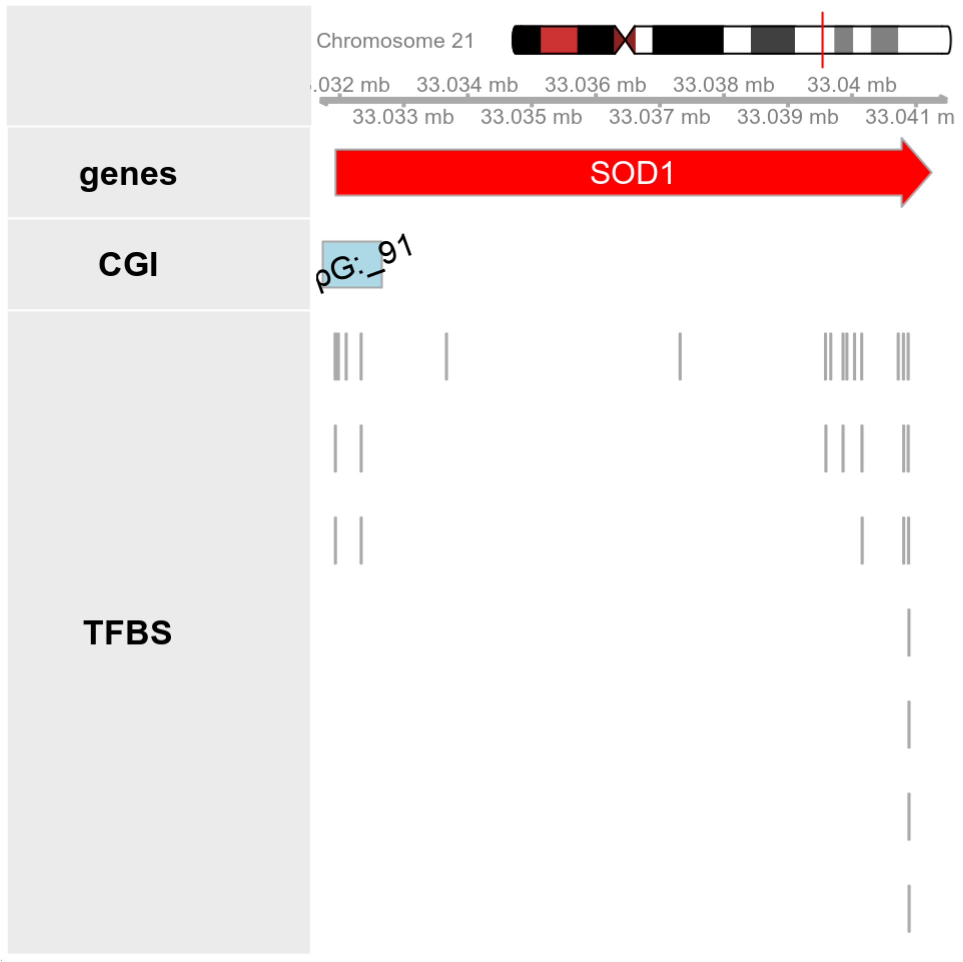
```
myOGRE <- sumPlot(myOGRE)
metadata(myOGRE)$barplot_summary
```



Using the `Gviz` visualization each query can be displayed with all overlapping subject elements. Choose labels for all region tracks by supplying a `trackRegionLabels` vector. Plots are stored in the same location as your dataset files.

```
myOGRE <- gvizPlot(myOGRE,"ENSG00000142168",showPlot = TRUE,
                   trackRegionLabels = setNames(c("name","name"),c("genes","CGI")))
```

```
## Plotting query: ENSG00000142168
```



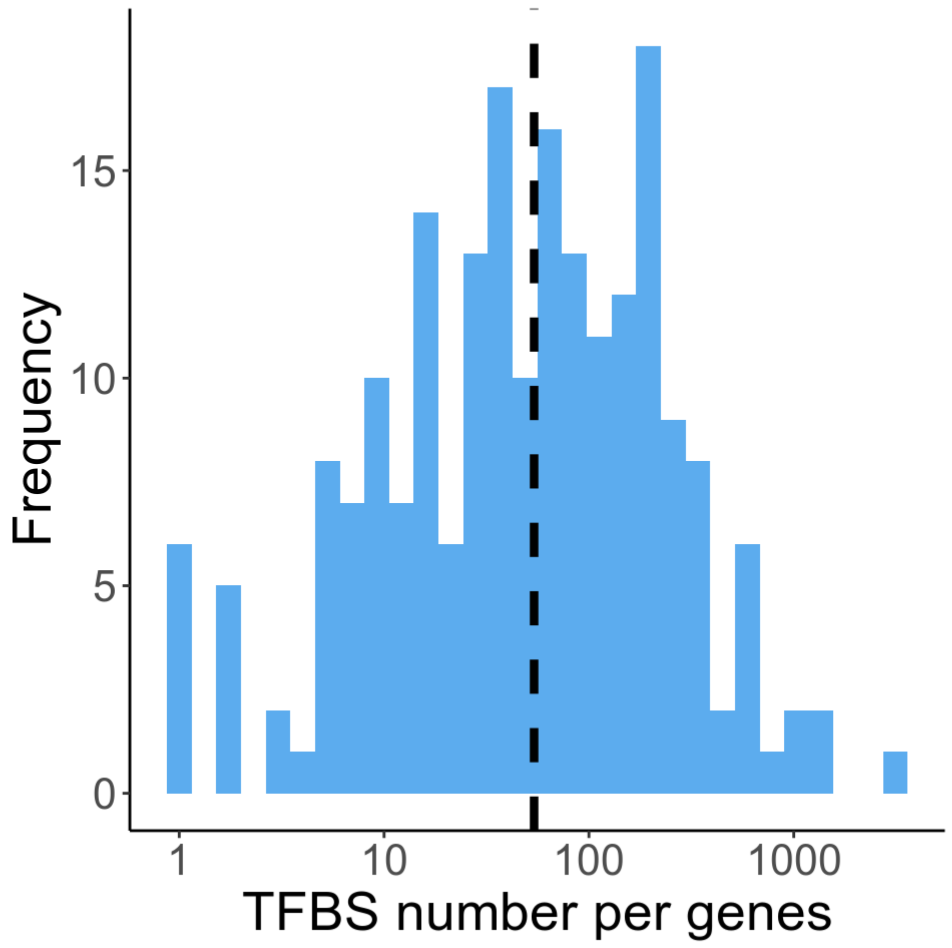
The overlap distribution can be generated with `summarizeOverlap(myOGRE)` and outputs a table with informative statistics such as minimum, lower quantile, mean, median, upper quantile, and maximum number of overlaps per region and per dataset. Overlap distribution can also be displayed as histograms using `plotHist(myOGRE)` and accessed by `metadata(myOGRE)$hist` and `metadata(myOGRE)$summaryDT`. Two tables / plots are generated. The first one showing numbers for regions with and without overlap and the second one showing numbers only for regions with overlap by excluding all others. Next, we generate an histogram with the number of TFBS per gene (x-axis, log scale) and the TFBS frequency (y-axis). When focusing only on regions with overlap, we see that genes have on average (median) 54 TFBS overlaps (black dashed line).

```
myOGRE <- summarizeOverlap(myOGRE)
myOGRE <- plotHist(myOGRE)
metadata(myOGRE)$summaryDT
```

```
## $includes0
##               CGI      TFBS
## Min.     0.000000    0.0000
## 1st Qu.  0.000000    8.0000
## Median   1.000000   36.0000
## Mean     1.210744  119.6116
## 3rd Qu.  1.750000  129.7500
## Max.    14.000000 3136.0000
##
## $excludes0
##              CGI      TFBS
## Min.     1.00000    1.0000
## 1st Qu.  1.00000   15.0000
## Median   1.00000   54.0000
## Mean     2.02069  139.8357
## 3rd Qu.  2.00000  159.5000
## Max.    14.00000 3136.0000
## NA's    97.00000   35.0000
```

```
metadata(myOGRE)$hist$TFBS
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



It is possible to create an average coverage profile of all gene-TFBS overlaps, split in 100 bins, which represent gene bodies of all 242 genes. Both, forward and reverse coding genes are arranged on the x-Axis and peaks indicate an TFBS overlap enrichment. Overlap coverage is calculated as the sum of all gene TFBS overlaps in 5'-3'direction. Generated plots can be accessed by `metadata(myOGRE)$covPlot$TFBS` and the resulting profile shows an accumulation of TFBS around gene start and end positions.

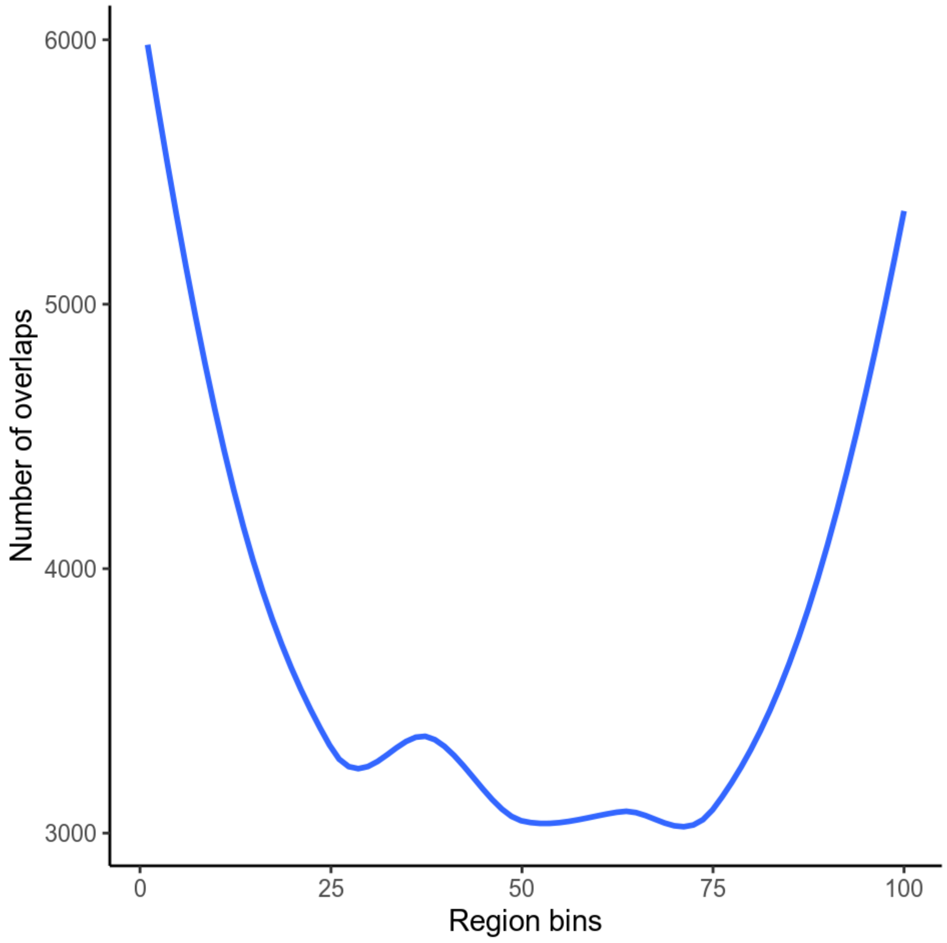
```
myOGRE <- covPlot(myOGRE)
```

```
## Generating coverage plot(s), this might take a while...
```

```
## Excluding regions with nucleotides<nbin
```

```
metadata(myOGRE)$covPlot$TFBS$plot
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Quick start- load datasets from AnnotationHub

AnnotationHub offers a wide range of annotated datasets which can be manually aquired but need some parsing to work with OGRE as detailed in vignette section Frequently Asked Questions(FAQ). For convenience `addDataSetFromHub()` adds one of the predefined human datasets of `listPredefinedDataSets()` to an OGREDataSet. Those are taken from AnnotationHub and are ready to use for OGRE. We start by creating an empty OGREDataSet and attaching one dataset after another, whereby one query and two subjects are added. The datasets are now ready for further analysis.

```
myOGRE <- OGREDataSet()
listPredefinedDataSets()
myOGRE <- addDataSetFromHub(myOGRE,"protCodingGenes","query")
myOGRE <- addDataSetFromHub(myOGRE,"CGI","subject")
myOGRE <- addDataSetFromHub(myOGRE,"TFBS","subject")
names(myOGRE)
```

As you can see, the three datasets proteinCodingGenes, CGI and TFBS are stored within OGRE. You can then continue with overlap analysis using `fOverlaps()` .

Quick start- load user defined GenomicRanges (GRanges) datasets

To offer more flexibility `addGRanges()` enables the user to attach additional datasets to OGRE in form of GenomicRanges objects. Again we start by creating an empty OGREDataSet and generate an example GenomicRanges object which is then added to your OGREDataSet either as “query” or “subject”.

```
myOGRE <- OGREDataSet()
myGRanges <- makeExampleGRanges()
myOGRE <- addGRanges(myOGRE,myGRanges,"query")
```

Frequently asked questions

How to add additional datasets from AnnotationHub?

Use `AnnotationHub()` to connect to AnnotationHub. Each dataset is stored under a unique ID and can be accessed in a list like fashion i.e. `aH[["AH5086"]]` . Queries like `c("GRanges","Homo sapiens", "CpG")` enable browsing through datasets. In this case we are searching for human CpG islands ranges stored as GenomicRanges objects. For more information refer to `?AnnotationHub` To make those datasets compatible with OGRE additional parsing is needed as stated in How to add custom GenomicRanges datasets?

```
aH <- AnnotationHub()
aH[["AH5086"]]
q <- query(aH, c("GRanges","Homo sapiens", "CpG"))
```

How to add custom GenomicRanges datasets?

Any GenomicRanges datasets can be added that fulfill basic compatibility requirements. GenomicRanges objects must:

- Originate from a common genome build i.e. “HG19”

Use `GenomeInfoDb::genome()` on any GenomicRanges object to get/set genome information

- Contain the same set of chromosomes i.e. chr1 != 1 or chrM != MT

Use `GenomeInfoDb::seqinfo()` on any GenomicRanges object to get/set chromosome information

- Contain a “name” and a (unique) “ID” column

Use `S4Vectors::mcols()` on any GenomicRanges object to get/set metadata information

How to add datasets stored as .gff files?

Datasets from external sources are often stored as .gff (v2&v3) files. Once those files exist in the query or subject folder and their attribute columns contain “ID” and “name” information, OGRE tries to load them. Working example .gff files can be found on OGRE’s github page (https://github.com/svenbioinf/OGRE) in folder: inst/extdata/gffTest.

```
myOGRE <- OGREDataSetFromDir(queryFolder = "pathToQueryFolder",
                             subjectFolder = "pathToSubjectFolder")
myOGRE <- loadAnnotations(myOGRE)
```

How to add datasets stored as tabular files?

Datasets stored as tabular files like .csv or .bed may need some preprocessing for them work with OGRE. We recommend reading them in with `read.table()` or `data.table::fread()` to obtain a data frame object. After making sure the dataset complies with the requirements in section How to add custom GenomicRanges datasets?, `GenomicRanges::makeGRangesFromDataFrame()` offers a convenient way to generate GenomicRanges object from data frames.

What type of overlaps are reported?

Both, partial overlap, where only a part of two (or more) regions are overlapping and complete overlap, where one region is completely overlapped by another, are reported.

How to change dataset names?

OGRE automatically infers dataset names based on your file names. You can either change your file names before you start OGRE or you can use `names(myOGRE) <- c("NewName1", "NewName2","...")` after you read in your datasets.

Session info

```
sessionInfo()
```

```
## R version 4.2.0 (2022-04-22)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Linux Mint 19
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## 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=de_DE.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=de_DE.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=de_DE.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] GenomeInfoDb_1.32.4 IRanges_2.30.1      OGRE_0.99.8
## [4] S4Vectors_0.34.0    BiocGenerics_0.42.0
##
## loaded via a namespace (and not attached):
##   [1] backports_1.4.1               Hmisc_4.7-1
##   [3] AnnotationHub_3.4.0           systemfonts_1.0.4
##   [5] BiocFileCache_2.4.0           lazyeval_0.2.2
##   [7] shinydashboard_0.7.2          splines_4.2.0
##   [9] BiocParallel_1.30.3           ggplot2_3.3.6
##  [11] digest_0.6.29                 ensembldb_2.20.2
##  [13] htmltools_0.5.3               fansi_1.0.3
##  [15] magrittr_2.0.3                checkmate_2.1.0
##  [17] memoise_2.0.1                 BSgenome_1.64.0
##  [19] cluster_2.1.3                 shinyFiles_0.9.3
##  [21] Biostrings_2.64.1             matrixStats_0.62.0
##  [23] prettyunits_1.1.1             jpeg_0.1-9
##  [25] colorspace_2.0-3              blob_1.2.3
##  [27] rappdirs_0.3.3                textshaping_0.3.6
##  [29] xfun_0.33                     dplyr_1.0.10
##  [31] crayon_1.5.1                  RCurl_1.98-1.8
##  [33] jsonlite_1.8.0                survival_3.2-13
##  [35] VariantAnnotation_1.42.1      glue_1.6.2
##  [37] gtable_0.3.1                  zlibbioc_1.42.0
##  [39] XVector_0.36.0                DelayedArray_0.22.0
##  [41] scales_1.2.1                  DBI_1.1.3
##  [43] Rcpp_1.0.9                    xtable_1.8-4
##  [45] progress_1.2.2                htmlTable_2.4.1
##  [47] foreign_0.8-82                bit_4.0.4
##  [49] Formula_1.2-4                 DT_0.25
##  [51] htmlwidgets_1.5.4             httr_1.4.4
##  [53] RColorBrewer_1.1-3            ellipsis_0.3.2
##  [55] farver_2.1.1                  pkgconfig_2.0.3
##  [57] XML_3.99-0.10                 Gviz_1.40.1
##  [59] nnet_7.3-18                   sass_0.4.2
##  [61] dbplyr_2.2.1                  deldir_1.0-6
##  [63] utf8_1.2.2                    labeling_0.4.2
##  [65] tidyselect_1.1.2              rlang_1.0.5
##  [67] later_1.3.0                   AnnotationDbi_1.58.0
##  [69] munsell_0.5.0                 BiocVersion_3.15.2
##  [71] tools_4.2.0                   cachem_1.0.6
##  [73] cli_3.4.0                     generics_0.1.3
##  [75] RSQLite_2.2.17                evaluate_0.16
##  [77] shinyBS_0.61.1                stringr_1.4.1
##  [79] fastmap_1.1.0                 yaml_2.3.5
##  [81] ragg_1.2.4                    knitr_1.40
##  [83] bit64_4.0.5                   fs_1.5.2
##  [85] purrr_0.3.4                   KEGGREST_1.36.3
##  [87] AnnotationFilter_1.20.0       nlme_3.1-157
##  [89] mime_0.12                     xml2_1.3.3
##  [91] biomaRt_2.52.0                compiler_4.2.0
##  [93] rstudioapi_0.14               filelock_1.0.2
##  [95] curl_4.3.2                    png_0.1-7
##  [97] interactiveDisplayBase_1.34.0 tibble_3.1.8
##  [99] bslib_0.4.0                   stringi_1.7.8
## [101] highr_0.9                     GenomicFeatures_1.48.3
## [103] lattice_0.20-45               ProtGenerics_1.28.0
## [105] Matrix_1.4-1                  vctrs_0.4.1
## [107] pillar_1.8.1                  lifecycle_1.0.2
## [109] BiocManager_1.30.18           jquerylib_0.1.4
## [111] data.table_1.14.2             bitops_1.0-7
## [113] httpuv_1.6.6                  rtracklayer_1.56.1
## [115] GenomicRanges_1.48.0          R6_2.5.1
## [117] BiocIO_1.6.0                  latticeExtra_0.6-30
## [119] promises_1.2.0.1              gridExtra_2.3
## [121] codetools_0.2-18              dichromat_2.0-0.1
## [123] assertthat_0.2.1              SummarizedExperiment_1.26.1
## [125] rjson_0.2.21                  GenomicAlignments_1.32.1
## [127] Rsamtools_2.12.0              GenomeInfoDbData_1.2.8
## [129] mgcv_1.8-40                   parallel_4.2.0
## [131] hms_1.1.2                     grid_4.2.0
## [133] rpart_4.1.16                  tidyr_1.2.1
## [135] rmarkdown_2.17                MatrixGenerics_1.8.1
## [137] biovizBase_1.44.0             Biobase_2.56.0
## [139] shiny_1.7.2                   base64enc_0.1-3
## [141] interp_1.1-3                  restfulr_0.0.15
```