Package 'wpm'

November 12, 2026

Type Package

Title Well Plate Maker

Version 1.22.0

Description The Well-Plate Maker (WPM) is a shiny application deployed as an R package. Functions for a command-line/script use are also available. The WPM allows users to generate well plate maps to carry out their experiments while improving the handling of batch effects. In particular, it helps controlling the ``plate effect'' thanks to its ability to randomize samples over multiple well plates. The algorithm for placing the samples is inspired by the backtracking algorithm: the samples are placed at random while respecting specific spatial constraints.

License Artistic-2.0

biocViews GUI, Proteomics, MassSpectrometry, BatchEffect, ExperimentalDesign

Depends R (>= 4.1.0)

Imports utils, methods, cli, Biobase, SummarizedExperiment, config, golem, shiny, DT, ggplot2, dplyr, rlang, stringr, shinydashboard, shinyWidgets, shinycustomloader, RColorBrewer, logging

Encoding UTF-8

LazyData false

RoxygenNote 7.1.1

Suggests MSnbase, testthat, BiocStyle, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

URL https://github.com/HelBor/wpm, https://bioconductor.org/packages/release/bioc/html/wpm.html

BugReports https://github.com/HelBor/wpm/issues

git_url https://git.bioconductor.org/packages/wpm

git_branch RELEASE_3_23

git_last_commit c37c2e9

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Helene Borges [aut, cre],
Thomas Burger [aut]

Maintainer Helene Borges <borges.helene.sophie@gmail.com>

Contents

backtracking 2
balancedGrpDistrib 3
checkConstraints 4
checkWpmInputs 5
convertCSV 5
convertESet 6
convertSE 7
convertVector2Df 8
data_test 8
defineBufferCoords 9
drawMap 10
findNEWSneighbors 11
findNSneighbors 11
findWEneighbors 12
generateMap 12
joinDataframes 13
randomWalk 14
resample 15
solveCell 15
wpm 16
wrapperWPM 16

Index **18**

`backtracking` *Backtracking Function*

Description

Function used to launch the backtracking algorithm on a dataframe with the corresponding plate parameters, number of iterations and special wells

Usage

```
backtracking(
  max_iter = 20,
  user_data,
  wells,
  rows,
  columns,
  nb_plates,
  constraint,
  prog = NULL
)
```

Arguments

max_iter	numeric, the maximal number of iterations to do, default value is 20
user_data	dataframe, user samples to place randomly on the plate
wells	dataframe, special wells not to be placed randomly on the plate
rows	numeric, number of lines on the plate(s)
columns	numeric, number of columns on the plate(s)
nb_plates	numeric, number of plates
constraint	character, spatial mode
prog	progress bar used for shiny app only

Value

a dataframe containing user samples and special wells with their coordinates for the corresponding plates.

---

balancedGrpDistrib *Makes a balanced distribution of the elements between several plates.*

Description

This function makes it possible to distribute the samples equitably on several plates, taking into account the numbers in the groups (if there are any). This means that, for example, if 2 plates are to be filled, then 50 generally, all the plates are assigned the same number of elements. When the numbers do not allow it (in particular when the total number of elements to be allocated is not a multiple of the number of plates), there will be a slight difference in the number of samples on the plates.

Usage

```
balancedGrpDistrib(d, nb_p, df_max_size)
```

Arguments

d	the user dataframe
nb_p	the number of plates to fill
df_max_size	the maximum number of samples that can be placed on the current plate

Value

a list of dataframes each corresponding to a plate to fill.

---

checkConstraints *Check for spatial constraints*

Description

Finds the neighbors of the current element (row, col) in the matrix m, depending on the chosen constraint pattern. Currently, there are only 3 valid patterns (NS, WE and NEWS)

Usage

```
checkConstraints(m, row, col, mode)
```

Arguments

m	matrix
row	current selected row in the matrix m
col	current selected column in the matrix m
mode	spatial constraint

Value

A vector containing the neighbors of element (row,col) of the matrix m.

checkWpmInputs *Check the inputs for the wrapper function*

Description

Checks if all the inputs given to the function WrapperWPM are correct and intercompatible.

Usage

```
checkWpmInputs(
  user_df,
  plate_dims,
  nb_plates,
  spatial_constraint,
  max_iteration
)
```

Arguments

user_df	expected dataframe, returns adapted message error
plate_dims	expected list of plate dimensions (rows and columns)
nb_plates	expected number of plates
spatial_constraint	expected character for spatial constraint
max_iteration	expected number of iterations

Value

returns an error message if a problem is found with some parameter.

convertCSV *Convert a CSV File into a valid dataframe for WPM*

Description

This function converts a CSV into a dataframe to make it usable by the shiny application of wpm as well as by the wrapper function (version of wpm in command line). Be sure that the first column of the CSV file corresponds to samples names.

Usage

```
convertCSV(dt_path, row_names = FALSE, gp_field = NULL, ...)
```

Arguments

dt_path	file path.
row_names	logical value, indicates if the file has rownames or not.
gp_field	the column name indicating the grouping factor for the samples in the csv. If there is no grouping factor, then gp_field must be set to NULL or "none".
...	parameters to give to read.csv2 function

Value

a list containing a dataframe containing the data of the imported CSV and a dataframe containing 3 fields (Sample, Group and ID) which will be used by WPM. Or NULL if there is an error when giving wrong parameters.

Examples

```
test <- data.frame("Sample" = c("s1","s2","s3","s4"),
                   "Group" = c("A","A","B", "C"))
tf <- tempfile()
write.csv2(test, tf, row.names = FALSE)
convertCSV(tf, gp_field = "Group", header = TRUE, sep = ";")

# if there are row names in the CSV file
write.csv2(test, tf)
convertCSV(tf, row_names = TRUE, gp_field="Group", header = TRUE, sep = ";")

# if there is no grouping factor in the CSV file
convertCSV(tf, row_names = TRUE, gp_field ="none", header = TRUE, sep = ";")
# gives the same output as the previous example
convertCSV(tf, row_names = TRUE, header = TRUE, sep = ";")
```

---

convertESet *Convert the phenotype data of an ExpressionSet or MsnSet into a dataframe for WPM*

Description

This function converts an ExpressionSet/MsnSet object into a dataframe to make it usable by the shiny application of wpm as well as by the wrapper function (version of wpm in command line)

Usage

```
convertESet(eSet_obj, gp_field = NULL)
```

Arguments

eSet_obj	an ExpressionSet/MsnSet object contaning the phenotype data
gp_field	character, corresponding to the phenotype data used to categorize samples into distinct groups if any

Value

a dataframe containing 3 fields: Sample, Group and ID.

Examples

```
sample_names <- c("s1","s2","s3","s4", "s5")
M <- matrix(NA, nrow = 4, ncol = 5)
colnames(M) <- sample_names
rownames(M) <- paste0("id", LETTERS[1:4])
pd <- data.frame(Environment = rep_len(LETTERS[1:3], 5),
                 Category = rep_len(1:2, 5), row.names = sample_names)
rownames(pd) <- colnames(M)
x <- MSnbase::MSnSet(exprs = M,pData =  pd)
convertESet(x, "Environment")
```

`convertSE` *Convert the phenotype data of a SummarizedExperiment into a dataframe for WPM*

Description

Convert the phenotype data of a SummarizedExperiment into a dataframe for WPM

Usage

```
convertSE(se_object, gp_field = NULL)
```

Arguments

`se_object`	a SummarizedExperiment object containing the phenotype data
`gp_field`	character, corresponding to the phenotype data used to categorize samples into distinct groups if any

Value

a dataframe containing 3 fields: Sample, Group and ID.

Examples

```
nrows <- 200
ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
colData <- data.frame(Treatment=rep(c("ChIP", "Input"), 3),
                      row.names=LETTERS[1:6])
se <- SummarizedExperiment::SummarizedExperiment(assays=list(counts=counts), colData=colData)
convertSE(se, "Treatment")
```

convertVector2Df *Convert a vector of plate coordinates into a dataframe*

Description

Function converting the format of "Letter-Digit" coordinates into a dataframe containing these coordinates in Row, Column.

Usage

```
convertVector2Df(chr_wells, max_Row, max_Col, status = NA)
```

Arguments

chr_wells	character string containing the wells
max_Row	integer, maximal number of lines in the plate
max_Col	integer, maximal number of columns in the plate
status	character, the status of the wells

Value

result, dataframe containing wells coordinates

Examples

```
# convert the vector of well coordinates into a dataframe
convertVector2Df("A1,C2,A3,B12,C42",3,42,"specify_status")

# supports uppercase / lowercase letters
convertVector2Df("a1,C2,A3,b12,C42",3,42,"specify_status")
```

data_test *Fictitious clinical data for demonstration.*

Description

A demo dataset containing the age and other attributes of 193 fictitious patients. It aims to help the user to test the shiny application of the wpm package.

Usage

```
data(data_test)
```

Format

A data frame with 193 rows and 7 variables

samples the samples to be analyzed representing fictitious patients.

age age of the patients under 5 age groups, in years (20-30;30-40;40-50;50-60;60-70)

gender gender of the patients, F (for Female) and M (for Male)

treatment the treatment each patient received, Ctrl (Control), treatment A, treatment B and treatment C

diabetes presence of diabetes, 0 (no) and 1 (yes)

gender-treatment A combination between the gender and treatment fields

age-diabetes A combination between the age and the diabetes fields

Author(s)

Helene Borges <borges.helene.sophie@gmail.com>

`defineBufferCoords` *Determines buffer wells coordinates on a plate*

Description

function to place the buffer solutions on the plate according to the selected mode: it generates a dataframe containing the row and column coordinates for each buffer solution.

Usage

```
defineBufferCoords(p_lines, p_cols, mod = "none", start_buffer)
```

Arguments

`p_lines`	number of rows on the plate
`p_cols`	number of columns on the plate
`mod`	character, can be "none", "by_column", "by_row" or "checkerboard"
`start_buffer`	character, "even" means that the buffers will be positioned on the even rows of the plate. Otherwise, they will be positioned on the odd rows.

Value

a dataframe containing the buffer wells with their coordinates.

drawMap *Generate a ggplot object of a plate plan*

Description

Function to plot the input dataframe containing the Sample names, the Row, Column coordinates, the group and the status

Usage

```
drawMap(df, sample_gps, gp_levels, plate_lines, plate_cols, project_title)
```

Arguments

df	dataframe containing user data and special wells if any.
sample_gps	number of distinct groups in the file before adding the special wells to df
gp_levels	is Group column levels before adding the special wells to df
plate_lines	integer, number of plate's lines
plate_cols	integer, number of plate's columns
project_title	character, the user's project title

Value

g, a ggplot object corresponding to the generated plate map.

Examples

```
# example of data containing 5 biological samples, 2 forbidden wells,
# 2 buffers and 3 not random wells
user_data <- data.frame("Sample" = c(as.character(seq_len(5)), rep_len(NA, 7)),
                        "Group" = c(c("A","B","C","A","B"),
                                    rep_len("forbidden", 2),
                                    rep_len("buffer", 2),
                                    rep_len("fixed", 3)),
                        "ID" = c(seq_len(5),rep_len(NA,7)),
                "Well" = c("A2","B3","C3","B4","A3","A1","A4","B2","C2","B1","C1","C4"),
                        "Status" = c(rep_len("toRandom", 5),
                                     rep_len("forbidden", 2),
                                     rep_len("buffer", 2),
                                     rep_len("fixed", 3)),
                        "Row" = c(1,2,3,2,1,1,1,2,3,2,3,3),
                        "Column" = c(2,3,3,4,3,1,4,2,2,1,1,4))
p <- "My Project"
gp_lvl <- levels(as.factor(c("A","B","C")))
drawMap(df = user_data, sample_gps = 3, gp_levels = gp_lvl, plate_lines = 3,
        plate_cols = 4, project_title = p)
```

```
# also works when giving a plate with more wells than the number of samples to place.
drawMap(df = user_data, sample_gps = 3, gp_levels = gp_lvl, plate_lines = 8,
        plate_cols = 12, project_title = p)
```

`findNEWSneighbors` *Find the 4 cardinal neighbors of an element of a matrix*

Description

Function for spatial contraints: the North, East, West and South neighbors of the current element (i,j) of the matrix m.

Usage

```
findNEWSneighbors(m, i, j)
```

Arguments

`m`	matrix
`i`	integer, line index in the matrix
`j`	integer, column index in the matrix

Value

A vector containing the North, East, West and South neighbors of the element (i,j) of the matrix being processed.

`findNSneighbors` *Find the top and bottom neighbors of an element of a matrix*

Description

Function for spatial constraint that only looks for North (top) and South (bottom) neighbors of the current element (i,j) of the matrix m.

Usage

```
findNSneighbors(m, i, j)
```

Arguments

`m`	matrix
`i`	integer, line index in the matrix
`j`	integer, column index in the matrix

Value

A vector containing the North and South neighbors of the element (i,j) of the matrix being processed.

---

findWEneighbors *Find the left and right neighbors of an element of a matrix*

Description

Function for spatial constraint that only looks for West (left) and East (right) neighbors of the current element (i,j) of the matrix m.

Usage

```
findWEneighbors(m, i, j)
```

Arguments

m	matrix
i	integer, line index in the matrix
j	integer, column index in the matrix

Value

A vector containing the West and East neighbors of the element (i,j) of the matrix being processed

---

generateMap *Generate a plate map according to the input parameters*

Description

This function generates a plate map using a backtracking algorithm and returns a dataframe if success. If it fails to find a solution, returns NULL. If there are not enough wells to place all the samples, returns 0.

Usage

```
generateMap(
  user_df,
  nb_rows,
  nb_cols,
  df_forbidden,
  mod,
  max_it,
  updateProgress = NULL
)
```

Arguments

user_df	dataframe containing 9 features: Sample, ID, Group, Sample.name, Well, Status, Row, Column, Plate. See details.
nb_rows	numeric, number of lines on the plate
nb_cols	numeric, number of columns on the plate
df_forbidden	dataframe with the same structure than user_df, but for the forbidden, buffer solutions and Not randomized wells.
mod	character, neighborhood spatial constraint
max_it	numeric, maximum number of attempts to generate a plate plan before returning a failure.
updateProgress	shiny object, reports progress to the user.

Details

The dataframe is generated using dedicated functions of the wpm package: 'convertCSV()', 'convertESet()' or 'convertSE()'. But the user can also generate it by hand.

A number of attempts is allowed. Consequently, if the maximal number if attempts is reeched and no solution was found with the backtracking (i.e. the randomWalk does not return a dataframe), this function prints a warning message and returns NULL. If a solution is found, then it returns the dataframe.

Value

Returns a dataframe containing all the data of the plate map(s)

---

joinDataframes *Binds multiple dataframes together*

Description

Function that merges dataframes that contain wells of different types. To do this, it verifies that all the conditions provided are compatible with each other in order to be able to launch WPM on this data.

Usage

```
joinDataframes(
  forbidden_w = NULL,
  buffer_w = NULL,
  fixed_w = NULL,
  nb_samples,
  totalNbWells,
  nb_p
)
```

Arguments

forbidden_w	dataframe, the forbidden wells
buffer_w	datarame, the buffer wells
fixed_w	dataframe, the quality control wells
nb_samples	numeric, the number of samples to place using the backtracking algorithm.
totalNbWells	numeric, the total number of wells that can be filled.
nb_p	numeric, number of plates to fill

Value

a dataframe containing all the special wells

---

randomWalk *Random walk of the matrix to fill*

Description

Returns the user dataframe updated after choosing randomly a well on the plate (matrix) and randomly choosing a sample ID that satisfies all the constraints.

Usage

```
randomWalk(m, toVisit, d, constraint)
```

Arguments

m	is a matrix corresponding to the plate to be filled.
toVisit	contains the wells in form "A1", and contains only the wells authorized to be filled in
d	is the dataframe containing the data supplied by the user.
constraint	character string corresponding to the spatial constraint selected by the user

Value

a dataframe corresponding to the user-supplied data. This dataframe is an updated version, where the columns ‘Row‘ and ‘Column‘ are filled with the coordinates of the chosen well. If no solution is found for the current selected well, then this function returns 1.

resample *Randomly take a number of elements in a vector*

Description

This function allows to pick up the last element in a vector when the parameter size is equal to 1. Passes parameters to 'sample.int' like size.

Usage

```
resample(x, ...)
```

Arguments

x	is a vector
...	parameters given to the function sample.int

Value

a vector of length equal to size parameter.

solveCell *Affects a sample to the chosen cell in the plate*

Description

This function chooses a sample randomly from among those who respect the neighborhood constraints and who have not yet been assigned to a well.

Usage

```
solveCell(m, d, i, j, already_drawn, constraint)
```

Arguments

m	Matrix representing the plate plan.
d	Dataframe containing the samples to place.
i	Line index of the chosen well.
j	Column index of the chosen well.
already_drawn	Vector of samples already affected to wells.
constraint	Character. Corresponds to the neighborhood constraint mode.

Value

If there is no possibility to find a valid sample, the function returns an error value (1). If a sample is chosen, then this function returns two objects: * __m__ The matrix updated with the new added sample. * __already_drawn__ The vector of already placed samples updated.

wpm *Run the Shiny Application of Well Plate Maker*

Description

Run the Shiny Application of Well Plate Maker

Usage

```
wpm(...)
```

Arguments

...	A series of options to be used inside the app.

Value

a shiny application object with golem options

Examples

```
if(interactive()) {wpm()}
```

wrapperWPM *Generate plate plans in a single step*

Description

Wrapper function that generates plate plans like the wpm shiny application. This feature allows the user to use the wpm package from the command line rather than going through a web application.

Usage

```
wrapperWPM(
  user_df,
  plate_dims,
  nb_plates,
  forbidden_wells = NULL,
  buffer_wells = NULL,
  fixed_wells = NULL,
  spatial_constraint = "none",
  max_iteration = 20
)
```

Arguments

user_df	dataframe containing user data obtained with the 'convertCSV()' or 'convertESet()' functions.
plate_dims	list, containing 2 values: the first is the number of plate's lines and second is the number of plate's columns.
nb_plates	numeric, corresponds to the number of plates to fill
forbidden_wells	character, the wells that will not be used at all for the experiment. This argument needs to be a character string giving the wells coordinates of the form "LetterNumber" (eg. "A1" for the well positionned in the first row/ first column of the plate).
buffer_wells	character, the wells that will be used during experiment but without biological sample in it. Same input structure as for forbidden_wells parameter.
fixed_wells	character, the wells that will be used for Quality Control samples or standards during the Experiment. Same input structure as for forbidden_wells parameter.
spatial_constraint	character, is the spatial constraint used to place the samples on the plate. It can also be called neighborhood constraint. Currently, the possible values are "none", "NS" (for North-South), "WE" (for West-East) and "NEWS" (North-South-East-West).
max_iteration	numeric, maximal number of attemps for wpm to find a valid solution.

Value

a dataframe if wpm finds a solution.

Examples

```
# create a MSnSet toy example
sample_names <- c("s1","s2","s3","s4", "s5")
M <- matrix(NA, nrow = 4, ncol = 5)
colnames(M) <- sample_names
rownames(M) <- paste0("id", LETTERS[1:4])
pd <- data.frame(Environment = rep_len(LETTERS[1:3], 5),
                 Category = rep_len(1:2, 5), row.names = sample_names)
rownames(pd) <- colnames(M)
x <- MSnbase::MSnSet(exprs = M,pData =  pd)
# convert it to a valid dataframe for wpm
df <- convertESet(x, "Environment")
# run wpm on the toy example
wrapperWPM(user_df = df, plate_dims = list(8,12), nb_plates = 1,
           forbidden_wells = "A1,A2,A3", fixed_wells = "B1,B2",
           spatial_constraint = "NS")
```

Index

∗ **datasets**
data_test, 8

backtracking, 2
balancedGrpDistrib, 3

checkConstraints, 4
checkWpmInputs, 5
convertCSV, 5
convertESet, 6
convertSE, 7
convertVector2Df, 8

data_test, 8
defineBufferCoords, 9
drawMap, 10

findNEWSneighbors, 11
findNSneighbors, 11
findWEneighbors, 12

generateMap, 12

joinDataframes, 13

randomWalk, 14
resample, 15

solveCell, 15

wpm, 16
wrapperWPM, 16