Package 'ondisc'

June 17, 2026

Title Algorithms and Data Structures for Large Single-Cell Expression Matrices

Version 1.3.5

Depends R (>= 4.1.0)

Description Single-cell datasets are growing in size, posing challenges as well as opportunities for genomics researchers. 'ondisc' is an R package that facilitates analysis of large-scale single-cell data out-of-core on a laptop or distributed across tens to hundreds of processors on a cluster or cloud. In both of these settings, 'ondisc' requires only a few gigabytes of memory, even if the input data are tens of gigabytes in size. 'ondisc' mainly is oriented toward single-cell CRISPR screen analysis, but also can be used for single-cell differential expression and single-cell co-expression analyses. 'ondisc' is powered by several new, efficient algorithms for manipulating and querying large, sparse expression matrices.

License MIT + file LICENSE

Encoding UTF-8

LinkingTo Rcpp, Rhdf5lib

biocViews DataImport, SingleCell, DifferentialExpression, CRISPR

RoxygenNote 7.3.2

Imports crayon, data.table, methods, Rcpp, Rhdf5lib, dplyr, readr, Matrix

SystemRequirements GNU make

Suggests sessioninfo, knitr, R.utils, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://timothy-barry.github.io/ondisc/, https://timothy-barry.github.io/sceptre-book/

BugReports https://github.com/timothy-barry/ondisc/issues

VignetteBuilder knitr

NeedsCompilation yes

Author Timothy Barry [aut, cre] (ORCID:
<https://orcid.org/0000-0002-4356-627X>),
Songcheng Dai [ctb],
Yixuan Qiu [ctb],
Eugene Katsevich [aut, ths]

Maintainer Timothy Barry <tbarry@hsph.harvard.edu>

Repository CRAN

Date/Publication 2026-06-17 13:30:02 UTC

Contents

`ondisc-package` *ondisc: Algorithms and Data Structures for Large Single-Cell Expression Matrices*

Description

Single-cell datasets are growing in size, posing challenges as well as opportunities for genomics researchers. 'ondisc' is an R package that facilitates analysis of large-scale single-cell data out-of-core on a laptop or distributed across tens to hundreds of processors on a cluster or cloud. In both of these settings, 'ondisc' requires only a few gigabytes of memory, even if the input data are tens of gigabytes in size. 'ondisc' mainly is oriented toward single-cell CRISPR screen analysis, but also can be used for single-cell differential expression and single-cell co-expression analyses. 'ondisc' is powered by several new, efficient algorithms for manipulating and querying large, sparse expression matrices. See Barry et al. (2024) doi:10.1186/s13059024032542.

Author(s)

Maintainer: Timothy Barry <tbarry@hsph.harvard.edu> (ORCID)

Authors:

- Eugene Katsevich <ekatsevi@wharton.upenn.edu> [thesis advisor]

Other contributors:

- Songcheng Dai [contributor]
- Yixuan Qiu [contributor]

See Also

Useful links:

- https://timothy-barry.github.io/ondisc/
- https://timothy-barry.github.io/sceptre-book/
- Report bugs at https://github.com/timothy-barry/ondisc/issues

Examples

```
# initialize odm objects from Cell Ranger output; also, compute the cellwise covariates
directories_to_load <- file.path(
 system.file("extdata", "highmoi_example", package = "ondisc"),
 paste0("gem_group_", c(1, 2))
)
directory_to_write <- tempdir()
# Set data.table threads to 1 to pass CRAN example timing checks.
old_threads <- data.table::setDTthreads(1L)
out_list <- create_odm_from_cellranger(
  directories_to_load = directories_to_load,
  directory_to_write = directory_to_write,
)
data.table::setDTthreads(old_threads)

# extract the odm corresponding to the gene modality
gene_odm <- out_list$gene
gene_odm

# obtain dimension information
dim(gene_odm)
nrow(gene_odm)
ncol(gene_odm)

# obtain rownames (i.e., the feature IDs)
rownames(gene_odm) |> head()

# extract row into memory, first by integer and then by string
expression_vector_1 <- gene_odm[10,]
expression_vector_2 <- gene_odm[rownames(gene_odm)[10],]

# delete the gene_odm object
rm(gene_odm)

# reinitialize the gene_odm object
gene_odm <- initialize_odm_from_backing_file(
  paste0(tempdir(), "/gene.odm")
)
gene_odm
```

create_odm_from_cellranger *Create* odm *object from Cell Ranger*

Description

`create_odm_from_cellranger()` initializes an `odm` object, taking as input the output of one or more calls to Cell Ranger count. The number of `odm` objects returned corresponds to the number of modalities in the input data. Additionally, the cell-wise covariate data frame is computed and returned. `create_odm_from_cellranger()` supports the Cell Ranger modalities "Gene Expression", "CRISPR Guide Capture", and "Antibody Capture".

Usage

```
create_odm_from_cellranger(
  directories_to_load,
  directory_to_write,
  write_cellwise_covariates = TRUE,
  chunk_size = 1000L,
  compression_level = 3L,
  grna_target_data_frame = NULL
)
```

Arguments

`directories_to_load`
: a character vector specifying the locations of the directories to load. Each directory should contain the files "matrix.mtx.gz" and "features.tsv.gz" (and optionally "barcodes.tsv.gz", which is ignored).

`directory_to_write`
: a string indicating the directory in which to write the backing .odm files.

`write_cellwise_covariates`
: (optional; default `TRUE`) a logical value indicating whether to write the cellwise covariate data frame to disk (`TRUE`) in addition to returning it from `create_odm_from_cellranger()`.

`chunk_size`
: (optional; default `1000L`) a positive integer specifying the chunk size to use to store the data in the backing HDF5 file.

`compression_level`
: (optional; default `3L`) an integer in the inveral [0, 9] specifying the compression level to use to store the data in the backing HDF5 file.

`grna_target_data_frame`
: (optional) a data frame mapping each gRNA ID to its target. Relevant only if the CRISPR modality is present within the data. (See note.)

Value

a list containing the odm object(s) and cellwise covariates.

Note

The grna_target_data_frame is relevant only for CRISPR screen data (i.e., data for which the "CRISPR Guide Capture" modality is present). In single-cell CRISPR screens, gRNAs are delivered to cells via a viral vector. Some recent single-cell CRISPR screens involve a special design in which each viral vector harbors multiple gRNAs. For example, Replogle et al. (2022, doi:10.1016/j.cell.2022.05.013) conducted a screen in which each viral vector contained two gRNAs, each targeting the same site. In such screens, users may wish to "collapse" the gRNA count matrix by summing over the UMI counts of gRNAs contained on the same vector. To do so, users can pass the argument grna_target_data_frame, which is a data frame containing two columns: grna_id and vector_id. grna_id should coincide with the gRNA IDs as contained within the features.tsv file, and vector_id should be a string indicating the vector to which a given gRNA ID belongs. The expression vectors of gRNAs contained within the same vector are summed.

The arguments chunk_size and compression_level control the extent to which the backing .odm files are compressed, with higher values corresponding to smaller file sizes (albeit possibly longer read and write times).

Examples

```
directories_to_load <- file.path(
 system.file("extdata", "highmoi_example", package = "ondisc"),
 paste0("gem_group_", c(1, 2))
)
directory_to_write <- tempdir()
# Set data.table threads to 1 to pass CRAN example timing checks.
old_threads <- data.table::setDTthreads(1L)
out_list <- create_odm_from_cellranger(
  directories_to_load = directories_to_load,
  directory_to_write = directory_to_write,
)
data.table::setDTthreads(old_threads)
```

---

create_odm_from_r_matrix

Create odm *object from R matrix*

Description

create_odm_from_r_matrix() converts an R matrix (stored in standard dense format or sparse format) into an odm object.

Usage

```
create_odm_from_r_matrix(
  mat,
  file_to_write,
  chunk_size = 1000L,
  compression_level = 3L
)
```

Arguments

mat	an R matrix of class "matrix", "dgCMatrix", "dgRMatrix", or "dgTMatrix".
file_to_write	a fully-qualified file path specifying the location in which to write the backing .odm file.
chunk_size	(optional; default 1000L) a positive integer specifying the chunk size to use to store the data in the backing HDF5 file.
compression_level	(optional; default 3L) an integer in the inveral [0, 9] specifying the compression level to use to store the data in the backing HDF5 file.

Value

an odm object

Examples

```
set.seed(4)
x <- rpois(100, lambda = 1)
gene_matrix <- matrix(
  x,
  nrow = 5L,
  dimnames = list(paste0("gene_", seq_len(5L)), paste0("cell_", seq_len(20L)))
)
file_to_write <- paste0(tempdir(), "/gene.odm")
odm_object <- create_odm_from_r_matrix(
  mat = gene_matrix,
  file_to_write = file_to_write,
  chunk_size = 5L
)
```

---

dim,odm-method *Return the number of columns and rows of an* odm *object*

Description

ncol() and nrow() return the number of rows (i.e., features) and columns (i.e., cells), respectively, contained within an odm object. dim() returns an integer vector of length two whose first and second entries, respectively, indicate the number of rows and columns in an odm object.

Usage

```
## S4 method for signature 'odm'
dim(x)
```

Arguments

x	an object of class odm

Value

the dimension of the odm object

Examples

```
directories_to_load <- file.path(
 system.file("extdata", "highmoi_example", package = "ondisc"),
 paste0("gem_group_", c(1, 2))
)
directory_to_write <- tempdir()
# Set data.table threads to 1 to pass CRAN example timing checks.
old_threads <- data.table::setDTthreads(1L)
out_list <- create_odm_from_cellranger(
  directories_to_load = directories_to_load,
  directory_to_write = directory_to_write,
)
data.table::setDTthreads(old_threads)
gene_odm <- out_list$gene
# return the dimension, number of rows, and number of columns
dim(gene_odm)
nrow(gene_odm)
ncol(gene_odm)
```

---

dimnames,odm-method *Return the rownames of an* odm *object*

Description

rownames() returns the rownames (i.e., feature IDs) of an odm object.

Usage

```
## S4 method for signature 'odm'
dimnames(x)
```

Arguments

x	an object of class odm

Value

the rownames of an odm object

Examples

```
directories_to_load <- file.path(
 system.file("extdata", "highmoi_example", package = "ondisc"),
 paste0("gem_group_", c(1, 2))
)
directory_to_write <- tempdir()
# Set data.table threads to 1 to pass CRAN example timing checks.
old_threads <- data.table::setDTthreads(1L)
out_list <- create_odm_from_cellranger(
  directories_to_load = directories_to_load,
  directory_to_write = directory_to_write,
)
data.table::setDTthreads(old_threads)
gene_odm <- out_list$gene
# return the rownames
rownames(gene_odm) |> head()
```

---

initialize_odm_from_backing_file

Initialize an odm *object*

Description

`initialize_odm_from_backing_file()` initializes an odm object from a backing `.odm` file.

Usage

```
initialize_odm_from_backing_file(odm_file)
```

Arguments

`odm_file`	file path to a backing `.odm` file.

Details

`initialize_odm_from_backing_file()` is portable: the user can create an `.odm` file (via `create_odm_from_cellranger()`
or `create_odm_from_r_matrix()`) on one computer, transfer the `.odm` file to another computer,
and then load the `.odm` file (via `initialize_odm_from_backing_file()`) on the second computer.

Value

an odm object

Examples

```
directories_to_load <- file.path(
 system.file("extdata", "highmoi_example", package = "ondisc"),
 paste0("gem_group_", c(1, 2))
)
directory_to_write <- tempdir()
# Set data.table threads to 1 to pass CRAN example timing checks.
old_threads <- data.table::setDTthreads(1L)
out_list <- create_odm_from_cellranger(
  directories_to_load = directories_to_load,
  directory_to_write = directory_to_write,
)
data.table::setDTthreads(old_threads)
gene_odm <- out_list$gene
gene_odm

# delete the gene_odm object
rm(gene_odm)

# reinitialize the gene_odm object
gene_odm <- initialize_odm_from_backing_file(
  paste0(tempdir(), "/gene.odm")
)
gene_odm
```

`write_example_cellranger_dataset`
Write example Cell Ranger dataset

Description

`write_example_cellranger_dataset()` creates an example dataset and writes the dataset to disk in Cell Ranger format. The dataset can be unimodal or multimodal, containing some subset of the gene expression, gRNA expression, and protein expression modalities.

Usage

```
write_example_cellranger_dataset(
  n_features,
  n_cells,
  n_batch,
  modalities,
  directory_to_write,
  p_zero = NULL,
  p_set_col_zero = NULL,
  p_set_row_zero = NULL
)
```

Arguments

n_features	an integer vector specifying the number of features to simulate for each modality.
n_cells	an integer specifying the number of cells to simulate.
n_batch	an integer specifying the number of batches to simulate. Cells from different batches are written to different directories.
modalities	a character vector indicating the modalities to simulate. The vector should contain some subset of the strings "gene", "grna", and "protein".
directory_to_write	the directory in which to write the Cell Ranger feature barcode files.
p_zero	(optional; default random) the fraction of entries to set randomly to zero.
p_set_col_zero	(optional; default random) the fraction of columns to set randomly to zero.
p_set_row_zero	(optional; default random) the fraction of rows to set randomly to zero.

Value

a list containing (i) a list of the simulated expression matrices, (ii) a character vector containing the names of the genes (or NULL if the gene modality was not simulated), and (iii) a character vector specifying the batch of each cell.

Examples

```
set.seed(4)
n_features <- c(30, 8, 6)
modalities <- c("gene", "protein", "grna")
n_cells <- 60
n_batch <- 2
directory_to_write <- tempdir()
p_set_col_zero <- 0
out <- write_example_cellranger_dataset(
  n_features = n_features,
  n_cells = n_cells,
  n_batch = n_batch,
  modalities = modalities,
  directory_to_write = directory_to_write,
  p_set_col_zero = p_set_col_zero
)

# directories written to directory_to_write
fs <- list.files(directory_to_write, pattern = "batch*", full.names = TRUE)
# files contained within the directories
list.files(fs[1])
list.files(fs[2])
```

`[,odm,ANY,missing,missing-method`

Load a row of an `odm` *object into memory*

Description

The operator `[` loads a specified row of an `odm` object into memory. The `odm` object can be indexed either by integer or feature ID.

Usage

```
## S4 method for signature 'odm,ANY,missing,missing'
x[i, j, drop]
```

Arguments

`x`	an object of class `odm`
`i`	the index of the row to load into memory. `i` can be either an integer (specifying the integer-based index of the row to load into memory) or a string (specifying the feature ID of the row to load into memory).
`j`	not used
`drop`	not used

Value

an expression vector (of class `numeric`)

Examples

```
directories_to_load <- file.path(
 system.file("extdata", "highmoi_example", package = "ondisc"),
 paste0("gem_group_", c(1, 2))
)
directory_to_write <- tempdir()
# Set data.table threads to 1 to pass CRAN example timing checks.
old_threads <- data.table::setDTthreads(1L)
out_list <- create_odm_from_cellranger(
  directories_to_load = directories_to_load,
  directory_to_write = directory_to_write,
)
data.table::setDTthreads(old_threads)
gene_odm <- out_list$gene
# extract rows into memory by index and ID
v1 <- gene_odm[10L,]
v2 <- gene_odm[rownames(gene_odm)[10],]
```

Index