Package ‘NAIR’

March 3, 2024

**Type** Package

**Title** Network Analysis of Immune Repertoire

**Version** 1.0.4

**Description** Pipelines for studying the adaptive immune repertoire of T cells and B cells via network analysis based on receptor sequence similarity. Relate clinical outcomes to immune repertoires based on their network properties, or to particular clusters and clones within a repertoire. Yang et al. (2023) <doi:10.3389/fimmu.2023.1181825>.

**License** GPL (>= 3)

**Encoding** UTF-8

**Depends** R (>= 3.1.0)

**Imports** Rcpp (>= 1.0.8), lifecycle, igraph, ggraph, ggplot2, grDevices, utils, Matrix, stats, dplyr, rlang

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**LinkingTo** Rcpp, RcppArmadillo (>= 0.10.8.0.0)

**VignetteBuilder** knitr

**NeedsCompilation** yes

**Author** Brian Neal [aut, cre], Hai Yang [aut], Daniil Matveev [aut], Phi Long Le [aut], Li Zhang [cph, aut]

**Maintainer** Brian Neal <Brian.Neal@ucsf.edu>

**Repository** CRAN

**Date/Publication** 2024-03-03 00:52:36 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAIR-package</td>
<td>2</td>
</tr>
<tr>
<td>addClusterMembership</td>
<td>3</td>
</tr>
<tr>
<td>addClusterStats</td>
<td>6</td>
</tr>
<tr>
<td>addNodeNetworkStats</td>
<td>11</td>
</tr>
<tr>
<td>addNodeStats</td>
<td>14</td>
</tr>
<tr>
<td>addPlots</td>
<td>16</td>
</tr>
<tr>
<td>aggregateIdenticalClones</td>
<td>20</td>
</tr>
<tr>
<td>buildAssociatedClusterNetwork</td>
<td>23</td>
</tr>
<tr>
<td>buildPublicClusterNetwork</td>
<td>26</td>
</tr>
<tr>
<td>buildPublicClusterNetworkByRepresentative</td>
<td>31</td>
</tr>
<tr>
<td>buildRepSeqNetwork</td>
<td>37</td>
</tr>
<tr>
<td>chooseNodeStats</td>
<td>43</td>
</tr>
<tr>
<td>combineSamples</td>
<td>46</td>
</tr>
<tr>
<td>extractLayout</td>
<td>51</td>
</tr>
<tr>
<td>filterInputData</td>
<td>52</td>
</tr>
<tr>
<td>findAssociatedClones</td>
<td>53</td>
</tr>
<tr>
<td>findAssociatedSeqs</td>
<td>57</td>
</tr>
<tr>
<td>findPublicClusters</td>
<td>63</td>
</tr>
<tr>
<td>generateAdjacencyMatrix</td>
<td>68</td>
</tr>
<tr>
<td>generateNetworkGraph</td>
<td>71</td>
</tr>
<tr>
<td>generateNetworkObjects</td>
<td>72</td>
</tr>
<tr>
<td>getClusterStats</td>
<td>74</td>
</tr>
<tr>
<td>getNeighborhood</td>
<td>77</td>
</tr>
<tr>
<td>hamDistBounded</td>
<td>79</td>
</tr>
<tr>
<td>labelClusters</td>
<td>81</td>
</tr>
<tr>
<td>labelNodes</td>
<td>83</td>
</tr>
<tr>
<td>levDistBounded</td>
<td>85</td>
</tr>
<tr>
<td>plotNetworkGraph</td>
<td>87</td>
</tr>
<tr>
<td>saveNetwork</td>
<td>89</td>
</tr>
<tr>
<td>saveNetworkPlots</td>
<td>91</td>
</tr>
<tr>
<td>simulateToyData</td>
<td>93</td>
</tr>
</tbody>
</table>

### Description

To learn about the NAIR package and get started, visit the package website, or browse the package vignettes offline:

```r
browseVignettes(package = "NAIR")
```

The following vignette is a good place to start:

```r
vignette("NAIR", package = "NAIR")
```
addClusterMembership

Author(s)
- Brian Neal (<Brian.Neal@ucsf.edu>), Maintainer
- Hai Yang (<Hai.Yang@ucsf.edu>)
- Phi-Long Le (<PhiLong.Le@ucsf.edu>)
- Li Zhang (<Li.Zhang@ucsf.edu>)

See Also
- Package website
- Github page
- Report bugs and issues here

addClusterMembership  Partition a Network Graph Into Clusters

Description
Given a list of network objects returned by buildRepSeqNetwork() or generateNetworkObjects(), partitions the network graph into clusters using the specified clustering algorithm, adding a cluster membership variable to the node metadata.

Usage
addClusterMembership(
  net,
  cluster_fun = "fast_greedy",
  cluster_id_name = "cluster_id",
  overwrite = FALSE,
  verbose = FALSE,
  ...
)

Arguments
- net A list of network objects conforming to the output of buildRepSeqNetwork() or generateNetworkObjects(). See details. Alternatively, this argument accepts the network igraph, with the node metadata passed to the data argument. However, this alternative functionality is deprecated and will eventually be removed.
- cluster_fun A character string specifying the clustering algorithm to use. See details.
- cluster_id_name A character string specifying the name of the cluster membership variable to be added to the node metadata.
addClusterMembership

overwrite Logical. Should the variable specified by `cluster_id_name` be overwritten if it already exists?

verbose Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to `stderr()`.

... Named optional arguments to the function specified by `cluster_fun`.

data [Deprecated] See `net`.

fun [Deprecated] Replaced by `cluster_fun`.

Details

The list `net` must contain the named elements `igraph` (of class `igraph`), `adjacency_matrix` (a `matrix` or `dgCMatrix` encoding edge connections), and `node_data` (a `data.frame` containing node metadata), all corresponding to the same network. The lists returned by `buildRepSeqNetwork()` and `generateNetworkObjects()` are examples of valid inputs for the `net` argument.

Alternatively, the `igraph` may be passed to `net` and the node metadata to `data`. However, this alternative functionality is deprecated and will eventually be removed.

A clustering algorithm is used to partition the network graph into clusters (densely-connected subgraphs). Each cluster represents a collection of clones/cells with similar receptor sequences. The method used to partition the graph depends on the choice of clustering algorithm, which is specified using the `cluster_fun` argument.

The available options for `cluster_fun` are listed below. Each refers to an `igraph` function implementing a particular clustering algorithm. Follow the links to learn more about the individual clustering algorithms.

- "edge_betweenness"
- "fast_greedy"
- "infomap"
- "label_prop"
- "leading_eigen"
- "leiden"
- "louvain"
- "optimal"
- "spinglass"
- "walktrap"

Optional arguments to each clustering algorithm can have their values specified using the ellipses (…) argument of `addClusterMembership()`.

Each cluster is assigned a numeric cluster ID. A cluster membership variable, whose name is specified by `cluster_id_name`, is added to the node metadata, encoding the cluster membership of the node for each row. The cluster membership is encoded as the cluster ID number of the cluster to which the node belongs.

The `overwrite` argument controls whether to overwrite pre-existing data. If the variable specified by `cluster_id_name` is already present in the node metadata, then `overwrite` must be set to TRUE.
in order to perform clustering and overwrite the variable with new cluster membership values. Alternatively, by specifying a value for `cluster_id_name` that is not among the variables in the node metadata, a new cluster membership variable can be created while preserving the old cluster membership variable. In this manner, clustering can be performed multiple times on the same network using different clustering algorithms, without losing the results.

**Value**

If the variable specified by `cluster_id_name` is not present in `net$node_data`, returns a copy of `net` with this variable added to `net$node_data` encoding the cluster membership of the network node corresponding to each row. If the variable is already present and `overwrite = TRUE`, then its values are replaced with the new values for cluster membership.

Additionally, if `net` contains a list named `details`, then the following elements will be added to `net$details` if they do not already exist:

- `clusters_in_network`: A named numeric vector of length 1. The first entry’s name is the name of the clustering algorithm, and its value is the number of clusters resulting from performing clustering on the network.
- `cluster_id_variable`: A named numeric vector of length 1. The first entry’s name is the name of the corresponding cluster membership variable in the node metadata (i.e., the value of `cluster_id_name`).

If `net$details` already contains these elements, they will be updated according to whether the cluster membership variable specified by `cluster_id_name` is added to `net$node_data` or already exists and is overwritten. In the former case (the cluster membership variable does not already exist), the length of each vector (`clusters_in_network` and `cluster_id_variable`) is increased by 1, with the new information appended as a new named entry to each. In the latter case (the cluster membership variable is overwritten), the new information overwrites the name and value of the last entry of each vector.

In the event where `overwrite = FALSE` and `net$node_data` contains a variable with the same name as the value of `cluster_id_name`, then an unaltered copy of `net` is returned with a message notifying the user.

Under the alternative (deprecated) input format where the node metadata is passed to `data` and the igraph is passed to `net`, the node metadata is returned instead of the list of network objects, with the cluster membership variable added or updated as described above.

**Author(s)**

Brian Neal (<Brian.Neal@ucsf.edu>)

**References**


Webpage for the NAIR package
See Also

addClusterStats() labelClusters()

Examples

set.seed(42)
toy_data <- simulateToyData()

net <- generateNetworkObjects(
  toy_data, "CloneSeq"
)

# Perform cluster analysis,
# add cluster membership to net$node_data
net <- addClusterMembership(net)

net$details$clusters_in_network
net$details$cluster_id_variable

# overwrite values in net$node_data$cluster_id
# with cluster membership values obtained using "cluster_leiden" algorithm
net <- addClusterMembership(
  net,
  cluster_fun = "leiden",
  overwrite = TRUE
)

net$details$clusters_in_network
net$details$cluster_id_variable

# perform clustering using "cluster_louvain" algorithm
# saves cluster membership values to net$node_data$cluster_id_louvain
# (net$node_data$cluster_id retains membership values from "cluster_leiden")
net <- addClusterMembership(
  net,
  cluster_fun = "louvain",
  cluster_id_name = "cluster_id_louvain",
)

net$details$clusters_in_network
net$details$cluster_id_variable

addClusterStats

Compute Cluster-Level Network Properties

Description

Given a list of network objects returned by buildRepSeqNetwork() or generateNetworkObjects(), computes cluster-level network properties, performing clustering first if needed. The list of network objects is returned with the cluster properties added as a data frame.
Usage

addClusterStats(
  net,
  cluster_id_name = "cluster_id",
  seq_col = NULL,
  count_col = NULL,
  degree_col = "degree",
  cluster_fun = "fast_greedy",
  overwrite = FALSE,
  verbose = FALSE,
  ...
)

Arguments

net
A list of network objects conforming to the output of buildRepSeqNetwork() or generateNetworkObjects(). See details.

cluster_id_name
A character string specifying the name of the cluster membership variable in net$node_data that identifies the cluster to which each node belongs. If the variable does not exist, it will be added by calling addClusterMembership(). If the variable does exist, its values will be used unless overwrite = TRUE, in which case its values will be overwritten and the new values used.

seq_col
Specifies the column(s) of net$node_data containing the receptor sequences upon whose similarity the network is based. Accepts a character or numeric vector of length 1 or 2, containing either column names or column indices. If provided, related cluster-level properties will be computed. The default NULL will use the value contained in net$details$seq_col if it exists and is valid.

count_col
Specifies the column of net$node_data containing a measure of abundance (such as clone count or UMI count). Accepts a character string containing the column name or a numeric scalar containing the column index. If provided, related cluster-level properties will be computed.

degree_col
Specifies the column of net$node_data containing the network degree of each node. Accepts a character string containing the column name. If the column does not exist, it will be added.

cluster_fun
A character string specifying the clustering algorithm to use when adding or overwriting the cluster membership variable in net$node_data specified by cluster_id_name. Passed to addClusterMembership().

overwrite
Logical. If TRUE and net already contains an element named cluster_data, it will be overwritten. Similarly, if overwrite = TRUE and net$node_data contains a variable whose name matches the value of cluster_id_name, then its values will be overwritten with new cluster membership values (obtained using addClusterMembership() with the specified value of cluster_fun), and cluster properties will be computed based on the new values.
addClusterStats

verbose Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().

Details

The list net must contain the named elements igraph (of class igraph), adjacency_matrix (a matrix or dgCMatrix encoding edge connections), and node_data (a data.frame containing node metadata), all corresponding to the same network. The lists returned by buildRepSeqNetwork() and generateNetworkObjects() are examples of valid inputs for the net argument.

If the network graph has previously been partitioned into clusters using addClusterMembership() and the user wishes to compute network properties for these clusters, the name of the cluster membership variable in net$node_data should be provided to the cluster_id_name argument.

If the value of cluster_id_name is not the name of a variable in net$node_data, then clustering is performed using addClusterMembership() with the specified value of cluster_fun, and the cluster membership values are written to net$node_data using the value of cluster_id_name as the variable name. If overwrite = TRUE, this is done even if this variable already exists.

Value

A modified copy of net, with cluster properties contained in the element cluster_data. This is a data.frame containing one row for each cluster in the network and the following variables:

- cluster_id: The cluster ID number.
- node_count: The number of nodes in the cluster.
- mean_seq_length: The mean sequence length in the cluster. Only present when length(seq_col) == 1.
- A_mean_seq_length: The mean first sequence length in the cluster. Only present when length(seq_col) == 2.
- B_mean_seq_length: The mean second sequence length in the cluster. Only present when length(seq_col) == 2.
- mean_degree: The mean network degree in the cluster.
- max_degree: The maximum network degree in the cluster.
- seq_w_max_degree: The receptor sequence possessing the maximum degree within the cluster. Only present when length(seq_col) == 1.
- A_seq_w_max_degree: The first sequence of the node possessing the maximum degree within the cluster. Only present when length(seq_col) == 2.
- B_seq_w_max_degree: The second sequence of the node possessing the maximum degree within the cluster. Only present when length(seq_col) == 2.
addClusterStat

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agg_count</td>
<td>The aggregate count among all nodes in the cluster (based on the counts in count_col).</td>
</tr>
<tr>
<td>max_count</td>
<td>The maximum count among all nodes in the cluster (based on the counts in count_col).</td>
</tr>
<tr>
<td>seq_w_max_count</td>
<td>The receptor sequence possessing the maximum count within the cluster. Only present when length(seq_col) == 1.</td>
</tr>
<tr>
<td>A_seq_w_max_count</td>
<td>The first sequence of the node possessing the maximum count within the cluster. Only present when length(seq_col) == 2.</td>
</tr>
<tr>
<td>B_seq_w_max_count</td>
<td>The second sequence of the node possessing the maximum count within the cluster. Only present when length(seq_col) == 2.</td>
</tr>
<tr>
<td>diameter_length</td>
<td>The longest geodesic distance in the cluster, computed as the length of the vector returned by get_diameter().</td>
</tr>
<tr>
<td>assortativity</td>
<td>The assortativity coefficient of the cluster’s graph, based on the degree (minus one) of each node in the cluster (with the degree computed based only upon the nodes within the cluster). Computed using assortativity_degree().</td>
</tr>
<tr>
<td>global_transitivity</td>
<td>The transitivity (i.e., clustering coefficient) for the cluster’s graph, which estimates the probability that adjacent vertices are connected. Computed using transitivity() with type = “global”.</td>
</tr>
<tr>
<td>edge_density</td>
<td>The number of edges in the cluster as a fraction of the maximum possible number of edges. Computed using edge_density().</td>
</tr>
<tr>
<td>degree_centrality_index</td>
<td>The centrality index of the cluster’s graph based on within-cluster network degree. Computed as the centralization element of the output from centr_degree().</td>
</tr>
<tr>
<td>closeness_centrality_index</td>
<td>The centrality index of the cluster’s graph based on closeness, i.e., distance to other nodes in the cluster. Computed using centralization().</td>
</tr>
<tr>
<td>eigen_centrality_index</td>
<td>The centrality index of the cluster’s graph based on the eigenvector centrality scores, i.e., values of the first eigenvector of the adjacency matrix for the cluster. Computed as the centralization element of the output from centr_eigen().</td>
</tr>
<tr>
<td>eigen_centrality_eigenvalue</td>
<td>The eigenvalue corresponding to the first eigenvector of the adjacency matrix for the cluster. Computed as the value element of the output from eigen_centrality().</td>
</tr>
</tbody>
</table>

If net$node_data did not previously contain a variable whose name matches the value of cluster_id_name, then this variable will be present and will contain values for cluster membership, obtained through a call to addClusterMembership() using the clustering algorithm specified by cluster_fun.

If net$node_data did previously contain a variable whose name matches the value of cluster_id_name and overwrite = TRUE, then the values of this variable will be overwritten with new values for cluster membership, obtained as above based on cluster_fun.
If `net$node_data` did not previously contain a variable whose name matches the value of `degree_col`, then this variable will be present and will contain values for network degree.

Additionally, if `net` contains a list named `details`, then the following elements will be added to `net$details`, or overwritten if they already exist:

- **cluster_data_goes_with**
  
  A character string containing the value of `cluster_id_name`. When `net$node_data` contains multiple cluster membership variables (e.g., from applying different clustering methods), `cluster_data_goes_with` allows the user to distinguish which of these variables corresponds to `net$cluster_data`.

- **count_col_for_cluster_data**
  
  A character string containing the value of `count_col`. If `net$node_data` contains multiple count variables, this allows the user to distinguish which of these variables corresponds to the count-related properties in `net$cluster_data`, such as `max_count`. If `count_col = NULL`, then the value will be NA.

### Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

### References


Webpage for the NAIR package

### See Also

- `addClusterMembership()`
- `getClusterStats()`
- `labelClusters()`

### Examples

```r
set.seed(42)
toy_data <- simulateToyData()

net <- generateNetworkObjects(
  toy_data, "CloneSeq"
)

net <- addClusterStats(
  net,
  count_col = "CloneCount"
)

head(net$cluster_data)
net$details

# won't change net since net$cluster_data exists
net <- addClusterStats(
  net,
```
addNodeNetworkStats

```
  count_col = "CloneCount",
  cluster_fun = "leiden",
  verbose = TRUE
)

# overwrites values in net$cluster_data
# and cluster membership values in net$node_data$cluster_id
# with values obtained using "cluster_leiden" algorithm
net <- addClusterStats(
  net,
  count_col = "CloneCount",
  cluster_fun = "leiden",
  overwrite = TRUE
)

net$details

# overwrites existing values in net$cluster_data
# with values obtained using "cluster_louvain" algorithm
# saves cluster membership values to net$node_data$cluster_id_louvain
# (net$node_data$cluster_id retains membership values from "cluster_leiden")
net <- addClusterStats(
  net,
  count_col = "CloneCount",
  cluster_fun = "louvain",
  cluster_id_name = "cluster_id_louvain",
  overwrite = TRUE
)

net$details

# perform clustering using "cluster_fast_greedy" algorithm,
# save cluster membership values to net$node_data$cluster_id_greedy
net <- addClusterMembership(
  net,
  cluster_fun = "fast_greedy",
  cluster_id_name = "cluster_id_greedy"
)

# compute cluster properties for the clusters from previous step
# overwrites values in net$cluster_data
net <- addClusterStats(
  net,
  cluster_id_name = "cluster_id_greedy",
  overwrite = TRUE
)

net$details
```

---

**addNodeNetworkStats**  
*Compute Node-Level Network Properties*
addNodeNetworkStats

Description

Given the node metadata and igraph for a network, computes a specified set of network properties for the network nodes. The node metadata is returned with each property added as a variable.

This function was deprecated in favor of addNodeStats() in NAIR 1.0.1. The new function accepts and returns the entire list of network objects returned by buildRepSeqNetwork() or by generateNetworkObjects(). It can compute cluster membership and add the values to the node metadata. It additionally updates the list element details with further information linking the node-level and cluster-level metadata.

Usage

addNodeNetworkStats(
  data,
  net,
  stats_to_include = chooseNodeStats(),
  cluster_fun = "fast_greedy",
  cluster_id_name = "cluster_id",
  overwrite = FALSE,
  verbose = FALSE,
  ...
)

Arguments

data A data frame containing the node-level metadata for the network, with each row corresponding to a network node.
net The network igraph.
stats_to_include Specifies which network properties to compute. Accepts a vector created using chooseNodeStats() or exclusiveNodeStats(), or the character string "all" to compute all network properties.
cluster_fun A character string specifying the clustering algorithm to use when computing cluster membership. Applicable only when stats_to_include = "all" or stats_to_include["cluster_id"] is TRUE. Passed to addClusterMembership().
cluster_id_name A character string specifying the name of the cluster membership variable to be added to data. Applicable only when stats_to_include = "all" or stats_to_include["cluster_id"] is TRUE. Passed to addClusterMembership().
overwrite Logical. If TRUE and data contains a variable whose name matches the value of cluster_id_name, then its values will be overwritten with new cluster membership values (obtained using addClusterMembership() with the specified value of cluster_fun). Applicable only when stats_to_include = "all" or stats_to_include["cluster_id"] is TRUE.
verbose Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().
Named optional arguments to the function specified by `cluster_fun`.

**Details**

Node-level network properties are properties that pertain to each individual node in the network graph.

Some are local properties, meaning that their value for a given node depends only on a subset of the nodes in the network. One example is the network degree of a given node, which represents the number of other nodes that are directly joined to the given node by an edge connection.

Other properties are global properties, meaning that their value for a given node depends on all of the nodes in the network. An example is the authority score of a node, which is computed using the entire graph adjacency matrix (if we denote this matrix by $A$, then the principal eigenvector of $A^T A$ represents the authority scores of the network nodes).

See `chooseNodeStats()` for a list of the available node-level network properties.

**Value**

A copy of `data` with with an additional column for each new network property computed. See `chooseNodeStats()` for the network property names, which are used as the column names, except for the cluster membership variable, whose name is the value of `cluster_id_name`.

**Author(s)**

Brian Neal (<Brian.Neal@ucsf.edu>)

**References**


Webpage for the NAIR package

**See Also**

`addNodeStats()` `chooseNodeStats()`

**Examples**

```r
set.seed(42)
toy_data <- simulateToyData()

net <-
genenerateNetworkObjects(
  toy_data,
  "CloneSeq"
)

net$node_data <-
addNodeNetworkStats(
  net$node_data,
  net$igraph
)```
addNodeStats

Compute Node-Level Network Properties

Description

Given a list of network objects returned by `buildRepSeqNetwork()` or `generateNetworkObjects()`, computes a specified set of network properties for the network nodes. The list of network objects is returned with each property added as a variable to the node metadata.

Usage

```r
addNodeStats(
  net,
  stats_to_include = chooseNodeStats(),
  cluster_fun = "fast_greedy",
  cluster_id_name = "cluster_id",
  overwrite = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

- **net**  
  A list of network objects conforming to the output of `buildRepSeqNetwork()` or `generateNetworkObjects()`. See details.

- **stats_to_include**  
  Specifies which network properties to compute. Accepts a vector created using `chooseNodeStats()` or `exclusiveNodeStats()`, or the character string "all" to compute all network properties.

- **cluster_fun**  
  A character string specifying the clustering algorithm to use when computing cluster membership. Applicable only when `stats_to_include = "all"` or `stats_to_include["cluster_id"]` is TRUE. Passed to `addClusterMembership()`.

- **cluster_id_name**  
  A character string specifying the name of the cluster membership variable to be added to the node metadata. Applicable only when `stats_to_include = "all"` or `stats_to_include["cluster_id"]` is TRUE. Passed to `addClusterMembership()`.

- **overwrite**  
  Logical. If TRUE and `net$node_data` contains a variable whose name matches the value of `cluster_id_name`, then its values will be overwritten with new cluster membership values (obtained using `addClusterMembership()`), to which the values of `cluster_fun`, `overwrite` are applied. Applicable only when `stats_to_include = "all"` or `stats_to_include["cluster_id"]` is TRUE.
verbose Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().

... Named optional arguments to the function specified by cluster_fun.

Details

Node-level network properties are properties that pertain to each individual node in the network graph.

Some are local properties, meaning that their value for a given node depends only on a subset of the nodes in the network. One example is the network degree of a given node, which represents the number of other nodes that are directly joined to the given node by an edge connection.

Other properties are global properties, meaning that their value for a given node depends on all of the nodes in the network. An example is the authority score of a node, which is computed using the entire graph adjacency matrix (if we denote this matrix by $A$, then the principal eigenvector of $A^T A$ represents the authority scores of the network nodes).

See chooseNodeStats() for a list of the available node-level network properties.

The list net must contain the named elements igraph (of class igraph), adjacency_matrix (a matrix or dgCMatrix encoding edge connections), and node_data (a data.frame containing node metadata), all corresponding to the same network. The lists returned by buildRepSeqNetwork() and generateNetworkObjects() are examples of valid inputs for the net argument.

Value

A modified copy of net, with net$node_data containing an additional column for each new network property computed. See chooseNodeStats() for the network property names, which are used as the column names, except for the cluster membership variable, whose name is the value of cluster_id_name.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package

See Also

chooseNodeStats()

Examples

set.seed(42)
toy_data <- simulateToyData()
```r
net <- generateNetworkObjects(
  toy_data, "CloneSeq"
)

# Add default set of node properties
net <- addNodeStats(net)

# Modify default set of node properties
net <- addNodeStats(
  net,
  stats_to_include =
  chooseNodeStats(
    closeness = TRUE,
    page_rank = FALSE
  )
)

# Add only the specified node properties
net <- addNodeStats(
  net,
  stats_to_include =
  exclusiveNodeStats(
    degree = TRUE,
    transitivity = TRUE
  )
)

# Add all node-level network properties
net <- addNodeStats(
  net,
  stats_to_include = "all"
)
```

---

**addPlots**

*Generate Plots of a Network Graph*

**Description**

Generates one or more ggraph plots of the network graph according to the user specifications.

`addPlots()` accepts and returns a list of network objects, adding the plots to the existing list contents. If the list already contains plots, the new plots will be created using the same coordinate layout as the existing plots.

`generateNetworkGraphPlots()` accepts the network igraph and node metadata, and returns a list containing plots.
addPlots

Usage

addPlots(
  net,
  print_plots = FALSE,
  plot_title = NULL,
  plot_subtitle = "auto",
  color_nodes_by = NULL,
  color_scheme = "default",
  color_legend = "auto",
  color_title = "auto",
  edge_width = 0.1,
  size_nodes_by = 0.5,
  node_size_limits = NULL,
  size_title = "auto",
  verbose = FALSE
)

generateNetworkGraphPlots(
  igraph,
  data,
  print_plots = FALSE,
  plot_title = NULL,
  plot_subtitle = NULL,
  color_nodes_by = NULL,
  color_scheme = "default",
  color_legend = "auto",
  color_title = "auto",
  edge_width = 0.1,
  size_nodes_by = 0.5,
  node_size_limits = NULL,
  size_title = "auto",
  layout = NULL,
  verbose = FALSE
)

Arguments

net A list of network objects conforming to the output of buildRepSeqNetwork() or generateNetworkObjects(). See details.
igraph An igraph object containing the network graph to be plotted.
data A data frame containing the node metadata for the network, with each row corresponding to a node.
print_plots A logical scalar; should plots be printed in the R plotting window?
plot_title A character string containing the plot title.
plot_subtitle A character string containing the plot subtitle. The default value "auto" generates a subtitle describing the settings used to construct the network, including the distance type and distance cutoff.
**addPlots**

- **color_nodes_by**: A vector specifying one or more node metadata variables used to encode the color of the nodes. One plot is generated for each entry, with each plot coloring the nodes according to the variable in the corresponding entry. This argument accepts a character vector where each entry is a column name of the node metadata. If this argument is `NULL`, generates a single plot with uncolored nodes.

- **color_scheme**: A character string specifying the color scale to use for all plots, or a character vector whose length matches that of `color_nodes_by`, with each entry specifying the color scale for the corresponding plot. "default" specifies the default `ggplot()` color scale. Other options are one of the viridis color scales (e.g., "plasma", "A" or other valid inputs to the option argument of `scale_color_viridis()` or (for discrete variables) a palette from `hcl.pals()` (e.g., "RdYlGn"). Each of the viridis color scales can include the suffix "-1" to reverse its direction (e.g., "plasma-1" or "A-1").

- **color_legend**: A logical scalar specifying whether to display the color legend in plots. The default value of "auto" shows the color legend if nodes are colored according to a continuous variable or according to a discrete variable with at most 20 distinct values.

- **color_title**: A character string specifying the title of the color legend in all plots, or a character vector whose length matches that of `color_nodes_by`, with each entry specifying the title of the color legend in the corresponding plot. Only applicable for plots with colored nodes. The value "auto" uses the corresponding value of `color_nodes_by`.

- **edge_width**: A numeric scalar specifying the width of the graph edges in the plot. Passed to the `width` argument of `geom_edge_link0()`.

- **size_nodes_by**: A numeric scalar specifying the size of the nodes in all plots, or the column name of a node metadata variable used to encode the size of the nodes in all plots. Alternatively, an argument value of `NULL` uses the default `ggraph` size for all nodes. Passed to the size aesthetic mapping of `geom_node_point()`.

- **node_size_limits**: A numeric vector of length 2, specifying the minimum and maximum node size. Only applicable if nodes are sized according to a variable. If `node_size_limits` = `NULL`, the default size scale will be used.

- **size_title**: A character string (or `NULL`) specifying the title for the size legend. Only applicable if nodes are sized according to a variable. The value "auto" uses the value of `size_nodes_by`.

- **layout**: A matrix specifying the coordinate layout of the network nodes, with one row for each node in the network and two columns. Each row specifies the x and y coordinates for the corresponding node. If `NULL`, the layout matrix is created using `igraph::layout_components()`. This argument can be used to create plots conforming to the same layout as previously-generated plots. It can also be used to generate plots with custom layouts.

- **verbose**: Logical. If `TRUE`, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to `stderr()`.
Details

The list `net` must contain the named elements `igraph` (of class `igraph`), `adjacency_matrix` (a `matrix` or `dgCMatrix` encoding edge connections), and `node_data` (a `data.frame` containing node metadata), all corresponding to the same network. The lists returned by `buildRepSeqNetwork()` and `generateNetworkObjects()` are examples of valid inputs for the `net` argument.

The arguments `color_nodes_by` and `size_nodes_by` accept the names of variables in the node metadata. For `addPlots()`, this is the data frame `node_data` contained in the list provided to the `net` argument. For `generateNetworkGraphPlots()`, this is the data frame provided to the `data` argument.

`addPlots()` adds the generated plots to the list `plots` contained in the list of network objects provided to `net`. The `plots` element is created if it does not already exist. If plots already exist, the new plots will be generated with the same coordinate layout as the existing plots. Each plot is named according to the variable used to color the nodes. If a plot already exists with the same name as one of the new plots, it will be overwritten with the new plot. If the `plots` list does not already contain an element named `graph_layout`, it will be added. This element contains the coordinate layout for the plots as a two-column matrix.

When calling `generateNetworkGraphPlots()`, if one wishes for the plots to be generated with the same coordinate layout as an existing plot, the layout matrix for the existing plot must be passed to the `layout` argument.

The plots can be printed to a pdf using `saveNetworkPlots()`.

Value

`addPlots()` returns a modified copy of `net` with the new plots contained in the element named `plots` (a list), in addition to any previously existing plots.

`generateNetworkGraphPlots()` returns a list containing the new plots.

Each plot is an object of class `ggraph`. Within the list of plots, each plot is named after the variable used to color the nodes. For a plot with uncolored nodes, the name is `uniform_color`.

The list containing the new plots also contains an element named `graph_layout`. This is a matrix specifying the coordinate layout of the nodes in the plots. It contains one row for each node in the network and two columns. Each row specifies the x and y coordinates for the corresponding node. This matrix can be used to generate additional plots with the same layout as the plots in the returned list.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package

Network Visualization article on package website
aggregateIdenticalClones

Aggregate Counts/Frequencies for Clones With Identical Receptor Sequences

Description

Given bulk Adaptive Immune Receptor Repertoire Sequencing (AIRR-Seq) data with clones indexed by row, returns a data frame containing one row for each unique receptor sequence. Includes the number of clones sharing each sequence, as well as aggregate values for clone count and clone frequency across all clones sharing each sequence. Clones can be grouped according to metadata, in which case aggregation is performed within (but not across) groups.

Usage

aggregateIdenticalClones(
  data,  # data frame
  clone_col,  # column containing clone index
  count_col,  # column containing clone count
  freq_col,  # column containing clone frequency
  grouping_cols = NULL,  # columns to group by
  verbose = FALSE  # print progress
)

Examples

```r
set.seed(42)
toy_data <- simulateToyData()

net <- buildNet(toy_data, "CloneSeq", node_stats = TRUE)

net <- addPlots(
  net,
  color_nodes_by =
    c("SampleID", "transitivity", "coreness"),
  color_scheme =
    c("Set 2", "mako-1", "plasma-1"),
  color_title =
    c("", "Transitvity", "Coreness"),
  size_nodes_by = "degree",
  node_size_limits = c(0.1, 1.5),
  plot_subtitle = NULL,
  print_plots = TRUE
)
```

See Also

labelNodes() labelClusters() saveNetworkPlots()
aggregateIdenticalClones

Arguments

- **data**: A data frame containing the bulk AIRR-Seq data, with clones indexed by row.
- **clone_col**: Specifies the column of data containing the receptor sequences. Accepts a character string containing the column name or a numeric scalar containing the column index.
- **count_col**: Specifies the column of data containing the clone counts. Accepts a character string containing the column name or a numeric scalar containing the column index.
- **freq_col**: Specifies the column of data containing the clone frequencies. Accepts a character string containing the column name or a numeric scalar containing the column index.
- **grouping_cols**: An optional character vector of column names or numeric vector of column indices, specifying one or more columns of data used to assign clones to groups. If provided, aggregation occurs within groups, but not across groups. See details.
- **verbose**: Logical. If `TRUE`, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to `stderr()`.

Details

If **grouping_cols** is left unspecified, the returned data frame will contain one row for each unique receptor sequence appearing in **data**.

If one or more columns of **data** are specified using the **grouping_cols** argument, then each clone (row) in **data** is assigned to a group based on its combination of values in these columns. If two clones share the same receptor sequence but belong to different groups, their receptor sequence will appear multiple times in the returned data frame, with one row for each group in which the sequence appears. In each such row, the aggregate clone count, aggregate clone frequency, and number of clones sharing the sequence are reported within the group for that row.

Value

A data frame whose first column contains the receptor sequences and has the same name as the column of **data** specified by **clone_col**. One additional column will be present for each column of **data** that is specified using the **grouping_cols** argument, with each having the same column name. The remaining columns are as follows:

- **AggregatedCloneCount**: The aggregate clone count across all clones (within the same group, if applicable) that share the receptor sequence in that row.
- **AggregatedCloneFrequency**: The aggregate clone frequency across all clones (within the same group, if applicable) that share the receptor sequence in that row.
- **UniqueCloneCount**: The number of clones (rows) in **data** (within the same group, if applicable) possessing the receptor sequence for the current row.
Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package

Examples

```r
my_data <- data.frame(
  clone_seq = c("ATCG", rep("ACAC", 2), rep("GGGG", 4)),
  clone_count = rep(1, 7),
  clone_freq = rep(1/7, 7),
  time_point = c("t_0", rep(c("t_0", "t_1"), 3)),
  subject_id = c(rep(1, 5), rep(2, 2))
)

my_data

aggregateIdenticalClones(
  my_data,
  "clone_seq",
  "clone_count",
  "clone_freq",
)

# group clones by time point
aggregateIdenticalClones(
  my_data,
  "clone_seq",
  "clone_count",
  "clone_freq",
  grouping_cols = "time_point"
)

# group clones by subject ID
aggregateIdenticalClones(
  my_data,
  "clone_seq",
  "clone_count",
  "clone_freq",
  grouping_cols = "subject_id"
)

# group clones by time point and subject ID
aggregateIdenticalClones(
  my_data,
  "clone_seq",
  "clone_count",
  "clone_freq",
"
buildAssociatedClusterNetwork

**Build Global Network of Associated TCR/BCR Clusters**

**Description**

Part of the workflow Searching for Associated TCR/BCR Clusters. Intended for use following findAssociatedClones().

Given data containing a neighborhood of similar clones around each associated sequence, combines the data into a global network and performs network analysis and cluster analysis.

**Usage**

```r
buildAssociatedClusterNetwork(
  file_list,
  input_type = "rds",
  data_symbols = "data", header = TRUE, sep,
  read.args = list(row.names = 1),
  seq_col,
  min_seq_length = NULL,
  drop_matches = NULL,
  drop_isolated_nodes = FALSE,
  node_stats = TRUE,
  stats_to_include =
    chooseNodeStats(cluster_id = TRUE),
  cluster_stats = TRUE,
  color_nodes_by = "GroupID",
  output_name = "AssociatedClusterNetwork",
  verbose = FALSE,
  ...
)
```

**Arguments**

- `file_list`: A character vector of file paths, or a list containing connections and file paths. Each element corresponds to a single file containing the data for a single sample. Passed to `loadDataFromFileList()`.
- `input_type`: A character string specifying the file format of the neighborhood data files. Options are "table", "txt", "tsv", "csv", "rds" and "rda". Passed to `loadDataFromFileList()`.
- `data_symbols`: Used when `input_type = "rda"`. Specifies the name of each neighborhood’s data frame within its respective Rdata file. Passed to `loadDataFromFileList()`.
buildAssociatedClusterNetwork

header
For values of input_type other than "rds" and "rda", this argument is used to specify the value of the header argument to read.table(), read.csv(), etc.

sep
For values of input_type other than "rds" and "rda", this argument can be used to specify a non-default value of the sep argument to read.table(), read.csv(), etc.

read.args
For values of input_type other than "rds" and "rda", this argument is used to specify values of optional arguments to read.table(), read.csv(), etc. Accepts a named list of argument values. Values of header and sep in this list take precedence over values specified via the header and sep arguments.

seq_col
Specifies the column of each neighborhood's data frame containing the TCR/BCR sequences. Accepts a character string containing the column name or a numeric scalar containing the column index.

min_seq_length
Passed to buildRepSeqNetwork() when constructing the global network.

drop_matches
Passed to buildRepSeqNetwork() when constructing the global network.

drop_isolated_nodes
Passed to buildRepSeqNetwork() when constructing the global network.

node_stats
Passed to buildRepSeqNetwork() when constructing the global network.

stats_to_include
Passed to buildRepSeqNetwork() when constructing the global network.

cluster_stats
Passed to buildRepSeqNetwork() when constructing the global network.

color_nodes_by
Passed to buildRepSeqNetwork() when constructing the global network.

output_name
Passed to buildRepSeqNetwork() when constructing the global network.

verbose
Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().

...
Other arguments to buildRepSeqNetwork() when constructing the global network.

Details

Each associated sequence's neighborhood contains clones (from all samples) with TCR/BCR sequences similar to the associated sequence. The neighborhoods are assumed to have been previously identified using findAssociatedClones().

The neighborhood data for all associated sequences are used to construct a single global network. Cluster analysis is used to partition the global network into clusters, which are considered as the associated TCR/BCR clusters. Network properties for the nodes and clusters are computed and returned as metadata. A plot of the global network graph is produced, with the nodes colored according to the binary variable of interest.

See the Searching for Associated TCR/BCR Clusters article on the package website for more details.

Value

A list of network objects as returned by buildRepSeqNetwork(). The list is returned invisibly. If the input data contains a combined total of fewer than two rows, or if the global network contains no nodes, then the function returns NULL, invisibly, with a warning.
Author(s)
Brian Neal (<Brian.Neal@ucsf.edu>)

References
Webpage for the NAIR package
Searching for Associated TCR/BCR Clusters article on package website

See Also
findAssociatedSeqs() findAssociatedClones()

Examples
set.seed(42)

## Simulate 30 samples from two groups (treatment/control) ##
 n_control <- n_treatment <- 15
 n_samples <- n_control + n_treatment
 sample_size <- 30 # (seqs per sample)
 base_seqs <- # first five are associated with treatment
  c("CASSGAYEQYF", "CSVDLGKGNNEQFF", "CASSIEQQLSTDTQYF",
   "CASSEEGQLSTDTQYF", "CASSPEGQLSTDTQYF",
   "RASSLAGNTEAFF", "CASSHRGTDTQYF", "CASDAGVFQPQHF")
 # Relative generation probabilities by control/treatment group
 pgen_c <- matrix(rep(c(rep(1, 5), rep(30, 3)), times = n_control),
   nrow = n_control, byrow = TRUE)
 pgen_t <- matrix(rep(c(1, 1, rep(1/3, 3), rep(2, 3)), times = n_treatment),
   nrow = n_treatment, byrow = TRUE)
 pgen <- rbind(pgen_c, pgen_t)
simulateToyData(
  samples = n_samples,
  sample_size = sample_size,
  prefix_length = 1,
  prefix_chars = c("", ""),
  prefix_probs = cbind(rep(1, n_samples), rep(0, n_samples)),
  affixes = base_seqs,
  affix_probs = pgen,
  num_edits = 0,
  output_dir = tempdir(),
  no_return = TRUE
)

## Step 1: Find Associated Sequences ##
sample_files <-
  file.path(tempdir(),
    paste0("Sample", 1:n_samples, ".rds")
  )
group_labels <- c(rep("reference", n_control),

  rep("experimental", n_treatment),
  paste0("control", 1:n_control, ".rds")
  )
rep("comparison", n_treatment))

associated_seqs <- findAssociatedSeqs(
  file_list = sample_files,
  input_type = "rds",
  group_ids = group_labels,
  seq_col = "CloneSeq",
  min_seq_length = NULL,
  drop_matches = NULL,
  min_sample_membership = 0,
  pval_cutoff = 0.1
)

head(associated_seqs[, 1:5])

## Step 2: Find Associated Clones ##
dir_step2 <- tempfile()

findAssociatedClones(
  file_list = sample_files,
  input_type = "rds",
  group_ids = group_labels,
  seq_col = "CloneSeq",
  assoc_seqs = associated_seqs$ReceptorSeq,
  min_seq_length = NULL,
  drop_matches = NULL,
  output_dir = dir_step2
)

## Step 3: Global Network of Associated Clusters ##

associated_clusters <- buildAssociatedClusterNetwork(
  file_list = list.files(dir_step2,
    full.names = TRUE
  ),
  seq_col = "CloneSeq",
  size_nodes_by = 1.5,
  print_plots = TRUE
)

---

**buildPublicClusterNetwork**

*Build Global Network of Public TCR/BCR Clusters*

**Description**

Part of the workflow *Searching for Public TCR/BCR Clusters*. Intended for use following `findPublicClusters()`.

Given node-level metadata for each sample’s filtered clusters, combines the data into a global network and performs network analysis and cluster analysis.
Usage

buildPublicClusterNetwork(

  ## Input ##
  file_list,
  input_type = "rds",
  data_symbols = "ndat",
  header = TRUE, sep,
  read.args = list(row.names = 1),
  seq_col,

  ## Network Settings ##
  drop_isolated_nodes = FALSE,
  node_stats = deprecated(),
  stats_to_include = deprecated(),
  cluster_stats = deprecated(),

  ## Visualization ##
  color_nodes_by = "SampleID",
  color_scheme = "turbo",
  plot_title = "Global Network of Public Clusters",

  ## Output ##
  output_dir = NULL,
  output_name = "PublicClusterNetwork",
  verbose = FALSE,

  ...
)

Arguments

file_list  A character vector of file paths, or a list containing connections and file paths. Each element corresponds to a single file containing the data for a single sample. Passed to `loadDataFromFileList()`.

input_type  A character string specifying the file format of the input files. Options are "csv", "rds" and "rda". Passed to `loadDataFromFileList()`.

data_symbols  Used when input_type = "rda". Specifies the name of the data frame within each Rdata file. Passed to `loadDataFromFileList()`.

header  For values of input_type other than "rds" and "rda", this argument can be used to specify a non-default value of the header argument to `read.table()`, `read.csv()`, etc.

sep  For values of input_type other than "rds" and "rda", this argument can be used to specify a non-default value of the sep argument to `read.table()`, `read.csv()`, etc.
read.args         For values of input_type other than "rds" and "rda", this argument can be used to specify non-default values of optional arguments to read.table(), read.csv(), etc. Accepts a named list of argument values. Values of header and sep in this list take precedence over values specified via the header and sep arguments.

seq_col           Specifies the column in the node-level metadata that contains the TCR/BCR sequences. Accepts a character string containing the column name or a numeric scalar containing the column index.

drop_isolated_nodes               Passed to buildRepSeqNetwork() when constructing the global network.

node_stats                         [Deprecated] All network properties are automatically computed.

stats_to_include                   [Deprecated] All network properties are automatically computed.

cluster_stats                      [Deprecated] All network properties are automatically computed.

color_nodes_by                     Passed to buildRepSeqNetwork() when constructing the global network. The node-level network properties for the global network (see details) are included among the valid options.

color_scheme                       Passed to addPlots() when constructing the global network.

plot_title                         Passed to buildRepSeqNetwork() when constructing the global network.

output_dir                         Passed to buildRepSeqNetwork() when constructing the global network.

output_name                        Passed to buildRepSeqNetwork() when constructing the global network.

verbose                            Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().

...                                Other arguments to buildRepSeqNetwork() (including arguments to addPlots()) when constructing the global network. Does not include node_stats, stats_to_include, cluster_stats or cluster_id_name.

Details

The node-level metadata for the filtered clusters from all samples is combined and the global network is constructed by calling buildNet() with node_stats = TRUE, stats_to_include = "all", cluster_stats = TRUE and cluster_id_name = "ClusterIDPublic".

The computed node-level network properties are renamed to reflect their correspondence to the global network. This is done to distinguish them from the network properties that correspond to the sample-level networks. The names are:

• ClusterIDPublic
• PublicNetworkDegree
• PublicTransitivity
• PublicCloseness
• PublicCentralityByCloseness
• PublicEigenCentrality
buildPublicClusterNetwork

- PublicCentralityByEigen
- PublicBetweenness
- PublicCentralityByBetweenness
- PublicAuthorityScore
- PublicCoreness
- PublicPageRank

See the Searching for Public TCR/BCR Clusters article on the package website.

Value

A list of network objects as returned by buildRepSeqNetwork(). The list is returned invisibly. If the input data contains a combined total of fewer than two rows, or if the global network contains no nodes, then the function returns NULL, invisibly, with a warning.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package
Searching for Public TCR/BCR Clusters article on package website

See Also

findPublicClusters()
buildPublicClusterNetworkByRepresentative()

Examples

c(“CASSIEGLSTDTQYF”, “CASSEEGQLSTDTQYF”, “CASSSVETQYF”,
“CASSPEGQLSTDTQYF”, “RASSLAGNTEAFF”, “CASSHRGTDTQYF”, “CASDAGVFQPQHF”,
“CASSLSGYNQEFQ”, “CASSETGYNEQFF”, “CASSLTGNEQFF”, “CASSYLTGNEQFF”,
“CASSLTGNEQFF”, “CASSLNQGEQFF”, “CASSFPWGDYQYTF”, “CASTLRQGGELEFTF”,
“CASTLSTQGGELEFTF”, “CSVELLPTGPLETSYNEQFF”, “CSVELLPTGPLETSYNEQFF”,
“CVELLPTGPLETSYNEQFF”, “CASLAGRGTQETQYF”, “CASLAGRGTQETQYF”,
“CASSLAGKRTETQYF”, “CASSLAGKRTETQYF”, “CASSRLAGRTQETQYF”,
“CASQYGGGGNQPQHF”, “CASSLGQGGNQPQHF”, “CASSNGGGNQPQHF”, “CASSYGGGQNGPQHF”,
“CASSYGGGGQNPQHF”, “CASSYGGGGQNPQHF”, “CASSYGQGQNPQHF”,
“CAWSSQETQYF”, “CAWWSSPETQYF”, “CASSGAYEQYF”, “CSVDLGKGNNEQFF”)

set.seed(42)

## Simulate 30 samples with a mix of public/private sequences ##
samples <- 30
sample_size <- 30 # (seqs per sample)
base_segs <- c(
“CASSIEGLSTDTQYF”, “CASSEEGQLSTDTQYF”, “CASSSVETQYF”,
“CASSPEGQLSTDTQYF”, “RASSLAGNTEAFF”, “CASSHRGTDTQYF”, “CASDAGVFQPQHF”,
“CASSLSGYNQEFQ”, “CASSETGYNEQFF”, “CASSLTGNEQFF”, “CASSYLTGNEQFF”,
“CASSLTGNEQFF”, “CASSLNQGEQFF”, “CASSFPWGDYQYTF”, “CASTLRQGGELEFTF”,
“CASTLSTQGGELEFTF”, “CSVELLPTGPLETSYNEQFF”, “CSVELLPTGPLETSYNEQFF”,
“CVELLPTGPLETSYNEQFF”, “CASLAGRGTQETQYF”, “CASLAGRGTQETQYF”,
“CASSLAGKRTETQYF”, “CASSLAGKRTETQYF”, “CASSRLAGRTQETQYF”,
“CASQYGGGGNQPQHF”, “CASSLGQGGNQPQHF”, “CASSNGGGNQPQHF”, “CASSYGGGQNGPQHF”,
“CASSYGGGGQNPQHF”, “CASSYGGGGQNPQHF”, “CASSYGQGQNPQHF”,
“CAWSSQETQYF”, “CAWWSSPETQYF”, “CASSGAYEQYF”, “CSVDLGKGNNEQFF”)}
# Relative generation probabilities
pgen <- cbind(
  stats::toeplitz(0.6^0:(sample_size - 1)),
  matrix(1, nrow = samples, ncol = length(base_seqs) - samples)
)
simulateToyData(
  samples = samples,
  sample_size = sample_size,
  prefix_length = 1,
  prefix_chars = c("", ""),
  prefix_probs = cbind(rep(1, samples), rep(0, samples)),
  affixes = base_seqs,
  affix_probs = pgen,
  num_edits = 0,
  output_dir = tempdir(),
  no_return = TRUE
)

## 1. Find Public Clusters in Each Sample
sample_files <-
  file.path(tempdir(),
    paste0("Sample", 1:samples, ".rds")
  )
findPublicClusters(
  file_list = sample_files,
  input_type = "rds",
  seq_col = "CloneSeq",
  count_col = "CloneCount",
  min_seq_length = NULL,
  drop_matches = NULL,
  top_n_clusters = 3,
  min_node_count = 5,
  min_clone_count = 15000,
  output_dir = tempdir()
)

## 2. Build Global Network of Public Clusters
public_clusters <-
buildPublicClusterNetwork(
  file_list =
    list.files(
      file.path(tempdir(), "node_meta_data"),
      full.names = TRUE
    ),
  seq_col = "CloneSeq",
  count_col = "CloneCount",
  plot_title = NULL,
  plot_subtitle = NULL,
  print_plots = TRUE
)
buildPublicClusterNetworkByRepresentative

Build Global Network of Public TCR/BCR Clusters Using Representative Clones

Description

Alternative step in the workflow Searching for Public TCR/BCR Clusters. Intended for use following findPublicClusters() in cases where buildPublicClusterNetwork() cannot be practically used due to the size of the full global network.

Given cluster-level metadata for each sample’s filtered clusters, selects a representative TCR/BCR from each cluster, combines the representatives into a global network and performs network analysis and cluster analysis.

Usage

```r
buildPublicClusterNetworkByRepresentative(

  ## Input ##
  file_list,
  input_type = "rds",
  data_symbols = "cdat",
  header, sep, read.args,
  seq_col = "seq_w_max_count",
  count_col = "agg_count",

  ## Network Settings ##
  dist_type = "hamming",
  dist_cutoff = 1,
  cluster_fun = "fast_greedy",

  ## Visualization ##
  plots = TRUE,
  print_plots = FALSE,
  plot_title = "auto",
  plot_subtitle = "auto",
  color_nodes_by = "SampleID",
  color_scheme = "turbo",
  ...

  ## Output ##
  output_dir = NULL,
  output_type = "rds",
  output_name = "PubClustByRepresentative",
  pdf_width = 12,
)
```
buildPublicClusterNetworkByRepresentative

```r
pdf_height = 10,
verbose = FALSE
)
```

**Arguments**

- `file_list` A vector of file paths where each file contains the cluster-level metadata for one sample's filtered clusters. Passed to `loadDataFromFileList()`.
- `input_type` A character string specifying the file format of the input files. Options are "csv", "rds" and "rda". Passed to `loadDataFromFileList()`.
- `data_symbols` Used when `input_type = "rda"`. Specifies the name of the data frame within each Rdata file. Passed to `loadDataFromFileList()`.
- `header` For values of `input_type` other than "rds" and "rda", this argument can be used to specify a non-default value of the header argument to `read.table()`, `read.csv()`, etc.
- `sep` For values of `input_type` other than "rds" and "rda", this argument can be used to specify a non-default value of the sep argument to `read.table()`, `read.csv()`, etc.
- `read.args` For values of `input_type` other than "rds" and "rda", this argument can be used to specify non-default values of optional arguments to `read.table()`, `read.csv()`, etc. Accepts a named list of argument values. Values of `header` and `sep` in this list take precedence over values specified via the header and sep arguments.
- `seq_col` Specifies the column in the cluster-level metadata that contains the representative TCR/BCR sequence for each cluster. Accepts a character string containing the column name or a numeric scalar containing the column index. By default, uses the sequence with the maximum clone count in each cluster.
- `count_col` Specifies the column in the cluster-level metadata that contains the aggregate clone count for each cluster. Accepts a character string containing the column name or a numeric scalar containing the column index.
- `dist_type` Passed to `buildRepSeqNetwork()` when constructing the global network.
- `dist_cutoff` Passed to `buildRepSeqNetwork()` when constructing the global network.
- `cluster_fun` Passed to `buildRepSeqNetwork()` when performing cluster analysis on the global network.
- `plots` Logical. Should plots of the global network graph be produced?
- `print_plots` Logical. If plots of the global network graph are produced, should they be printed to the R plotting window?
- `plot_title` Passed to `addPlots()` when producing plots of the global network graph.
- `plot_subtitle` Passed to `addPlots()` when producing plots of the global network graph.
- `color_nodes_by` Passed to `addPlots()` when producing plots of the global network graph. Valid options include the default "SampleID", as well as node-level properties (see `addNodeNetworkStats`) and sample-level cluster properties (see `getClusterStats`), which correspond to the representative TCRs/BCRs and the original sample-level clusters they represent, respectively.
**buildPublicClusterNetworkByRepresentative**

```
color_scheme  Passed to `addPlots()` when producing plots of the global network graph.
...            Other arguments to `addPlots()` when producing plots of the global network graph.
output_dir    Passed to `saveNetwork()` after constructing the global network.
output_type   Passed to `saveNetwork()` after constructing the global network.
output_name   Passed to `saveNetwork()` after constructing the global network.
pdf_width     Passed to `saveNetwork()` after constructing the global network. Only applicable if `plots = TRUE`.
pdf_height    Passed to `saveNetwork()` after constructing the global network. Only applicable if `plots = TRUE`.
verbose       Logical. If `TRUE`, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to `stderr()`.
```

**Details**

From each filtered cluster in each sample’s network, a representative TCR/BCR is selected. By default, this is the sequence with the greatest clone count in each cluster. The representatives from all clusters and all samples are then used to construct a single global network. Cluster analysis is used to partition this global network into clusters. Network properties for the nodes and clusters are computed and returned as metadata. A plot of the global network graph is produced, with the nodes colored according to sample ID.

Within this network, clusters containing nodes from multiple samples can be considered as the skeletons of the complete public clusters. The filtered cluster data for each sample can then be subset to keep the sample-level clusters whose representative TCR/BCRs belong to the skeletons of the public clusters. After subsetting in this manner, `buildPublicClusterNetwork()` can be used to construct the global network of complete public clusters.

See the [Searching for Public TCR/BCR Clusters](https://example.com) article on the package website.

**Value**

If the input data contains a combined total of fewer than two rows, or if the global network contains no nodes, then the function returns `NULL`, invisibly, with a warning. Otherwise, invisibly returns a list of network objects as returned by `buildRepSeqNetwork()`. The global cluster membership variable in the data frame `node_data` is named `ClusterIDPublic`.

The data frame `cluster_data` includes the following variables that represent properties of the clusters in the global network of representative TCR/BCRs:

- `cluster_id`  The global cluster ID number.
- `node_count`  The number of global network nodes in the global cluster.
- `TotalSampleLevelNodes`  For each representative TCR/BCR in the global cluster, we record the number of nodes in the sample-level cluster for which it is the representative TCR/BCR. We then sum these node counts across all the representative TCR/BCRs in the global cluster.
TotalCloneCount
For each representative TCR/BCR in the global cluster, we record the aggregate clone count from all nodes in the sample-level cluster for which it is the representative TCR/BCR. We then sum these aggregate clone counts across all the representative TCR/BCRs in the global cluster.

MeanOfMeanSeqLength
For each representative TCR/BCR in the global cluster, we record the mean sequence length over all clones (nodes) in the sample-level cluster for which it is the representative TCR/BCR. We then average these mean sequence lengths over all the representative TCR/BCRs in the global cluster.

MeanDegreeInPublicNet
For each representative TCR/BCR in the global cluster, we record the mean network degree over all nodes in the sample-level cluster for which it is the representative TCR/BCR. We then average these mean degree values over all the representative TCR/BCRs in the global cluster.

MaxDegreeInPublicNet
For each representative TCR/BCR in the global cluster, we record the maximum network degree across all nodes in the sample-level cluster for which it is the representative TCR/BCR. We then take the maximum of these maximum degree values over all the representative TCR/BCRs in the global cluster.

SeqWithMaxDegree
For each representative TCR/BCR in the global cluster, we record the maximum network degree across all nodes in the sample-level cluster for which it is the representative TCR/BCR. We then identify the representative TCR/BCR with the maximum value of these maximum degrees over all the representative TCR/BCRs in the global cluster. The TCR/BCR sequence of the identified representative TCR/BCR is recorded in this variable.

MaxCloneCount
For each representative TCR/BCR in the global cluster, we record the maximum clone count across all clones (nodes) in the sample-level cluster for which it is the representative TCR/BCR. We then take the maximum of these maximum clone counts over all the representative TCR/BCRs in the global cluster.

SampleWithMaxCloneCount
For each representative TCR/BCR in the global cluster, we record the maximum clone count across all clones (nodes) in the sample-level cluster for which it is the representative TCR/BCR. We then identify the representative TCR/BCR with the maximum value of these maximum clone counts over all the representative TCR/BCRs in the global cluster. The sample to which the identified representative TCR/BCR belongs is recorded in this variable.

SeqWithMaxCloneCount
For each representative TCR/BCR in the global cluster, we record the maximum clone count across all clones (nodes) in the sample-level cluster for which it is the representative TCR/BCR. We then identify the representative TCR/BCR with the maximum value of these maximum clone counts over all the representative TCR/BCRs in the global cluster. The TCR/BCR sequence of the identified representative TCR/BCR is recorded in this variable.

MaxAggCloneCount
For each representative TCR/BCR in the global cluster, we record the aggregate clone count across all clones (nodes) in the sample-level cluster for which it
is the representative TCR/BCR. We then take the maximum of these aggregate clone counts over all the representative TCR/BCRs in the global cluster.

**SampleWithMaxAggCloneCount**
For each representative TCR/BCR in the global cluster, we record the aggregate clone count across all clones (nodes) in the sample-level cluster for which it is the representative TCR/BCR. We then identify the representative TCR/BCR with the maximum value of these aggregate clone counts over all the representative TCR/BCRs in the global cluster. The sample to which the identified representative TCR/BCR belongs is recorded in this variable.

**SeqWithMaxAggCloneCount**
For each representative TCR/BCR in the global cluster, we record the aggregate clone count across all clones (nodes) in the sample-level cluster for which it is the representative TCR/BCR. We then identify the representative TCR/BCR with the maximum value of these aggregate clone counts over all the representative TCR/BCRs in the global cluster. The TCR/BCR sequence of the identified representative TCR/BCR is recorded in this variable.

**DiameterLength**
See `getClusterStats`. Based on edge connections between representative TCR/BCRs in the global cluster.

**Assortativity**
See `getClusterStats`. Based on edge connections between representative TCR/BCRs in the global cluster.

**GlobalTransitivity**
See `getClusterStats`. Based on edge connections between representative TCR/BCRs in the global cluster.

**EdgeDensity**
See `getClusterStats`. Based on edge connections between representative TCR/BCRs in the global cluster.

**DegreeCentralityIndex**
See `getClusterStats`. Based on edge connections between representative TCR/BCRs in the global cluster.

**ClosenessCentralityIndex**
See `getClusterStats`. Based on edge connections between representative TCR/BCRs in the global cluster.

**EigenCentralityIndex**
See `getClusterStats`. Based on edge connections between representative TCR/BCRs in the global cluster.

**EigenCentralityEigenvalue**
See `getClusterStats`. Based on edge connections between representative TCR/BCRs in the global cluster.

**Author(s)**
Brian Neal (<Brian.Neal@ucsf.edu>)

**References**
Webpage for the NAIR package
Searching for Public TCR/BCR Clusters article on package website

See Also

findPublicClusters() buildPublicClusterNetwork()

Examples

set.seed(42)

## Simulate 30 samples with a mix of public/private sequences ##
samples <- 30
sample_size <- 30 # (seqs per sample)
base_seqs <- c(
  "CASSIEGQLSTDQYF", "CASSEEQQLSTDQYF", "CASSSVETQYF",
  "CASSPEGQLSTDQYF", "RASSLAGNEAFF", "CASSHRGTDQYF", "CASDAGVFQPQHF",
  "CASSLTGYNQEF", "CASETGYNQEF", "CASLGTHGNEQFF", "CASSYLTGYNQEF",
  "CASSLGNQEF", "CASLNGYNQEF", "CASSFPGDGYQTF", "CASTLRQGGELFF",
  "CASTLARQGGELFF", "CVELLPTGPLETSYQEF", "CSVELLPTGPLETSYQEF",
  "CVELLPTGPLETSYQEF", "CASSLAGRGTQETQYF", "CASRLAGRGTQETQYF",
  "CASLAGRGTQETQYF", "CASSLAGRGTQETQYF", "CASRLAGRGTQETQYF",
  "CASQYGGGNQPQHF", "CASLGGGNQPQHF", "CASSNGGGNQPQHF", "CASSYGGGNQPQHF",
  "CASSYGGQPPQHF", "CASSYKGGNPQHF", "CASSYGGQNQPOHF", "CASSYGGQNPQHF",
  "CASSYGGQNPQHF", "CASSYGGQNPQHF", "CASSYGGQNPQHF", "CASSYGGQNPQHF",
  "CASSYGGQNPQHF", "CASSYGGQNPQHF")
# Relative generation probabilities
pgen <- cbind(  
  stats::toeplitz(0.6^((0:(sample_size - 1)))),
  matrix(1, nrow = samples, ncol = length(base_seqs) - samples)
)
simulateToyData(
  samples = samples,
  sample_size = sample_size,
  prefix_length = 1,
  prefix_chars = c("", ""),
  prefix_probs = cbind(rep(1, samples), rep(0, samples)),
  affixes = base_seqs,
  affix_probs = pgen,
  num_edits = 0,
  output_dir = tempdir(),
  no_return = TRUE
)

## 1. Find Public Clusters in Each Sample
sample_files <-
  file.path(tempdir(),
    paste0("Sample", 1:samples, ".rds")
)
findPublicClusters(
  file_list = sample_files,
  input_type = "rds",
)
buildRepSeqNetwork  
Network Analysis of Immune Repertoire  

Description  
Given Adaptive Immune Receptor Repertoire Sequencing (AIRR-Seq) data, builds the network graph for the immune repertoire based on sequence similarity, computes specified network properties and generates customized visualizations.

buildNet() is identical to buildRepSeqNetwork(), existing as an alias for convenience.

Usage  
buildRepSeqNetwork(
    
    # Input #
    data,  
    seq.col,  
    count.col = NULL,  
    subset.cols = NULL,  
    min_seq_length = 3,  
    drop_matches = NULL,  

    # Network #
    dist_type = "hamming",  
    dist_cutoff = 1,  

)  

## 2. Build Public Cluster Network by Representative TCR/BCRs  
buildPublicClusterNetworkByRepresentative(
    file_list =  
        list.files(  
            file.path(tempdir(), "cluster_meta_data"),  
            full.names = TRUE  
        ),  
    size_nodes_by = 1,  
    print_plots = TRUE  
)
drop_isolated_nodes = TRUE,
node_stats = FALSE,
stats_to_include = chooseNodeStats(),
cluster_stats = FALSE,
cluster_fun = "fast_greedy",
cluster_id_name = "cluster_id",

## Visualization ##
plots = TRUE,
print_plots = FALSE,
plot_title = "auto",
plot_subtitle = "auto",
color_nodes_by = "auto",
...

## Output ##
output_dir = NULL,
output_type = "rds",
output_name = "MyRepSeqNetwork",
pdf_width = 12,
pdf_height = 10,
verbose = FALSE

# Alias for buildRepSeqNetwork()
buildNet(
  data,
  seq_col,
  count_col = NULL,
  subset_cols = NULL,
  min_seq_length = 3,
  drop_matches = NULL,
  dist_type = "hamming",
  dist_cutoff = 1,
  drop_isolated_nodes = TRUE,
  node_stats = FALSE,
  stats_to_include = chooseNodeStats(),
  cluster_stats = FALSE,
  cluster_fun = "fast_greedy",
  cluster_id_name = "cluster_id",
  plots = TRUE,
  print_plots = FALSE,
  plot_title = "auto",
  plot_subtitle = "auto",
color_nodes_by = "auto",
...,
  output_dir = NULL,
Arguments

data

A data frame containing the AIRR-Seq data, with variables indexed by column and observations (e.g., clones or cells) indexed by row.

seq_col

Specifies the column(s) of data containing the receptor sequences to be used as the basis of similarity between rows. Accepts a character string containing the column name or a numeric scalar containing the column index. Also accepts a vector of length 2 specifying distinct sequence columns (e.g., alpha chain and beta chain), in which case similarity between rows depends on similarity in both sequence columns (see details).

count_col

Optional. Specifies the column of data containing a measure of abundance, e.g., clone count or unique molecular identifier (UMI) count. Accepts either the column name or column index. Passed to addClusterStats(); only relevant if cluster_stats = TRUE.

subset_cols

Specifies which columns of the AIRR-Seq data are included in the output. Accepts a vector of column names or a vector of column indices. The default NULL includes all columns. The receptor sequence column is always included regardless of this argument’s value. Passed to filterInputData().

min_seq_length

A numeric scalar, or NULL. Observations whose receptor sequences have fewer than min_seq_length characters are removed prior to network analysis.

drop_matches

Optional. Passed to filterInputData(). Accepts a character string containing a regular expression (see regex). Checks receptor sequences for a pattern match using grep(). Those returning a match are removed prior to network analysis.

dist_type

Specifies the function used to quantify the similarity between sequences. The similarity between two sequences determines the pairwise distance between their respective nodes in the network graph, with greater similarity corresponding to shorter distance. Valid options are "hamming" (the default), which uses hamDistBounded(), and "levenshtein", which uses levDistBounded().

dist_cutoff

A nonnegative scalar. Specifies the maximum pairwise distance (based on dist_type) for an edge connection to exist between two nodes. Pairs of nodes whose distance is less than or equal to this value will be joined by an edge connection in the network graph. Controls the stringency of the network construction and affects the number and density of edges in the network. A lower cutoff value requires greater similarity between sequences in order for their respective nodes to be joined by an edge connection. A value of 0 requires two sequences to be identical in order for their nodes to be joined by an edge.
buildRepSeqNetwork

drop_isolated_nodes
  A logical scalar. When TRUE, removes each node that is not joined by an edge connection to any other node in the network graph.

node_stats
  A logical scalar. Specifies whether node-level network properties are computed.

stats_to_include
  A named logical vector returned by chooseNodeStats() or exclusiveNodeStats(). Specifies the node-level network properties to compute. Also accepts the value "all". Only relevant if node_stats = TRUE.

cluster_stats
  A logical scalar. Specifies whether to compute cluster-level network properties.

cluster_fun
  Passed to addClusterMembership(). Specifies the clustering algorithm used when cluster analysis is performed. Cluster analysis is performed when cluster_stats = TRUE or when node_stats = TRUE with the cluster_id property enabled via the stats_to_include argument.

cluster_id_name
  Passed to addClusterMembership(). Specifies the name of the cluster membership variable added to the node metadata when cluster analysis is performed (see cluster_fun).

plots
  A logical scalar. Specifies whether to generate plots of the network graph.

print_plots
  A logical scalar. If plots = TRUE, specifies whether the plots should be printed to the R plotting window.

plot_title
  A character string or NULL. If plots = TRUE, this is the title used for each plot. The default value "auto" generates the title based on the value of the output_name argument.

plot_subtitle
  A character string or NULL. If plots = TRUE, this is the subtitle used for each plot. The default value "auto" generates a subtitle based on the values of the dist_type and dist_cutoff arguments.

color_nodes_by
  Optional. Specifies a variable to be used as metadata for coloring the nodes in the network graph plot. Accepts a character string. This can be a column name of data or (if node_stats = TRUE) the name of a computed node-level network property (based on stats_to_include). Also accepts a character vector specifying multiple variables, in which case one plot will be generated for each variable. The default value "auto" attempts to use one of several potential variables to color the nodes, depending on what is available. A value of NULL leaves the nodes uncolored.

... Other named arguments to addPlots().

output_dir
  A file path specifying the directory for saving the output. The directory will be created if it does not exist. If NULL, output will be returned but not saved.

output_type
  A character string specifying the file format to use when saving the output. The default value "individual" saves each element of the returned list as an individual uncompressed file, with data frames saved in csv format. For better compression, the values "rda" and "rds" save the returned list as a single file using the rda and rds format, respectively (in the former case, the list will be named net within the rda file). Regardless of the argument value, any plots generated will saved to a pdf file containing one plot per page.

output_name
  A character string. All files saved will have file names beginning with this value.
buildRepSeqNetwork

pdf_width
Sets the width of each plot when writing to pdf. Passed to `saveNetwork()`.

pdf_height
Sets the height of each plot when writing to pdf. Passed to `saveNetwork()`.

verbose
Logical. If `TRUE`, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to `stderr()`.

Details

To construct the immune repertoire network, each TCR/BCR clone (bulk data) or cell (single-cell data) is modeled as a node in the network graph, corresponding to a single row of the AIRR-Seq data. For each node, the corresponding receptor sequence is considered. Both nucleotide and amino acid sequences are supported for this purpose. The receptor sequence is used as the basis of similarity and distance between nodes in the network.

Similarity between sequences is measured using either the Hamming distance or Levenshtein (edit) distance. The similarity determines the pairwise distance between nodes in the network graph. The more similar two sequences are, the shorter the distance between their respective nodes. Two nodes in the graph are joined by an edge if the distance between them is sufficiently small, i.e., if their receptor sequences are sufficiently similar.

For single-cell data, edge connections between nodes can be based on similarity in both the alpha chain and beta chain sequences. This is done by providing a vector of length 2 to `seq_cols` specifying the two sequence columns in `data`. The distance between two nodes is then the greater of the two distances between sequences in corresponding chains. Two nodes will be joined by an edge if their alpha chain sequences are sufficiently similar and their beta chain sequences are sufficiently similar.

See the `buildRepSeqNetwork` package vignette for more details. The vignette can be accessed offline using `vignette("buildRepSeqNetwork")`.

Value

If the constructed network contains no nodes, the function will return `NULL`, invisibly, with a warning. Otherwise, the function invisibly returns a list containing the following items:

- `details` A list containing information about the network and the settings used during its construction.
- `igraph` An object of class `igraph` containing the list of nodes and edges for the network graph.
- `adjacency_matrix` The network graph adjacency matrix, stored as a sparse matrix of class `dgCMatrix` from the `Matrix` package. See `dgCMatrix-class`.
- `node_data` A data frame containing containing metadata for the network nodes, where each row corresponds to a node in the network graph. This data frame contains all variables from `data` (unless otherwise specified via `subset_cols`) in addition to the computed node-level network properties if `node_stats = TRUE`. Each row’s name is the name of the corresponding row from `data`.
- `cluster_data` A data frame containing network properties for the clusters, where each row corresponds to a cluster in the network graph. Only included if `cluster_stats = TRUE`. 

plots

A list containing one element for each plot generated as well as an additional element for the matrix that specifies the graph layout. Each plot is an object of class `ggraph`. Only included if `plots = TRUE`.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package

buildRepSeqNetwork vignette

Examples

```r
set.seed(42)
toy_data <- simulateToyData()

# Simple call
network = buildNet(
  toy_data,
  seq_col = "CloneSeq",
  print_plots = TRUE
)

# Customized:
network <- buildNet(
  toy_data, "CloneSeq",
  dist_type = "levenshtein",
  node_stats = TRUE,
  cluster_stats = TRUE,
  cluster_fun = "louvain",
  cluster_id_name = "cluster_membership",
  count_col = "CloneCount",
  color_nodes_by = c("SampleID", "cluster_membership", "coreness"),
  color_scheme = c("default", "Viridis", "plasma-1"),
  size_nodes_by = "degree",
  node_size_limits = c(0.1, 1.5),
  plot_title = NULL,
  plot_subtitle = NULL,
  print_plots = TRUE,
  verbose = TRUE
)

typeof(network)
names(network)
network$details
```
chooseNodeStats

head(network$node_data)

head(network$cluster_data)

---

**chooseNodeStats**

**Specify Node-level Network Properties to Compute**

**Description**

Create a vector specifying node-level network properties to compute. Intended for use with `buildRepSeqNetwork()` or `addNodeNetworkStats`.

`node_stat_settings()` is a deprecated equivalent of `chooseNodeStats()`.

**Usage**

```r
degree = TRUE,
cluster_id = FALSE,
transitivity = TRUE,
closeness = FALSE,
centrality_by_closeness = FALSE,
eigen_centrality = TRUE,
centrality_by_eigen = TRUE,
betweenness = TRUE,
centrality_by_betweenness = TRUE,
authority_score = TRUE,
coreness = TRUE,
page_rank = TRUE,
all_stats = FALSE
```

```r
degree = FALSE,
cluster_id = FALSE,
transitivity = FALSE,
closeness = FALSE,
centrality_by_closeness = FALSE,
eigen_centrality = FALSE,
centrality_by_eigen = FALSE,
betweenness = FALSE,
centrality_by_betweenness = FALSE,
authority_score = FALSE,
coreness = FALSE,
page_rank = FALSE
```
chooseNodeStats

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree</td>
<td>Logical. Whether to compute network degree.</td>
</tr>
<tr>
<td>cluster_id</td>
<td>Logical. Whether to perform cluster analysis and record the cluster membership of each node. See addClusterMembership().</td>
</tr>
<tr>
<td>transitivity</td>
<td>Logical. Whether to compute node-level network transitivity using transitivity() with type = &quot;local&quot;. The local transitivity of a node is the the number of triangles connected to the node relative to the number of triples centered on that node.</td>
</tr>
<tr>
<td>closeness</td>
<td>Logical. Whether to compute network closeness using closeness().</td>
</tr>
<tr>
<td>centrality_by_closeness</td>
<td>Logical. Whether to compute network centrality by closeness. The values are the entries of the res element of the list returned by centr_clo().</td>
</tr>
<tr>
<td>eigen_centrality</td>
<td>Logical. Whether to compute the eigenvector centrality scores of node network positions. The scores are the entries of the vector element of the list returned by eigen_centrality() with weights = NA. The centrality scores correspond to the values of the first eigenvector of the adjacency matrix for the cluster graph.</td>
</tr>
<tr>
<td>centrality_by_eigen</td>
<td>Logical. Whether to compute node-level network centrality scores based on eigenvector centrality scores. The scores are the entries of the vector element of the list returned by centr_eigen().</td>
</tr>
<tr>
<td>betweenness</td>
<td>Logical. Whether to compute network betweenness using betweenness().</td>
</tr>
<tr>
<td>centrality_by_betweenness</td>
<td>Logical. Whether to compute network centrality scores by betweenness. The scores are the entries of the res element of the list returned by centr_betw().</td>
</tr>
<tr>
<td>authority_score</td>
<td>Logical. Whether to compute the authority score using authority_score().</td>
</tr>
<tr>
<td>coreness</td>
<td>Logical. Whether to compute network coreness using coreness().</td>
</tr>
<tr>
<td>page_rank</td>
<td>Logical. Whether to compute page rank. The page rank values are the entries of the vector element of the list returned by page_rank().</td>
</tr>
<tr>
<td>all_stats</td>
<td>Logical. If TRUE, all other argument values are overridden and set to TRUE.</td>
</tr>
</tbody>
</table>

Details

These functions return a vector that can be passed to the stats_to_include argument of addNodeStats() (or buildRepSeqNetwork(), if node_stats = TRUE) in order to specify which node-level network properties to compute.

chooseNodeStats and exclusiveNodeStats each have default argument values suited to a different use case, in order to reduce the number of argument values that must be set manually.

chooseNodeStats has most arguments TRUE by default. It is best suited for including a majority of the available properties. It can be called with all_stats = TRUE to set all values to TRUE.

exclusiveNodeStats has all of its arguments set to FALSE by default. It is best suited for including only a few properties.
chooseNodeStats

Value
A named logical vector with one entry for each of the function’s arguments (except for all_stats).
Each entry has the same name as the corresponding argument, and its value matches the argument’s value.

Author(s)
Brian Neal (<Brian.Neal@ucsf.edu>)

References
Webpage for the NAIR package

See Also
addNodeStats()

Examples

```r
set.seed(42)
toy_data <- simulateToyData()

net <- generateNetworkObjects(
toy_data, "CloneSeq"
)

# Add default set of node properties
net <- addNodeStats(net)

# Modify default set of node properties
net <- addNodeStats(
  net,
  stats_to_include =
    chooseNodeStats(
      closeness = TRUE,
      page_rank = FALSE
    )
)

# Add only the specified node properties
net <- addNodeStats(
  net,
  stats_to_include =
    exclusiveNodeStats(
      degree = TRUE,
      transitivity = TRUE
    )
)
```
# Add all node-level network properties
net <- addNodeStats(
  net,
  stats_to_include = "all"
)

## combineSamples

### Description

Given multiple data frames stored in separate files, `loadDataFromFileList()` loads and combines them into a single data frame.

`combineSamples()` has the same default behavior as `loadDataFromFileList()`, but possesses additional arguments that allow the data frames to be filtered, subsetted and augmented with sample-level variables before being combined.

### Usage

```r
loadDataFromFileList(
  file_list,
  input_type,
  data_symbols = NULL,
  header, sep, read.args
)
```

```r
combineSamples(
  file_list,
  input_type,
  data_symbols = NULL,
  header, sep, read.args,
  seq_col = NULL,
  min_seq_length = NULL,
  drop_matches = NULL,
  subset_cols = NULL,
  sample_ids = NULL,
  subject_ids = NULL,
  group_ids = NULL,
  verbose = FALSE
)
```

### Arguments

- **file_list**: A character vector of file paths, or a list containing `connections` and file paths. Each element corresponds to a single file containing the data for a single sample.
**input_type**  
A character string specifying the file format of the sample data files. Options are "rds", "rda", "csv", "csv2", "tsv", "table". See details.

**data_symbols**  
Used when input_type = "rda". Specifies the name of each sample’s data frame within its respective RData file. Accepts a character vector of the same length as file_list. Alternatively, a single character string can be used if all data frames have the same name.

**header**  
For values of input_type other than "rds" and "rda", this argument can be used to specify a non-default value of the header argument to read.table(), read.csv(), etc.

**sep**  
For values of input_type other than "rds" and "rda", this argument can be used to specify a non-default value of the sep argument to read.table(), read.csv(), etc.

**read.args**  
For values of input_type other than "rds" and "rda", this argument can be used to specify non-default values of optional arguments to read.table(), read.csv(), etc. Accepts a named list of argument values. Values of header and sep in this list take precedence over values specified via the header and sep arguments.

**seq_col**  
If provided, each sample’s data will be filtered based on the values of min_seq_length and drop_matches. Passed to filterInputData() for each sample.

**min_seq_length**  
Passed to filterInputData() for each sample.

**drop_matches**  
Passed to filterInputData() for each sample.

**subset_cols**  
Passed to filterInputData() for each sample.

**sample_ids**  
A character or numeric vector of sample IDs, whose length matches that of file_list.

**subject_ids**  
An optional character or numeric vector of subject IDs, whose length matches that of file_list. Used to assign a subject ID to each sample.

**group_ids**  
A character or numeric vector of group IDs whose length matches that of file_list. Used to assign each sample to a group.

**verbose**  
Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().

**Details**

Each file is assumed to contain the data for a single sample, with observations indexed by row, and with the same columns across samples.

Valid options for input_type (and the corresponding function used to load each file) include:

- "rds": readRDS()
- "rda": load()
- "csv": read.csv()
- "csv2": read.csv2()
combineSamples

• "tsv": read.delim()
• "table": read.table()

If input_type = "rda", the data_symbols argument specifies the name of each data frame within its respective file.

When calling combineSamples(), for each of sample_ids, subject_ids and group_ids that is non-null, a corresponding variable will be added to the combined data frame; these variables are named SampleID, SubjectID and GroupID.

Value

A data frame containing the combined data rows from all files.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package

Examples

# Generate example data
set.seed(42)
samples <- simulateToyData(sample_size = 5)
sample_1 <- subset(samples, SampleID == "Sample1")
sample_2 <- subset(samples, SampleID == "Sample2")

# RDS format
rdsfiles <- tempfile(c("sample1", "sample2"), fileext = ".rds")
saveRDS(sample_1, rdsfiles[1])
saveRDS(sample_2, rdsfiles[2])

loadDataFromFileList(
  rdsfiles,
  input_type = "rds"
)

# With filtering and subsetting
combineSamples(
  rdsfiles,
  input_type = "rds",
  seq_col = "CloneSeq",
  min_seq_length = 13,
  drop_matches = "GGG",
  subset_cols = "CloneSeq",
  sample_ids = c("id01", "id02"),
  verbose = TRUE"
# RData, different data frame names
rdafiles <- tempfile(c("sample1", "sample2"), fileext = ".rda")
save(sample_1, file = rdafiles[1])
save(sample_2, file = rdafiles[2])
loadDataFromFileList(
  rdafiles,
  input_type = "rda",
  data_symbols = c("sample_1", "sample_2")
)

# RData, same data frame names
df <- sample_1
save(df, file = rdafiles[1])
df <- sample_2
save(df, file = rdafiles[2])
loadDataFromFileList(
  rdafiles,
  input_type = "rda",
  data_symbols = "df"
)

# comma-separated values with header row; row names in first column
csvfiles <- tempfile(c("sample1", "sample2"), fileext = ".csv")
utils::write.csv(sample_1, csvfiles[1], row.names = TRUE)
utils::write.csv(sample_2, csvfiles[2], row.names = TRUE)
loadDataFromFileList(
  csvfiles,
  input_type = "csv",
  read.args = list(row.names = 1)
)

# semicolon-separated values with decimals as commas;
# header row, row names in first column
utils::write.csv2(sample_1, csvfiles[1], row.names = TRUE)
utils::write.csv2(sample_2, csvfiles[2], row.names = TRUE)
loadDataFromFileList(
  csvfiles,
  input_type = "csv2",
  read.args = list(row.names = 1)
)

# tab-separated values with header row and decimals as commas
.tsvfiles <- tempfile(c("sample1", "sample2"), fileext = ".tsv")
utils::write.table(sample_1, tsvfiles[1], sep = "\t", dec = ",")
utils::write.table(sample_2, tsvfiles[2], sep = "\t", dec = ",")
loadDataFromFileList(
  tsvfiles,
  input_type = "tsv",
  header = TRUE,
  read.args = list(dec = ",")
)
# space-separated values with header row and NAs encoded as as "No Value"

txtfiles <- tempfile(c("sample1", "sample2"), fileext = ".txt")
utils::write.table(sample_1, txtfiles[1], na = "No Value")
utils::write.table(sample_2, txtfiles[2], na = "No Value")
loadDataFromFileList(
  txtfiles,
  input_type = "table",
  read.args = list(  
    header = TRUE,
    na.strings = "No Value"
  )
)

# custom value separator and row names in first column
utils::write.table(sample_1, txtfiles[1],
  sep = "@", row.names = TRUE, col.names = FALSE)
utils::write.table(sample_2, txtfiles[2],
  sep = "@", row.names = TRUE, col.names = FALSE)
loadDataFromFileList(
  txtfiles,
  input_type = "table",
  sep = "@",
  read.args = list(  
    row.names = 1,
    col.names = c("rownames",
                  "CloneSeq", "CloneFrequency",
                  "CloneCount", "SampleID"
    )
  )
)

# same as previous example
# (value of sep in read.args overrides value in sep argument)
loadDataFromFileList(
  txtfiles,
  input_type = "table",
  sep = "\t",
  read.args = list(  
    sep = "@",
    row.names = 1,
    col.names = c("rownames",
                  "CloneSeq", "CloneFrequency",
                  "CloneCount", "SampleID"
    )
  )
)
extractLayout

Get Coordinate Layout From Graph Plot

Description

Given a `ggraph` plot, extract the coordinate layout of the graph nodes as a two-column matrix.

Usage

```r
extractLayout(plot)
```

Arguments

- `plot` An object of class `ggraph`.

Details

Equivalent to `as.matrix(plot$data[c("x", "y")])`.

Value

A matrix with two columns and one row per network node. Each row contains the Cartesian coordinates of the corresponding node.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package

Examples

```r
set.seed(42)
toy_data <- simulateToyData()
net <- buildRepSeqNetwork(toy_data, "CloneSeq", print_plots = TRUE)

my_layout <- extractLayout(net$plots[[1]])

# same as `graph_layout` element in the plot list
all.equal(my_layout, net$plots$graph_layout, check.attributes = FALSE)
```
filterInputData  Filter Data Rows and Subset Data Columns

Description
Given a data frame with a column containing receptor sequences, filter data rows by sequence length and sequence content. Keep all data columns or choose which columns to keep.

Usage
filterInputData(
  data,
  seq_col,
  min_seq_length = NULL,
  drop_matches = NULL,
  subset_cols = NULL,
  count_col = deprecated(),
  verbose = FALSE
)

Arguments

  data          A data frame.
  seq_col       Specifies the column(s) of data containing the receptor sequences. Accepts a character or numeric vector of length 1 or 2, containing either column names or column indices. Each column specified will be coerced to a character vector. Data rows containing a value of NA in any of the specified columns will be dropped.
  min_seq_length Observations whose receptor sequences have fewer than min_seq_length characters are dropped.
  drop_matches  Accepts a character string containing a regular expression (see \texttt{regex}). Checks values in the receptor sequence column for a pattern match using \texttt{grep()}. Rows in which a match is found are dropped.
  subset_cols   Specifies which columns of the AIRR-Seq data are included in the output. Accepts a character vector of column names or a numeric vector of column indices. The default \texttt{NULL} includes all columns. The receptor sequence column is always included regardless of this argument’s value.
  count_col     [\textbf{Deprecated}] Does nothing.
  verbose       Logical. If \texttt{TRUE}, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to \texttt{stderr()}.

Value
A data frame.
findAssociatedClones

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package

Examples

```r
set.seed(42)
raw_data <- simulateToyData()

# Remove sequences shorter than 13 characters,
# as well as sequences containing the subsequence "GGGG".
# Keep variables for clone sequence, clone frequency and sample ID
filterInputData(
  raw_data,
  seq_col = "CloneSeq",
  min_seq_length = 13,
  drop_matches = "GGGG",
  subset_cols =
    c("CloneSeq", "CloneFrequency", "SampleID"),
  verbose = TRUE
)
```

findAssociatedClones

Identify TCR/BCR Clones in a Neighborhood Around Each Associated Sequence

Description

Part of the workflow Searching for Associated TCR/BCR Clusters. Intended for use following `findAssociatedSeqs()` and prior to `buildAssociatedClusterNetwork()`.

Given multiple samples of bulk Adaptive Immune Receptor Repertoire Sequencing (AIRR-Seq) data and a vector of associated sequences, identifies for each associated sequence a global "neigh- borhood" comprised of clones with TCR/BCR sequences similar to the associated sequence.

Usage

```r
findAssociatedClones(

  ## Input ##
  file_list, input_type,
  data_symbols = NULL,
  `...`
)
```
findAssociatedClones

header, sep, read.args,
sample_ids =
paste0("Sample", 1:length(file_list)),
sample_ids = NULL,
subject_ids,
seq_col,
assoc_seqs,

## Neighborhood Criteria ##
nbd_radius = 1,
dist_type = "hamming",
min_seq_length = 6,
drop_matches = NULL,

## Output ##
subset_cols = NULL,
output_dir,
output_type = "rds",
verbose = FALSE

}  

Arguments

file_list A character vector of file paths, or a list containing connections and file paths. Each element corresponds to a single file containing the data for a single sample. Passed to loadDataFromFileList().

input_type A character string specifying the file format of the sample data files. Options are "table", "txt", "tsv", "csv", "rds" and "rda". Passed to loadDataFromFileList().

data_symbols Used when input_type = "rda". Specifies the name of each sample’s data frame within its respective Rdata file. Passed to loadDataFromFileList().

header For values of input_type other than "rds" and "rda", this argument can be used to specify a non-default value of the header argument to read.table(), read.csv(), etc.

sep For values of input_type other than "rds" and "rda", this argument can be used to specify a non-default value of the sep argument to read.table(), read.csv(), etc.

read.args For values of input_type other than "rds" and "rda", this argument can be used to specify non-default values of optional arguments to read.table(), read.csv(), etc. Accepts a named list of argument values. Values of header and sep in this list take precedence over values specified via the header and sep arguments.

sample_ids A character or numeric vector of sample IDs, whose length matches that of file_list. Each entry is assigned as the sample ID to the corresponding entry of file_list.

subject_ids An optional character or numeric vector of subject IDs, whose length matches that of file_list. Used to assign a subject ID to each sample.
group_ids A character or numeric vector of group IDs whose length matches that of file_list. Used to assign each sample to a group. The two groups represent the levels of the binary variable of interest.

seq_col Specifies the column of each sample’s data frame containing the TCR/BCR sequences. Accepts a character string containing the column name or a numeric scalar containing the column index.

assoc_seqs A character vector containing the TCR/BCR sequences associated with the binary variable of interest.

nbd_radius The maximum distance (based on dist_type) between an associated sequence and other TCR/BCR sequences belonging to its neighborhood. Lower values require sequences to be more similar to an associated sequence in order to belong to its neighborhood.

dist_type Specifies the function used to quantify the similarity between sequences. The similarity between two sequences determines their pairwise distance, with greater similarity corresponding to shorter distance. Valid options are “hamming” (the default), which uses hamDistBounded(), and “levenshtein”, which uses levDistBounded().

min_seq_length Clones with TCR/BCR sequences below this length will be removed. Passed to filterInputData() when loading each sample.

drop_matches Passed to filterInputData(). Accepts a character string containing a regular expression (see regex). Checks TCR/BCR sequences for a pattern match using grep(). Those returning a match are dropped. By default, sequences containing any of the characters *, | or _ are dropped.

subset_cols Controls which columns of the AIRR-Seq data from each sample are included in the output. Accepts a character vector of column names or a numeric vector of column indices. The default NULL includes all columns. Passed to filterInputData().

output_dir A file path to a directory for saving the output. A valid output directory is required, since no output is returned in R. The specified directory will be created if it does not already exist.

output_type A character string specifying the file format to use for saving the output. Valid options are “rda”, “csv”, “csv2”, “tsv” and “table”. For “rda”, data frames are named data in the R environment. For the remaining options, write.table() is called with row.names = TRUE.

verbose Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().

Details

For each associated sequence, its neighborhood is defined to include all clones with TCR/BCR sequences that are sufficiently similar to the associated sequence. The arguments dist_type and nbd_radius control how the similarity is measured and the degree of similarity required for neighborhood membership.

For each associated sequence, a data frame is saved to an individual file. The data frame contains one row for each clone in the associated sequence’s neighborhood (from all samples). It includes variables for sample ID, group ID and (if provided) subject ID, as well as variables from the AIRR-Seq data.
The files saved by this function are intended for use with `buildAssociatedClusterNetwork()`. See the Searching for Associated TCR/BCR Clusters article on the package website for more details.

**Value**

Returns TRUE, invisibly. The function is called for its side effects.

**Author(s)**

Brian Neal (<Brian.Neal@ucsf.edu>)

**References**


Webpage for the NAIR package
Searching for Associated TCR/BCR Clusters article on package website

**See Also**

`findAssociatedSeqs()` `buildAssociatedClusterNetwork()`

**Examples**

```r
set.seed(42)

## Simulate 30 samples from two groups (treatment/control) ##
n_control <- n_treatment <- 15
n_samples <- n_control + n_treatment
sample_size <- 30 # (seqs per sample)
base_seqs <- # first five are associated with treatment
c("CASSGAYEQYF", "CSVDLGKGNNEQFF", "CASSIEGQLSTDTQYF",
  "CASSIEGQLSTDTQYF", "CASSIEGQLSTDTQYF",
  "RASSLAGNTEAFF", "CASSHRGTDTQYF", "CASDAGVFQPQHF")
# Relative generation probabilities by control/treatment group
pgen_c <- matrix(rep(c(rep(1, 5), rep(30, 3)), times = n_control),
  nrow = n_control, byrow = TRUE)
pgen_t <- matrix(rep(c(1, 1, rep(1/3, 3), rep(2, 3)), times = n_treatment),
  nrow = n_treatment, byrow = TRUE)
pgen <- rbind(pgen_c, pgen_t)
simulateToyData(
  samples = n_samples,
  sample_size = sample_size,
  prefix_length = 1,
  prefix_chars = c("", ""),
  prefix_probs = cbind(rep(1, n_samples), rep(0, n_samples)),
  affixes = base_seqs,
  affix_probs = pgen,
  num_edits = 0,
  output_dir = tempdir(),
  no_return = TRUE
)
findAssociatedSeqs

## Step 1: Find Associated Sequences ##
sample_files <-
  file.path(tempdir(),
     paste0("Sample", 1:n_samples, ".rds")
)

group_labels <- c(rep("reference", n_control),
   rep("comparison", n_treatment))

associated_seqs <-
  findAssociatedSeqs(
    file_list = sample_files,
    input_type = "rds",
    group_ids = group_labels,
    seq_col = "CloneSeq",
    min_seq_length = NULL,
    drop_matches = NULL,
    min_sample_membership = 0,
    pval_cutoff = 0.1
  )

head(associated_seqs[, 1:5])

## Step 2: Find Associated Clones ##
dir_step2 <- tempfile()

findAssociatedClones(
  file_list = sample_files,
  input_type = "rds",
  group_ids = group_labels,
  seq_col = "CloneSeq",
  assoc_seqs = associated_seqs$ReceptorSeq,
  min_seq_length = NULL,
  drop_matches = NULL,
  output_dir = dir_step2
)

## Step 3: Global Network of Associated Clusters ##
associated_clusters <-
  buildAssociatedClusterNetwork(
    file_list = list.files(dir_step2,
       full.names = TRUE
    ),
    seq_col = "CloneSeq",
    size_nodes_by = 1.5,
    print_plots = TRUE
  )

---

findAssociatedSeqs | Identify TCR/BCR Sequences Associated With a Binary Variable
**Description**

Part of the workflow **Searching for Associated TCR/BCR Clusters**.

Given multiple samples of bulk Adaptive Immune Receptor Repertoire Sequencing (AIRR-Seq) data and a binary variable of interest such as a disease condition, treatment or clinical outcome, identify receptor sequences that exhibit a statistically significant difference in frequency between the two levels of the binary variable.

`findAssociatedSeqs()` is designed for use when each sample is stored in a separate file. `findAssociatedSeqs2()` is designed for use with a single data frame containing all samples.

**Usage**

```r
defindAssociatedSeqs(
  ## Input ##
  file_list,
  input_type,
  data_symbols = NULL,
  header, sep, read.args,
  sample_ids = deprecated(),
  subject_ids = NULL,
  group_ids,
  groups = deprecated(),
  seq_col,
  freq_col = NULL,

  ## Search Criteria ##
  min_seq_length = 7,
  drop_matches = "[*|_]",
  min_sample_membership = 5,
  pval_cutoff = 0.05,

  ## Output ##
  outfile = NULL,
  verbose = FALSE
)
```

```r
defindAssociatedSeqs2(
  ## Input ##
  data,
  seq_col,
  sample_col,
  subject_col = sample_col,
  group_col,
  groups = deprecated(),
  freq_col = NULL,

  ## Search Criteria ##
  min_seq_length = 7,
```
```r
findAssociatedSeqs

Arguments

file_list A character vector of file paths, or a list containing connections and file paths. Each element corresponds to a single file containing the data for a single sample. Passed to `loadDataFromFileList()`.

input_type A character string specifying the file format of the sample data files. Options are "table", "txt", "tsv", "csv", "rds" and "rda". Passed to `loadDataFromFileList()`.

data_symbols Used when `input_type = "rda"`. Specifies the name of each sample’s data frame within its respective Rdata file. Passed to `loadDataFromFileList()`.

header For values of `input_type` other than "rds" and "rda", this argument can be used to specify a non-default value of the header argument to `read.table()`, `read.csv()`, etc.

sep For values of `input_type` other than "rds" and "rda", this argument can be used to specify a non-default value of the sep argument to `read.table()`, `read.csv()`, etc.

read.args For values of `input_type` other than "rds" and "rda", this argument can be used to specify non-default values of optional arguments to `read.table()`, `read.csv()`, etc. Accepts a named list of argument values. Values of header and sep in this list take precedence over values specified via the header and sep arguments.

sample_ids [Deprecated] Does nothing.

subject_ids A character or numeric vector of subject IDs, whose length matches that of `file_list`. Only relevant when the binary variable of interest is subject-specific and multiple samples belong to the same subject.

group_ids A character or numeric vector of group IDs containing exactly two unique values and with length matching that of `file_list`. The two groups correspond to the two values of the binary variable of interest.

groups [Deprecated] Does nothing.

seq_col Specifies the column of each sample’s data frame containing the TCR/BCR sequences. Accepts a character string containing the column name or a numeric scalar containing the column index.

freq_col Optional. Specifies the column of each sample’s data frame containing the clone frequency (i.e., clone count divided by the sum of the clone counts across all clones in the sample). Accepts a character string containing the column name or a numeric scalar containing the column index. If this argument is specified,
the maximum clone frequency (across all samples) for each associated sequence will be included in the content of the label variable of the returned data frame.

**min_seq_length** Controls the minimum TCR/BCR sequence length considered when searching for associated sequences. Passed to `filterInputData()`.

**drop_matches** Passed to `filterInputData()`. Accepts a character string containing a regular expression (see `regex`). Checks TCR/BCR sequences for a pattern match using `grep()`. Those returning a match are excluded from consideration as associated sequences. It is recommended to filter out sequences containing special characters that are invalid for use in file names. By default, sequences containing any of the characters *, | or _ are dropped.

**min_sample_membership** Controls the minimum number of samples in which a TCR/BCR sequence must be present in order to be considered when searching for associated sequences. Setting this value to `NULL` bypasses the check.

**pval_cutoff** Controls the P-value cutoff below which an association is detected by Fisher’s exact test (see details).

**outfile** A file path for saving the output (using `write.csv()`).

**verbose** Logical. If `TRUE`, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to `stderr()`.

**data** A data frame containing the combined AIRR-seq data for all samples.

**sample_col** The column of data containing the sample IDs. Accepts a character string containing the column name or a numeric scalar containing the column index.

**subject_col** Optional. The column of data containing the subject IDs. Accepts a character string containing the column name or a numeric scalar containing the column index. Only relevant when the binary variable of interest is subject-specific and multiple samples belong to the same subject.

**group_col** The column of data containing the group IDs. Accepts a character string containing the column name or a numeric scalar containing the column index. The groups correspond to the two values of the binary variable of interest. Thus there should be exactly two unique values in this column.

**Details**

The TCR/BCR sequences from all samples are first filtered according to minimum sequence length and sequence content based on the specified values in `min_seq_length` and `drop_matches`, respectively. The sequences are further filtered based on sample membership, removing sequences appearing in fewer than `min_sample_membership` samples.

For each remaining TCR/BCR sequence, a P-value is computed for Fisher’s exact test of independence between the binary variable of interest and the presence of the sequence within a repertoire. The samples/subjects are divided into two groups based on the levels of the binary variable. If subject IDs are provided, then the test is based on the number of subjects in each group for whom the sequence appears in one of their samples. Without subject IDs, the test is based on the number of samples possessing the sequence in each group.

Fisher’s exact test is performed using `fisher.test()`. TCR/BCR sequences with a P-value below `pval_cutoff` are sorted by P-value and returned along with some additional information.
The returned output is intended for use with the `findAssociatedClones()` function. See the Searching for Associated TCR/BCR Clusters article on the package website.

**Value**

A data frame containing the TCR/BCR sequences found to be associated with the binary variable using Fisher's exact test (see details). Each row corresponds to a unique TCR/BCR sequence and includes the following variables:

- **ReceptorSeq**: The unique receptor sequence.
- **fisher_pvalue**: The P-value on Fisher's exact test for independence between the receptor sequence and the binary variable of interest.
- **shared_by_n_samples**: The number of samples in which the sequence was observed.
- **samples_g0**: Of the samples in which the sequence was observed, the number of samples belonging to the first group.
- **samples_g1**: Of the samples in which the sequence was observed, the number of samples belonging to the second group.
- **shared_by_n_subjects**: The number of subjects in which the sequence was observed (only present if subject IDs are specified).
- **subjects_g0**: Of the subjects in which the sequence was observed, the number of subjects belonging to the first group (only present if subject IDs are specified).
- **subjects_g1**: Of the subjects in which the sequence was observed, the number of subjects belonging to the second group (only present if subject IDs are specified).
- **max_freq**: The maximum clone frequency across all samples. Only present if `freq_col` is non-null.
- **label**: A character string summarizing the above information. Also includes the maximum in-sample clone frequency across all samples, if available.

**Author(s)**

Brian Neal (<Brian.Neal@ucsf.edu>)

**References**


Webpage for the NAIR package

Searching for Associated TCR/BCR Clusters article on package website

**See Also**

`findAssociatedClones()` `buildAssociatedClusterNetwork()`
Examples

set.seed(42)

## Simulate 30 samples from two groups (treatment/control) ##
n_control <- n_treatment <- 15
n_samples <- n_control + n_treatment
sample_size <- 30 # (seqs per sample)
base_seqs <- # first five are associated with treatment
c("CASSGAYEQYF", "CSVDLGKGNNEQFF", "CASSIEQQLSTDQTQYF",
"CASSEIEQLSTDQTQYF", "CASSEEGQLSTDQTQYF",
"RASSLAGNTEAFF", "CASSHRGTDTQYF", "CASSAGVFQPQHF")
# Relative generation probabilities by control/treatment group
pgen_c <- matrix(rep(c(rep(1, 5), rep(30, 3)), times = n_control),
nrow = n_control, byrow = TRUE)
pген_t <- matrix(rep(c(1, 1, rep(1/3, 3), rep(2, 3)), times = n_treatment),
nrow = n_treatment, byrow = TRUE)
pген <- rbind(pgen_c, pgen_t)
simulateToyData(
  samples = n_samples,
  sample_size = sample_size,
  prefix_length = 1,
  prefix_chars = c("", ""),
  prefix_probs = cbind(rep(1, n_samples), rep(0, n_samples)),
  affixes = base_seqs,
  affix_probs = pgen,
  num_edits = 0,
  output_dir = tempdir(),
  no_return = TRUE
)

## Step 1: Find Associated Sequences ##
sample_files <-
  file.path(tempdir(),
            paste0("Sample", 1:n_samples, ".rds")
  )
group_labels <- c(rep("reference", n_control),
                   rep("comparison", n_treatment))
associated_seqs <-
  findAssociatedSeqs(
    file_list = sample_files,
    input_type = "rds",
    group_ids = group_labels,
    seq_col = "CloneSeq",
    min_seq_length = NULL,
    drop_matches = NULL,
    min_sample_membership = 0,
    pval_cutoff = 0.1
  )
head(associated_seqs[, 1:5])

## Step 2: Find Associated Clones ##
dir_step2 <- tempfile()
findPublicClusters

Find Public Clusters Among RepSeq Samples

Description

Part of the workflow Searching for Public TCR/BCR Clusters.

Given multiple samples of bulk Adaptive Immune Receptor Repertoire Sequencing (AIRR-Seq) data, construct the repertoire network for each sample. Within each sample’s network, perform cluster analysis and filter the clusters based on node count and aggregate clone count.

Usage

findPublicClusters()

## Input ##

file_list, input_type, 
data_symbols = NULL, 
header, sep, read.args, 
sample_ids = 
  paste0("Sample", 1:length(file_list)), 
seq_col, 
count_col = NULL,
findPublicClusters

## Search Criteria ##
min_seq_length = 3,
drop_matches = "[\*\|\_]",
top_n_clusters = 20,
min_node_count = 10,
min_clone_count = 100,

## Optional Visualization ##
plots = FALSE,
print_plots = FALSE,
plot_title = "auto",
color_nodes_by = "cluster_id",

## Output ##
output_dir,
output_type = "rds",

## Optional Output ##
output_dir_unfiltered = NULL,
output_type_unfiltered = "rds",
verbose = FALSE,
...

Arguments

file_list A character vector of file paths, or a list containing connections and file paths. Each element corresponds to a single file containing the data for a single sample. Passed to loadDataFromFileList().

input_type A character string specifying the file format of the sample data files. Options are "table", "txt", "tsv", "csv", "rds" and "rda". Passed to loadDataFromFileList().

data_symbols Used when input_type = "rda". Specifies the name of each sample’s data frame within its respective Rdata file. Passed to loadDataFromFileList().

header For values of input_type other than "rds" and "rda", this argument can be used to specify a non-default value of the header argument to read.table(), read.csv(), etc.

sep For values of input_type other than "rds" and "rda", this argument can be used to specify a non-default value of the sep argument to read.table(), read.csv(), etc.

read.args For values of input_type other than "rds" and "rda", this argument can be used to specify non-default values of optional arguments to read.table(), read.csv(), etc. Accepts a named list of argument values. Values of header and sep in this list take precedence over values specified via the header and sep arguments.
sample_ids A character or numeric vector of sample IDs, whose length matches that of file_list. The values should be valid for use as filenames and should avoid using the forward slash or backslash characters (/ or \).

seq_col Specifies the column of each sample’s data frame containing the TCR/BCR sequences. Accepts a character string containing the column name or a numeric scalar containing the column index.

count_col Specifies the column of each sample’s data frame containing the clone count (measure of clonal abundance). Accepts a character string containing the column name or a numeric scalar containing the column index. If NULL, the clusters in each sample’s network will be selected solely based upon node count.

min_seq_length Passed to buildRepSeqNetwork() when constructing the network for each sample.

drop_matches Passed to buildRepSeqNetwork() when constructing the network for each sample. Accepts a character string containing a regular expression (see regex). Checks TCR/BCR sequences for a pattern match using grep(). Those returning a match are dropped. By default, sequences containing any of the characters *, | or _ are dropped.

top_n_clusters The number of clusters from each sample to be automatically be included among the filtered clusters, based on greatest node count.

min_node_count Clusters with at least this many nodes will be included among the filtered clusters.

min_clone_count Clusters with an aggregate clone count of at least this value will be included among the filtered clusters. A value of NULL ignores this criterion and does not select additional clusters based on clone count.

plots Passed to buildRepSeqNetwork() when constructing the network for each sample.

print_plots Passed to buildRepSeqNetwork() when constructing the network for each sample.

plot_title Passed to buildRepSeqNetwork() when constructing the network for each sample.

color_nodes_by Passed to buildRepSeqNetwork() when constructing the network for each sample.

output_dir The file path of the directory for saving the output. The directory will be created if it does not already exist.

output_type A character string specifying the file format to use for saving the output. Valid options include “csv”, “rds” and “rda”.

output_dir_unfiltered An optional directory for saving the unfiltered network data for each sample. By default, only the filtered results are saved.

output_type_unfiltered A character string specifying the file format to use for saving the unfiltered network data for each sample. Only applicable if output_dir_unfiltered is non-null. Passed to buildRepSeqNetwork() when constructing the network for each sample.
findPublicClusters

verbose Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().

... Other arguments to buildRepSeqNetwork when constructing the network for each sample, not including node_stats, stats_to_include, cluster_stats, cluster_id_name or output_name (see details).

Details

Each sample's network is constructed using an individual call to buildNet() with node_stats = TRUE, stats_to_include = "all", cluster_stats = TRUE and cluster_id_name = "ClusterIDInSample". The node-level properties are renamed to reflect their correspondence to the sample-level network. Specifically, the properties are named:

- SampleLevelNetworkDegree
- SampleLevelTransitivity
- SampleLevelCloseness
- SampleLevelCentralityByCloseness
- SampleLevelCentralityByEigen
- SampleLevelEigenCentrality
- SampleLevelBetweenness
- SampleLevelCentralityByBetweenness
- SampleLevelAuthorityScore
- SampleLevelCoreness
- SampleLevelPageRank

A variable SampleID is added to both the node-level and cluster-level meta data for each sample. After the clusters in each sample are filtered, the node-level and cluster-level metadata are saved in the respective subdirectories node_meta_data and cluster_meta_data of the output directory specified by output_dir.

The unfiltered network results for each sample can also be saved by supplying a directory to output_dir_unfiltered, if these results are desired for downstream analysis. Each sample's unfiltered network results will then be saved to its own subdirectory created within this directory.

The files containing the node-level metadata for the filtered clusters can be supplied to buildPublicClusterNetwork() in order to construct a global network of public clusters. If the full global network is too large to practically construct, the files containing the cluster-level meta data for the filtered clusters can be supplied to buildPublicClusterNetworkByRepresentative() to build a global network using only a single representative sequence from each cluster. This allows prominent public clusters to still be identified.

See the Searching for Public TCR/BCR Clusters article on the package website.

Value

Returns TRUE, invisibly.
findPublicClusters

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package

Searching for Public TCR/BCR Clusters vignette

See Also

buildPublicClusterNetwork()
buildPublicClusterNetworkByRepresentative()

Examples

```r
set.seed(42)

## Simulate 30 samples with a mix of public/private sequences ##
samples <- 30
sample_size <- 30  # (seqs per sample)
base_segs <- c(
  "CASSIEGQLSTDQYF", "CASSEEQLSTDQYF", "CASSSVETQYF",
  "CASSPEGQLSTDQYF", "RASSLAGNTEAFF", "CASSHRGTSTDQYF", "CASDAGVFQPQHF",
  "CASSLTSQYNEQFF", "CASSETQYNEQFF", "CASSLTGQNEQFF", "CASSYLTGQNEQFF",
  "CASSLGTQNEQFF", "CASSLNGQNEQFF", "CASSFPWDQYGQYF", "CASTLARQGQGLF",
  "CASTLARQGQGLF", "CSVELLPTPQTSYNEQFF", "CSVELLPTGQTSYNEQFF",
  "CSVELLPTGQTSYNEQFF", "CSVELLPTAQTSQETQYF", "CASLAGRPQTSQETQYF",
  "CASRRAGRTAQETQYF", "CASRRALGRTAQETQYF",
  "CASSQGQGQGPQQH", "CASSLGGQGQGPQH", "CASSNQGQGQGPQH", "CASSYGGQGQGPQH",
  "CASSYGGQGQGPQH", "CASSYGGQGQGPQH", "CASSYGGQGQGPQH",
  "CASSQGQGQGPQH", "CASSSAGQGQGPQH", "CASSAGYEQYF", "CSVDLQGKNEQFF")

# Relative generation probabilities
pgen <- cbind(
  stats::toeplitz(0.6*(0:(sample_size - 1))),
  matrix(1, nrow = samples, ncol = length(base_segs) - samples)
)

simulateToyData(
  samples = samples,
  sample_size = sample_size,
  prefix_length = 1,
  prefix_chars = c("", ""),
  prefix_probs = cbind(rep(1, samples), rep(0, samples)),
  affixes = base_segs,
  affix_probs = pgen,
  num_edits = 0,
  output_dir = tempdir(),
  no_return = TRUE
)
```

```r
```
sample_files <-
  file.path(tempdir(),
    paste0("Sample", 1:samples, ".rds")
  )
findPublicClusters(
  file_list = sample_files,
  input_type = "rds",
  seq_col = "CloneSeq",
  count_col = "CloneCount",
  min_seq_length = NULL,
  drop_matches = NULL,
  top_n_clusters = 3,
  min_node_count = 5,
  min_clone_count = 15000,
  output_dir = tempdir()
)

---

generateAdjacencyMatrix

*Compute Graph Adjacency Matrix for Immune Repertoire Network*

**Description**

Given a list of receptor sequences, computes the adjacency matrix for the network graph based on sequence similarity.

`sparseAdjacencyMatFromSeqs()` is a deprecated equivalent of `generateAdjacencyMatrix()`.

**Usage**

```r
generateAdjacencyMatrix(
  seqs,
  dist_type = "hamming",
  dist_cutoff = 1,
  drop_isolated_nodes = TRUE,
  method = "default",
  verbose = FALSE
)
```

# Deprecated equivalent:

```r
sparseAdjacencyMatFromSeqs(
  seqs,
  dist_type = "hamming",
  dist_cutoff = 1,
  drop_isolated_nodes = TRUE,
```
Arguments

seqs
A character vector containing the receptor sequences.

dist_type
Specifies the function used to quantify the similarity between sequences. The similarity between two sequences determines the pairwise distance between their respective nodes in the network graph, with greater similarity corresponding to shorter distance. Valid options are "hamming" (the default), which uses hamDistBounded, and "levenshtein", which uses levDistBounded.

dist_cutoff
A nonnegative scalar. Specifies the maximum pairwise distance (based on dist_type) for an edge connection to exist between two nodes. Pairs of nodes whose distance is less than or equal to this value will be joined by an edge connection in the network graph. Controls the stringency of the network construction and affects the number and density of edges in the network. A lower cutoff value requires greater similarity between sequences in order for their respective nodes to be joined by an edge connection. A value of 0 requires two sequences to be identical in order for their nodes to be joined by an edge.

drop_isolated_nodes
Logical. When TRUE, removes each node that is not joined by an edge connection to any other node in the network graph.

method
A character string specifying the algorithm to use. Choices are "default" and "pattern". "pattern" is only valid when dist_cutoff < 3, but tends to be faster than "default" for sparsely connected networks, at the cost of greater memory usage (can cause crashes for large or densely-connected networks, particularly for dist_cutoff = 2). The default algorithm tends to be faster for densely-connected networks or long sequences.

verbose
Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr.

max_dist
Equivalent to dist_cutoff.

Details

The adjacency matrix of a graph with n nodes is the symmetric n × n matrix for which entry (i, j) is equal to 1 if nodes i and j are connected by an edge in the network graph and 0 otherwise.

To construct the graph of the immune repertoire network, each receptor sequence is modeled as a node. The similarity between receptor sequences, as measured using either the Hamming or Levenshtein distance, determines the distance between nodes in the network graph. The more similar two sequences are, the shorter the distance between their respective nodes. Two nodes in the graph are joined by an edge if the distance between them is sufficiently small, i.e., if their receptor sequences are sufficiently similar.
generateAdjacencyMatrix

Value

A sparse matrix of class dgCMatrix (see dgCMatrix-class).

If drop_isolated_nodes = TRUE, the row and column names of the matrix indicate which receptor sequences in the seqs vector correspond to each row and column of the matrix. The row and column names can be accessed using dimnames. This returns a list containing two character vectors, one for the row names and one for the column names. The name of the i\text{th} matrix row is the index of the seqs vector corresponding to the i\text{th} row and i\text{th} column of the matrix. The name of the j\text{th} matrix column is the receptor sequence corresponding to the j\text{th} row and j\text{th} column of the matrix.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package

Examples

generateAdjacencyMatrix(
  c("fee", "fie", "foe", "fum", "foo")
)

# No edge connections exist based on a Hamming distance of 1
# (returns a 0x0 sparse matrix)
generateAdjacencyMatrix(
  c("foo", "foobar", "fubar", "bar")
)

# Same as the above example, but keeping all nodes
# (returns a 4x4 sparse matrix)
generateAdjacencyMatrix(
  c("foo", "foobar", "fubar", "bar"),
  drop_isolated_nodes = FALSE
)

# Relaxing the edge criteria using a Hamming distance of 2
# (still results in no edge connections)
generateAdjacencyMatrix(
  c("foo", "foobar", "fubar", "bar"),
  dist_cutoff = 2
)

# Using a Levenshtein distance of 2, however,
# does result in edge connections
generateAdjacencyMatrix(
  c("foo", "foobar", "fubar", "bar"),
  dist_type = "levenshtein",
)
generateNetworkGraph


dist_cutoff = 2
)

# Using a Hamming distance of 3
# also results in (different) edge connections
generateAdjacencyMatrix(
  c("foo", "foobar", "fubar", "bar"),
  dist_cutoff = 3
)

generateNetworkGraph  Generate the igraph for a Network Adjacency Matrix

Description

Given the adjacency matrix of an undirected graph, returns the corresponding igraph containing the list of nodes and edges.
generateNetworkFromAdjacencyMat() is a deprecated equivalent of generateNetworkGraph().

Usage

generateNetworkGraph(
  adjacency_matrix
)

# Deprecated equivalent:
generateNetworkFromAdjacencyMat(
  adjacency_matrix
)

Arguments

adjacency_matrix
  A symmetric matrix. Passed to graph_from_adjacency_matrix().

Value

An object of class igraph, containing the list of nodes and edges corresponding to adjacency_matrix.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References

Webpage for the NAIR package
Examples

```r
set.seed(42)
toy_data <- simulateToyData(sample_size = 10)

adj_mat <-
genenerateAdjacencyMatrix(
  toy_data$CloneSeq
)

igraph <-
genenerateNetworkGraph(
  adj_mat
)
```

Description

Given Adaptive Immune Receptor Repertoire Sequencing (AIRR-Seq) data, builds the network graph for the immune repertoire based on sequence similarity.

Usage

```r
generateNetworkObjects(
  data,  
  seq_col, 
  dist_type = "hamming",  
  dist_cutoff = 1, 
  drop_isolated_nodes = TRUE, 
  verbose = FALSE 
)
```

Arguments

- **data**: A data frame containing the AIRR-Seq data, with variables indexed by column and observations (e.g., clones or cells) indexed by row.
- **seq_col**: Specifies the column(s) of data containing the receptor sequences to be used as the basis of similarity between rows. Accepts a character string containing the column name or a numeric scalar containing the column index. Also accepts a vector of length 2 specifying distinct sequence columns (e.g., alpha chain and beta chain), in which case similarity between rows depends on similarity in both sequence columns (see details).
- **dist_type**: Specifies the function used to measure the similarity between sequences. The similarity between two sequences determines the pairwise distance between their respective nodes in the network graph. Valid options are "hamming" (the default), which uses `hamDistBounded()`, and "levenshtein", which uses `levDistBounded()`. 
- **dist_cutoff**: A cutoff for the similarity distance. Includes all nodes that are within this distance of each other as neighbors.
- **drop_isolated_nodes**: A logical value. If `TRUE` (the default), all isolated nodes (nodes with no connections) are removed from the graph.
- **verbose**: A logical value. If `TRUE`, additional information is printed on the progress of the function.
dist_cutoff  A nonnegative scalar. Specifies the maximum pairwise distance (based on dist_type) for an edge connection to exist between two nodes. Pairs of nodes whose distance is less than or equal to this value will be joined by an edge connection in the network graph. Controls the stringency of the network construction and affects the number and density of edges in the network. A lower cutoff value requires greater similarity between sequences in order for their respective nodes to be joined by an edge connection. A value of 0 requires two sequences to be identical in order for their nodes to be joined by an edge.

drop_isolated_nodes  A logical scalar. When TRUE, removes each node that is not joined by an edge connection to any other node in the network graph.

verbose  Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().

Details

To construct the immune repertoire network, each TCR/BCR clone (bulk data) or cell (single-cell data) is modeled as a node in the network graph, corresponding to a single row of the AIRR-Seq data. For each node, the corresponding receptor sequence is considered. Both nucleotide and amino acid sequences are supported for this purpose. The receptor sequence is used as the basis of similarity and distance between nodes in the network.

Similarity between sequences is measured using either the Hamming distance or Levenshtein (edit) distance. The similarity determines the pairwise distance between nodes in the network graph. The more similar two sequences are, the shorter the distance between their respective nodes. Two nodes are joined by an edge if their receptor sequences are sufficiently similar, i.e., if the distance between the nodes is sufficiently small.

For single-cell data, edge connections between nodes can be based on similarity in both the alpha chain and beta chain sequences. This is done by providing a vector of length 2 to seq_cols specifying the two sequence columns in data. The distance between two nodes is then the greater of the two distances between sequences in corresponding chains. Two nodes will be joined by an edge if their alpha chain sequences are sufficiently similar and their beta chain sequences are sufficiently similar.

See the buildRepSeqNetwork package vignette for more details. The vignette can be accessed offline using vignette("buildRepSeqNetwork")..

Value

If the constructed network contains no nodes, the function will return NULL, invisibly, with a warning. Otherwise, the function invisibly returns a list containing the following items:

igraph  An object of class igraph containing the list of nodes and edges for the network graph.

adjacency_matrix  The network graph adjacency matrix, stored as a sparse matrix of class dgCMatrix from the Matrix package. See dgCMatrix-class.
node_data A data frame containing metadata for the network nodes, where each row corresponds to a node in the network graph. This data frame contains all variables from data (unless otherwise specified via subset.cols) in addition to the computed node-level network properties if node_stats = TRUE. Each row's name is the name of the corresponding row from data.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Examples

```r
set.seed(42)
toy_data <- simulateToyData()

net <-
generateNetworkObjects(
  toy_data,
  "CloneSeq"
)
```

```r
getClusterStats(data, adjacency_matrix, seq_col = NULL, count_col = NULL, cluster_id_col = "cluster_id", degree_col = NULL, cluster_fun = deprecated(), verbose = FALSE)
```

Description

Given the node-level metadata and adjacency matrix for a network graph that has been partitioned into clusters, computes network properties for the clusters and returns them in a data frame.

addClusterStats() is preferred to getClusterStats() in most situations.


getClusterStats

Arguments

data A data frame containing the node-level metadata for the network, with each row corresponding to a network node.

adjacency_matrix The adjacency matrix for the network.

seq_col Specifies the column(s) of data containing the receptor sequences upon whose similarity the network is based. Accepts a character or numeric vector of length 1 or 2, containing either column names or column indices. If provided, then related cluster-level properties will be computed.

count_col Specifies the column of data containing a measure of abundance (such as clone count or UMI count). Accepts a character string containing the column name or a numeric scalar containing the column index. If provided, related cluster-level properties will be computed.

cluster_id_col Specifies the column of data containing the cluster membership variable that identifies the cluster to which each node belongs. Accepts a character string containing the column name or a numeric scalar containing the column index.

degree_col Specifies the column of data containing the network degree of each node. Accepts a character string containing the column name or a numeric scalar containing the column index. If the column does not exist, the network degree will be computed.

cluster_fun [Deprecated] Does nothing.

verbose Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().

Details

To use getClusterStats(), the network graph must first be partitioned into clusters, which can be done using addClusterMembership(). The name of the cluster membership variable in the node metadata must be provided to the cluster_id_col argument when calling getClusterStats().

Value

A data frame containing one row for each cluster in the network and the following variables:

cluster_id The cluster ID number.

node_count The number of nodes in the cluster.

mean_seq_length The mean sequence length in the cluster. Only present when length(seq_col) == 1.

A_mean_seq_length The mean first sequence length in the cluster. Only present when length(seq_col) == 2.

B_mean_seq_length The mean second sequence length in the cluster. Only present when length(seq_col) == 2.
mean_degree     The mean network degree in the cluster.
max_degree      The maximum network degree in the cluster.
seq_w_max_degree
                The receptor sequence possessing the maximum degree within the cluster. Only present when length(seq_col) == 1.
A_seq_w_max_degree
                The first sequence of the node possessing the maximum degree within the cluster. Only present when length(seq_col) == 2.
B_seq_w_max_degree
                The second sequence of the node possessing the maximum degree within the cluster. Only present when length(seq_col) == 2.
agg_count       The aggregate count among all nodes in the cluster (based on the counts in count_col).
max_count       The maximum count among all nodes in the cluster (based on the counts in count_col).
seq_w_max_count
                The receptor sequence possessing the maximum count within the cluster. Only present when length(seq_col) == 1.
A_seq_w_max_count
                The first sequence of the node possessing the maximum count within the cluster. Only present when length(seq_col) == 2.
B_seq_w_max_count
                The second sequence of the node possessing the maximum count within the cluster. Only present when length(seq_col) == 2.
diameter_length
                The longest geodesic distance in the cluster, computed as the length of the vector returned by get_diameter().
assortativity   The assortativity coefficient of the cluster’s graph, based on the degree (minus one) of each node in the cluster (with the degree computed based only upon the nodes within the cluster). Computed using assortativity_degree().
global_transitivity
                The transitivity (i.e., clustering coefficient) for the cluster’s graph, which estimates the probability that adjacent vertices are connected. Computed using transitivity() with type = "global".
edge_density    The number of edges in the cluster as a fraction of the maximum possible number of edges. Computed using edge_density().
degree_centrality_index
                The centrality index of the cluster’s graph based on within-cluster network degree. Computed as the centralization element of the output from centr_degree().
closeness_centrality_index
                The centrality index of the cluster’s graph based on closeness, i.e., distance to other nodes in the cluster. Computed using centralization().
eigen_centrality_index
                The centrality index of the cluster’s graph based on the eigenvector centrality scores, i.e., values of the first eigenvector of the adjacency matrix for the cluster. Computed as the centralization element of the output from centr_eigen().
getNeighborhood

 identify cells or clones in a neighborhood around a target sequence

eigen_centrality_eigenvalue

The eigenvalue corresponding to the first eigenvector of the adjacency matrix for the cluster. Computed as the value element of the output from eigen_centrality().

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package

See Also

addClusterStats() addClusterMembership() labelClusters()

Examples

set.seed(42)
toy_data <- simulateToyData()

net <-
generateNetworkObjects(
  toy_data, "CloneSeq"
)

net <- addClusterMembership(net)

net$cluster_data <-
getClusterStats(
  net$node_data,
  net$adjacency_matrix,
  seq_col = "CloneSeq",
  count_col = "CloneCount"
)

description

Given Adaptive Immune Receptor Repertoire Sequencing (AIRR-Seq) data and a target receptor sequence that is present within the data, identifies a "neighborhood" comprised of cells/clones with receptor sequences sufficiently similar to the target sequence.
getNeighborhood

Usage

getNeighborhood(
    data, 
    seq_col, 
    target_seq, 
    dist_type = "hamming", 
    max_dist = 1 
)

Arguments

data A data frame containing the AIRR-Seq data.

seq_col Specifies the column of data containing the receptor sequences. Accepts a character string containing the column name or a numeric scalar containing the column index.

target_seq A character string containing the target receptor sequence. Must be a receptor sequence possessed by one of the clones/cells in the AIRR-Seq data.

dist_type Specifies the function used to quantify the similarity between receptor sequences. The similarity between two sequences determines their pairwise distance, with greater similarity corresponding to shorter distance. Valid options are "hamming" (the default), which uses `hamDistBounded()`, and "levenshtein", which uses `levDistBounded()`.

max_dist Determines whether each cell/clone belongs to the neighborhood based on its receptor sequence’s distance from the target sequence. The distance is based on the dist_type argument. max_dist specifies the maximum distance at which a cell/clone belongs to the neighborhood. Lower values require greater similarity between the target sequence and the receptor sequences of cells/clones in its neighborhood.

Value

A data frame containing the rows of data corresponding to the cells/clones in the neighborhood.

If no cell/clone in the AIRR-Seq data possesses the target sequence as its receptor sequence, then a value of NULL is returned.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package
### Examples

```r
set.seed(42)
toy_data <- simulateToyData(sample_size = 500)

# Get neighborhood around first clone sequence
nbd <-
  getNeighborhood(
    toy_data,
    seq_col = "CloneSeq",
    target_seq = "GGGGGGGAATTGG"
  )

head(nbd)
```

---

### hamDistBounded

**Bounded Computation of Hamming Distance**

**Description**

Computes the Hamming distance between two strings subject to a specified upper bound.

**Usage**

```r
hamDistBounded(a, b, k)
```

**Arguments**

- `a`: A character string.
- `b`: A character string to be compared to `a`.
- `k`: The upper bound on the Hamming distance between `a` and `b`.

**Details**

For two character strings of equal length, the Hamming distance measures the total number of character differences between characters in corresponding positions. That is, for each position in one string, the character in that position is checked to see whether it differs from the character in the same position of the other string.

For two character strings of different lengths, the Hamming distance is not defined. However, `hamDistBounded()` will accommodate strings of different lengths, doing so in a conservative fashion that seeks to yield a meaningful result for the purpose of checking whether two strings are sufficiently similar. If the two strings differ in length, placeholder characters are appended to the shorter string until its length matches that of the longer string. Each appended placeholder character is treated as different from the character in the corresponding position of the longer string. This is effectively the same as truncating the end of the longer string and adding the number of deleted characters to the Hamming distance between the shorter string and the truncated longer string (which is what is actually done in practice, as the computation is faster).
The above method used by `hamDistBounded()` to accommodate unequal string lengths results in distance values whose meaning may be questionable, depending on context, when the two strings have different lengths. The decision to append placeholder characters to the end of the shorter string (as opposed to prepending them to the beginning) is ad hoc and somewhat arbitrary. In effect, it allows two strings of different lengths to be considered sufficiently similar if the content of the shorter string sufficiently matches the beginning content of the longer string and the difference in string length is not too great.

For comparing sequences of different lengths, the Levenshtein distance (see `levDistBounded()`) is more appropriate and meaningful than using `hamDistBounded()`, but comes at the cost of greater computational burden.

Computation is aborted early if the Hamming distance is determined to exceed the specified upper bound. This functionality is designed for cases when distinguishing between values above the upper bound is not meaningful, taking advantage of this fact to reduce the computational burden.

**Value**

An integer. If the Hamming distance exceeds the specified upper bound $k$, then a value of $-1$ is returned. Otherwise, returns the Hamming distance between $a$ and $b$.

**Note**

The computed value may be invalid when the length of either string is close to or greater than the value of `INT_MAX` in the compiler that was used at build time (typically 2147483647).

**Author(s)**

Brian Neal (<Brian.Neal@ucsf.edu>)

**References**


Webpage for the NAIR package

**See Also**

`levDistBounded()`

**Examples**

```r
# using an upper bound of 3
# (trivial since strings have length 3)
hamDistBounded("foo", "foo", 3)
hamDistBounded("foo", "fee", 3)
hamDistBounded("foo", "fie", 3)
hamDistBounded("foo", "foe", 3)
hamDistBounded("foo", "fum", 3)
hamDistBounded("foo", "bar", 3)

# using an upper bound of 1
```
# (most distances exceed the upper bound)
hamDistBounded("foo", "fee", 1)
hamDistBounded("foo", "fie", 1)
hamDistBounded("foo", "foe", 1)
hamDistBounded("foo", "fum", 1)
hamDistBounded("foo", "bar", 1)

# comparing strings of nonmatching length
hamDistBounded("foo", "fubar", 10)
hamDistBounded("foo", "foobar", 10)
hamDistBounded("foo", "barfoo", 10)

---

**labelClusters**

*Label Clusters in a Network Graph Plot*

**Description**

Functions for labeling the clusters in network graph plots with their cluster IDs. The user can specify a cluster-level property by which to rank the clusters, labeling only those clusters above a specified rank.

**Usage**

```r
labelClusters(
  net,
  plots = NULL,
  top_n_clusters = 20,
  cluster_id_col = "cluster_id",
  criterion = "node_count",
  size = 5, color = "black",
  greatest_values = TRUE
)
```

```r
addClusterLabels(
  plot,
  net,
  top_n_clusters = 20,
  cluster_id_col = "cluster_id",
  criterion = "node_count",
  size = 5,
  color = "black",
  greatest_values = TRUE
)
```

**Arguments**

- **net**
  A list of network objects conforming to the output of `buildRepSeqNetwork()` or `generateNetworkObjects()`. See details.
labelClusters

- **plots** Specifies which plots in net$plots to annotate. Accepts a character vector of element names or a numeric vector of element position indices. The default NULL annotates all plots.
- **plot** A `gggraph` object containing the network graph plot.
- **top_n_clusters** A positive integer specifying the number of clusters to label. Those with the highest rank according to the criterion argument will be labeled.
- **cluster_id_col** Specifies the column of net$node_data containing the variable for cluster membership. Accepts a character string containing the column name.
- **criterion** Can be used to specify a cluster-level network property by which to rank the clusters. Non-default values are ignored unless net$cluster_data exists and corresponds to the cluster membership variable specified by cluster_id_col. Accepts a character string containing a column name of net$cluster_data. The property must be quantitative for the ranking to be meaningful. By default, clusters are ranked by node count, which is computed based on the cluster membership values if necessary.
- **size** The font size of the cluster ID labels. Passed to the size argument of `geom_node_text()`.  
- **color** The color of the cluster ID labels. Passed to the color argument of `geom_node_text()`.  
- **greatest_values** Logical. Controls whether clusters are ranked according to the greatest or least values of the property specified by the criterion argument. If TRUE, clusters with greater values will be ranked above those with lower values, thereby receiving a higher priority to be labeled.

**Details**

The list net must contain the named elements igraph (of class `igraph`), adjacency_matrix (a `matrix` or `dgCMatrix` encoding edge connections), and node_data (a `data.frame` containing node metadata), all corresponding to the same network. The lists returned by `buildRepSeqNetwork()` and `generateNetworkObjects()` are examples of valid inputs for the net argument.

**Value**

`labelClusters()` returns a copy of net with the specified plots annotated.  
`addClusterLabels()` returns an annotated copy of plot.

**Author(s)**

Brian Neal (<Brian.Neal@ucsf.edu>)

**References**

Webpage for the NAIR package

**See Also**

`addClusterMembership()`, `getClusterStats()`, `generateNetworkGraphPlots()`
Examples

```r
set.seed(42)
toy_data <- simulateToyData()

network <- buildRepSeqNetwork(
  toy_data, "CloneSeq",
  cluster_stats = TRUE,
  color_nodes_by = "cluster_id",
  color_scheme = "turbo",
  color_legend = FALSE,
  plot_title = NULL,
  plot_subtitle = NULL,
  size_nodes_by = 1
)

network <- labelClusters(network)

network$plots$cluster_id
```

labelNodes

Label Nodes in a Network Graph Plot

Description

Functions for annotating a graph plot to add custom labels to the nodes.

Usage

```r
labelNodes(
  net, 
  node_labels, 
  plots = NULL, 
  size = 5, 
  color = "black"
)
```

```r
addGraphLabels(
  plot, 
  node_labels, 
  size = 5, 
  color = "black"
)
```

Arguments

- **net**: A list of network objects conforming to the output of `buildRepSeqNetwork()` or `generateNetworkObjects()`. See details.
- **plot**: A `ggraph` object containing the network graph plot.
### labelNodes

- **node_labels**: A vector containing the node labels, where each entry is the label for a single node. The length should match the number of nodes in the plot.
- **plots**: Specifies which plots in `net$plots` to annotate. Accepts a character vector of element names or a numeric vector of element position indices. The default `NULL` annotates all plots.
- **size**: The font size of the node labels. Passed to the `size` argument of `geom_node_text()`.
- **color**: The color of the node labels. Passed to the `size` argument of `geom_node_text()`.

### Details

The list `net` must contain the named elements `igraph` (of class `igraph`), `adjacency_matrix` (a `matrix` or `dgCMatrix` encoding edge connections), and `node_data` (a `data.frame` containing node metadata), all corresponding to the same network. The lists returned by `buildRepSeqNetwork()` and `generateNetworkObjects()` are examples of valid inputs for the `net` argument.

Labels are added using `geom_node_text()`.

### Value

`labelNodes()` returns a copy of `net` with the specified plots annotated.

`addGraphLabels()` returns a `ggraph` object containing the original plot annotated with the node labels.

### Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

### References


Webpage for the NAIR package

### Examples

```r
set.seed(42)
toy_data <- simulateToyData(
  samples = 1,
  sample_size = 10,
  prefix_length = 1
)

# Generate network
network <- buildNet(
  toy_data,
  seq_col = "CloneSeq",
  plot_title = NULL,
  plot_subtitle = NULL
)```
# Label each node with its receptor sequence
network <- labelNodes(network, "CloneSeq", size = 3)

network$plots[[1]]
Note

The computed value may be invalid when the length of either string is close to or greater than the value of INT_MAX in the compiler that was used at build time (typically 2147483647).

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References


Webpage for the NAIR package

See Also

hamDistBounded

Examples

# equal string lengths,
# character transmutations only
levDistBounded("foo", "bar", 3)
hamDistBounded("foo", "bar", 3) # agrees with Hamming distance

# one insertion, one deletion
levDistBounded("1234567", "1.23457", 7)
hamDistBounded("1234567", "1.23457", 7) # compare to Hamming distance

# same as above, but with a different lower bound
levDistBounded("1234567", "1.23457", 3) # within the bound
hamDistBounded("1234567", "1.23457", 3) # exceeds the bound

# one deletion (last position)
levDistBounded("1234567890", "123456789", 10)
hamDistBounded("1234567890", "123456789", 10)

# note the Hamming distance agrees with the Levenshtein distance
# for the above example, since the deletion occurs in the final
# character position. This is due to how hamDistBounded() handles
# strings of different lengths. In the example below, however...

# one deletion (first position)
levDistBounded("1234567890", "234567890", 10)
hamDistBounded("1234567890", "234567890", 10) # compare to Hamming distance

# one deletion, one transmutation
levDistBounded("foobar", "fubar", 6)
hamDistBounded("foobar", "fubar", 6) # compare to Hamming distance
plotNetworkGraph  Plot the Graph of an Immune Repertoire Network

Description

Given the igraph of an immune repertoire network, generates a plot of the network graph according to the user specifications.

Deprecated. Replaced by addPlots().

Usage

plotNetworkGraph(
  igraph,
  plot_title = NULL,
  plot_subtitle = NULL,
  color_nodes_by = NULL,
  color_scheme = "default",
  color_legend = "auto",
  color_title = "auto",
  edge_width = 0.1,
  size_nodes_by = 0.5,
  node_size_limits = NULL,
  size_title = "auto",
  outfile = NULL,
  pdf_width = 12,
  pdf_height = 8
)

Arguments

igraph An object of class igraph.
plot_title A character string containing the plot title. Passed to labs().
plot_subtitle A character string containing the plot subtitle. Passed to labs().
color_nodes_by A vector whose length matches the number of nodes in the network. The values are used to encode the color of each node. An argument value of NULL (the default) leaves the nodes uncolored. Passed to the color aesthetic mapping of geom_node_point().
color_scheme A character string specifying the color scale used to color the nodes. "default" uses default ggplot() colors. Other options are one of the viridis color scales (e.g., "plasma", "A" or other valid inputs to the option argument of scale_color_viridis()) or (for discrete variables) a palette from hcl.pals() (e.g., "RdYlGn"). Each of the viridis color scales can include the suffix "-1" to reverse its direction (e.g., "plasma-1" or "A-1").
color_legend A logical scalar specifying whether to display the color legend in the plot. The default value of "auto" shows the color legend if color_nodes_by is a continuous variable or a discrete variable with at most 20 distinct values.
color_title  A character string (or NULL) specifying the title for the color legend. Only applicable if color_nodes_by is a vector. If color_title = "auto" (the default), the title for the color legend will be the name of the vector provided to color_nodes_by.

group_width  A numeric scalar specifying the width of the graph edges in the plot. Passed to the width argument of geom_edge_link().

size_nodes_by  A numeric scalar specifying the size of the nodes, or a numeric vector with positive entries that encodes the size of each node (and whose length matches the number of nodes in the network). Alternatively, an argument value of NULL uses the default ggraph() size for all nodes. Passed to the size aesthetic mapping of geom_node_point().

size_title  A character string (or NULL) specifying the title for the size legend. Only applicable if size_nodes_by is a vector. If size_title = "auto" (the default), the title for the color legend will be the name of the vector provided to size_nodes_by.

node_size_limits  A numeric vector of length 2, specifying the minimum and maximum node size. Only applicable if size_nodes_by is a vector. If node_size_limits = NULL, the default size scale will be used.

outfile  An optional file path for saving the plot as a pdf. If NULL (the default), no pdf will be saved.

pdf_width  Sets the plot width when writing to pdf. Passed to the width argument of pdf().

pdf_height  Sets the plot height when writing to pdf. Passed to the height argument of pdf().

Value  
A ggraph object.

Author(s)  
Brian Neal (<Brian.Neal@ucsf.edu>)

References  

Webpage for the NAIR package  
Network Visualization article on package website

See Also  
addPlots()
Examples

set.seed(42)
toy_data <- simulateToyData()

# Generate network for data
net <- buildNet(toy_data, "CloneSeq")

# Plot network graph
net_plot <- plotNetworkGraph(
  net$igraph,
  color_nodes_by =
    net$node_data$SampleID,
  color_title = NULL,
  size_nodes_by =
    net$node_data$CloneCount,
  size_title = "Clone Count",
  node_size_limits = c(0.5, 1.5))

print(net_plot)

---

saveNetwork  Save List of Network Objects

Description

Given a list of network objects such as that returned by buildRepSeqNetwork() or generateNetworkObjects, saves its contents according to the specified file format scheme.

Usage

saveNetwork(
  net,
  output_dir,
  output_type = "rds",
  output_name = "MyRepSeqNetwork",
  pdf_width = 12,
  pdf_height = 10,
  verbose = FALSE,
  output_filename = deprecated()
)

Arguments

net A list of network objects returned by buildRepSeqNetwork() or generateNetworkObjects.
output_dir A file path specifying the directory in which to write the file(s).
output_type A character string specifying the file format scheme to use when writing output to file. Valid options are "individual", "rds" and "rda". See details.
output_name  A character string. All files saved will have file names beginning with this value.

pdf_width    If the list contains plots, this controls the width of each plot when writing to pdf. Passed to the width argument of the pdf function.

pdf_height   If the list contains plots, this controls the height of each plot when writing to pdf. Passed to the height argument of the pdf function.

verbose      Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().

output_filename

[Deprecated] Equivalent to output_name.

Details

The list net must contain the named elements igraph (of class igraph), adjacency_matrix (a matrix or dgCMATRIX encoding edge connections), and node_data (a data.frame containing node metadata), all corresponding to the same network. The list returned by buildRepSeqNetwork() and generateNetworkObjects() is an example of a valid input for the net argument.

The additional elements cluster_data (a data.frame) and plots (a list containing objects of class ggraph and possibly one matrix named graph_layout) will also be saved, if present.

By default, the list net is saved to a compressed data file in the RDS format, while any plots present are printed to a single pdf containing one plot per page.

The name of each saved file begins with the value of output_name. When output_type is one of "rds" or "rda", only two files are saved (the rds/rda and the pdf); for each file, output_name is followed by the appropriate file extension.

When output_type = "individual", each element of net is saved as a separate file, where output_name is followed by:

- _NodeMetadata.csv for node_data
- _ClusterMetadata.csv for cluster_data
- _EdgeList.txt for igraph
- _AdjacencyMatrix.mtx for adjacency_matrix
- _Plots.rda for plots
- _GraphLayout.txt for plots$graph_layout
- _Details.rds for details

node_data and cluster_data are saved using write.csv(), with row.names being TRUE for node_data and FALSE for cluster_data. The igraph is saved using write_graph() with format = "edgelist". The adjacency matrix is saved using writeMM(). The graph layout is saved using write() with ncolumns = 2.

Value

Returns TRUE if output is saved, otherwise returns FALSE (with a warning if output_dir is non-null and the specified directory does not exist and could not be created).
saveNetworkPlots

Author(s)
Brian Neal (<Brian.Neal@ucsf.edu>)

References

Webpage for the NAIR package

Examples

```r
set.seed(42)
toy_data <- simulateToyData()

net <- buildRepSeqNetwork(
  toy_data,
  seq_col = "CloneSeq",
  node_stats = TRUE,
  cluster_stats = TRUE,
  color_nodes_by = c("transitivity", "SampleID")
)

# save as single RDS file
saveNetwork(
  net,
  output_dir = tempdir(),
  verbose = TRUE
)

saveNetwork(
  net,
  output_dir = tempdir(),
  output_type = "individual",
  verbose = TRUE
)
```

---

**saveNetworkPlots**      **Write Plots to a PDF**

**Description**
Given a list of plots, write all plots to a single pdf file containing one plot per page, and optionally save the graph layout as a csv file.
Usage

saveNetworkPlots(
  plotlist,
  outfile,
  pdf_width = 12,
  pdf_height = 10,
  outfile_layout = NULL,
  verbose = FALSE
)

Arguments

plotlist  A named list whose elements are of class ggraph. May also contain an element named graph_layout with the matrix specifying the graph layout.
outfile  A connection or a character string containing the file path used to save the pdf.
pdf_width  Sets the page width. Passed to the width argument of pdf().
pdf_height  Sets the page height. Passed to the height argument of pdf().
outfile_layout  An optional connection or file path for saving the graph layout. Passed to the file argument of write(), which is called with ncolumns = 2.
verbose  Logical. If TRUE, generates messages about the tasks performed and their progress, as well as relevant properties of intermediate outputs. Messages are sent to stderr().

Value

Returns TRUE, invisibly.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)

References

Webpage for the NAIR package

Examples

```r
set.seed(42)
toy_data <- simulateToyData()

net <-
  generateNetworkObjects(
    toy_data,
    "CloneSeq"
  )
```
simulateToyData

Generate Toy AIRR-Seq Data

Description

Generates toy data that can be used to test or demonstrate the behavior of functions in the NAIR package. Created as a lightweight tool for use in tests, examples and vignettes. This function is not intended to simulate realistic data.

Usage

simulateToyData(
  samples = 2,
  chains = 1,
  sample_size = 100,
  prefix_length = 7,
  prefix_chars = c("G", "A", "T", "C"),
  prefix_probs = rbind(
    "sample1" = c(12, 4, 1, 1),
    "sample2" = c(4, 12, 1, 1)),
  affixes = c("AATTGG", "AATCGG", "AATTCG",
             "AATTGC", "AATTG", "AATTC"),
  affix_probs = rbind(
    "AATTGG" = c(1, 1, 1, 1, 1),
    "AATCGG" = c(1, 1, 1, 1, 1),
    "AATTCG" = c(1, 1, 1, 1, 1),
    "AATTGC" = c(1, 1, 1, 1, 1),
    "AATTG" = c(1, 1, 1, 1, 1),
    "AATTC" = c(1, 1, 1, 1, 1))
)
simulateToyData

```
"sample1" = c(10, 4, 2, 2, 1, 1),
"sample2" = c(1, 1, 2, 2.5, 2.5)),
num_edits = 0,
edit_pos_probs = function(seq_length) {
  stats::dnorm(seq(-4, 4, length.out = seq_length))
},
edit_ops = c("insertion", "deletion", "transmutation"),
edit_probs = c(5, 1, 4),
new_chars = prefix_chars,
new_probs = prefix_probs,
output_dir = NULL,
no_return = FALSE
```

Arguments

- **samples**: The number of distinct samples to include in the data.
- **chains**: The number of chains (either 1 or 2) for which to generate receptor sequences.
- **sample_size**: The number of observations to generate per sample.
- **prefix_length**: The length of the random prefix generated for each observed sequence. Specifically, the number of elements of `prefix_chars` that are sampled with replacement and concatenated to form each prefix.
- **prefix_chars**: A character vector containing characters or strings from which to sample when generating the prefix for each observed sequence.
- **prefix_probs**: A numeric matrix whose column dimension matches the length of `prefix_chars` and with row dimension matching the value of `samples`. The `i`th row specifies the relative probability weights assigned to each element of `prefix_chars` when sampling to form the prefix for each sequence in the `i`th sample.
- **affixes**: A character vector containing characters or strings from which to sample when generating the suffix for each observed sequence.
- **affix_probs**: A numeric matrix whose column dimension matches the length of `affixes` and with row dimension matching the value of `samples`. The `i`th row specifies the relative probability weights assigned to each element of `affixes` when sampling to form the suffix for each sequence in the `i`th sample.
- **num_edits**: A nonnegative integer specifying the number of random edit operations to perform on each observed sequence after its initial generation.
- **edit_pos_probs**: A function that accepts a nonnegative integer (the character length of a sequence) as its argument and returns a vector of this length containing probability weights. Each time an edit operation is performed on a sequence, the character position at which to perform the operation is randomly determined according to the probabilities given by this function.
- **edit_ops**: A character vector specifying the possible operations that can be performed for each edit. The default value includes all valid operations (insertion, deletion, transmutation).
- **edit_probs**: A numeric vector of the same length as `edit_ops`, specifying the relative probability weights assigned to each edit operation.
simulateToyData

new_chars  A character vector containing characters or strings from which to sample when performing an insertion edit operation.

new_probs  A numeric matrix whose column dimension matches the length of new_chars and with row dimension matching the value of samples. The ith row specifies, for the ith sample, the relative probability weights assigned to each element of new_chars when performing a transmutation or insertion as a random edit operation.

output_dir  An optional character string specifying a file directory to save the generated data. One file will be generated per sample.

no_return  A logical flag that can be used to prevent the function from returning the generated data. If TRUE, the function will instead return TRUE once all processes are complete.

Details

Each observed sequence is obtained by separately generating a prefix and suffix according to the specified settings, then joining the two and performing sequential rounds of edit operations randomized according to the user’s specifications.

Count data is generated for each observation; note that this count data is generated independently from the observed sequences and has no relationship to them.

Value

If no_return = FALSE (the default), a data.frame whose contents depend on the value of the chains argument.

For chains = 1, the data frame contains the following variables:

CloneSeq  The "receptor sequence" for each observation.

CloneFrequency  The "clone frequency" for each observation (clone count as a proportion of the aggregate clone count within each sample).

CloneCount  The "clone count" for each observation.

SampleID  The sample ID for each observation.

For chains = 2, the data frame contains the following variables:

AlphaSeq  The "alpha chain" receptor sequence for each observation.

BetaSeq  The "beta chain" receptor sequence for each observation.

UMIs  The "unique molecular identifier count" for each observation.

Count  The "count" for each observation.

SampleID  The sample ID for each observation.

If no_return = TRUE, the function returns TRUE upon completion.

Author(s)

Brian Neal (<Brian.Neal@ucsf.edu>)
Examples

```r
set.seed(42)

# Bulk data from two samples
dat1 <- simulateToyData()

# Single-cell data with alpha and beta chain sequences
dat2 <- simulateToyData(chains = 2)

# Write data to file, return nothing
simulateToyData(sample_size = 500,
  num_edits = 10,
  no_return = TRUE,
  output_dir = tempdir())

# Example customization
dat4 <-
simulateToyData(
  samples = 5,
  sample_size = 50,
  prefix_length = 0,
  prefix_chars = "",
  prefix_probs = matrix(1, nrow = 5),
  affixes = c("CASSLGYEQYF", "CASSLGETQYF",
              "CASSLGDQTQYF", "CASSLGETAQEF",
              "CASSLGDRTQYF", "CAGLGRDQETQYF",
              "CASSSTQYTF", "CASSLTDQTQYF",
              "CANGYTF", "CANTGELFF",
              "CSANYGYTF"),
  affix_probs = matrix(1, ncol = 11, nrow = 5),
)

## Simulate 30 samples with a mix of public/private sequences ##
samples <- 30
sample_size <- 30 # (seqs per sample)
base_freqs <- c(
  "CASSIEGQLSTDTQYF", "CASSEEGQLSTDTQYF", "CASSSVETQYF",
  "CASSPEQQLSTDTQYF", "RASSLAGEAFF", "CASHRGTDQYF", "CASDAGVFQPOHF",
  "CASSLTSGYNEQFF", "CASSETGYNEQFF", "CASSLTGNEQFF", "CASSLYTGYNEQFF",
  "CASSLTGNEQFF", "CASSLNGYNEQFF", "CASSFPWDQGYYTF", "CASTLRQGELFF",
  "CASTLSRQGELFF", "CSVELLTPGPEYSNEQFF", "CSVELLTPGSETSNEQFF",
  "CVEELLTPGSETSNEQFF", "CASSLRAGRTQETQYF", "CASHAGRTQETQYF",
  "CASSLAGRTQETQYF", "CASSRAGRTQETQYF", "CASSRAGRTQETQYF",
  "CASSQYGGGQPOHF", "CASSLGGGQPOHF", "CASSNGGQPOHF", "CASSYGGGQPOHF",
  "CASSYGGGQPOHF", "CASSYKGGQPOHF", "CASSYTGGQPOHF",
  "CAWSSQETQYF", "CASSSPETQYF", "CASSGAYEQYF", "CSVDLGKGNNEQFF")
# Relative generation probabilities
gen <- cbind(
  stats::toeplitz(0.6^(0:(sample_size - 1))),
  matrix(1, nrow = samples, ncol = length(base_freqs) - samples))
dat5 <-
simulateToyData(

```
samples = samples,
sample_size = sample_size,
prefix_length = 1,
prefix_chars = c("",""),
prefix_probs = cbind(rep(1, samples), rep(0, samples)),
affixes = base_seqs,
affix_probs = pgen,
num_edits = 0
)

### Simulate 30 samples from two groups (treatment/control) ###
samples_c <- samples_t <- 15 # Number of samples by control/treatment group
samples <- samples_c + samples_t
sample_size <- 30 # (seqs per sample)
base_seqs <- # first five are associated with treatment
c("CASSGAYEQYF", "CSVDLGKGNNEQFF", "CASSIEGQLSTDQYF",
   "CASSEEGQLSTDQYF", "CASSPEGQLSTDQYF",
   "RASSLAGNTEAFF", "CASSHRGTSTDQYF", "CASDAGVFQPQHF")
# Relative generation probabilities by control/treatment group
pgen_c <- matrix(rep(c(rep(1, 5), rep(30, 3)), times = samples_c),
   nrow = samples_c, byrow = TRUE)
pgen_t <- matrix(rep(c(1, 1, rep(1/3, 3), rep(2, 3)), times = samples_t),
   nrow = samples_t, byrow = TRUE)
pgen <- rbind(pgen_c, pgen_t)
dat6 <-
simulateToyData(
   samples = samples,
   sample_size = sample_size,
   prefix_length = 1,
   prefix_chars = c("",""),
   prefix_probs =
       cbind(rep(1, samples), rep(0, samples)),
   affixes = base_seqs,
   affix_probs = pgen,
   num_edits = 0
)
Index

addClusterLabels (labelClusters), 81
addClusterMembership, 3
addClusterMembership(), 7–10, 12, 14, 40, 44, 75, 77, 82
addClusterStats, 6
addClusterStats(), 6, 39, 74, 77
addGraphLabels (labelNodes), 83
addNodeNetworkStats, 11, 32, 43
addNodeStats, 14
addNodeStats(), 12, 13, 44, 45
addPlots, 16
addPlots(), 28, 32, 33, 40, 87, 88
aggregateIdenticalClones, 20
assortativity_degree(), 9, 76
authority_score(), 44
betweenness(), 44
buildAssociatedClusterNetwork, 23
buildAssociatedClusterNetwork(), 53, 56, 61
buildNet (buildRepSeqNetwork), 37
buildNet(), 28, 66
buildPublicClusterNetwork, 26
buildPublicClusterNetwork(), 31, 33, 36, 66, 67
buildPublicClusterNetworkByRepresentative, 31
buildPublicClusterNetworkByRepresentative(), 29, 66, 67
buildRepSeqNetwork, 37
buildRepSeqNetwork(), 3, 4, 6–8, 12, 14, 15, 17, 19, 24, 28, 29, 32, 33, 43, 44, 65, 81–84, 89, 90
centr_betw(), 44
centr_clo(), 44
centr_degree(), 9, 76
centr_eigen(), 9, 44, 76
centralization(), 9, 76
chooseNodeStats, 43
chooseNodeStats(), 12–15, 40
closeness(), 44
combineSamples, 46
correlation, 92
connections, 23, 27, 46, 54, 59, 64
coreness(), 44
data.frame, 4, 8, 15, 19, 82, 84, 90
dgCMatrix, 4, 8, 15, 19, 82, 84, 90
dimnames, 70
density(), 9, 76
eigen_centrality(), 9, 44, 77
exclusiveNodeStats (chooseNodeStats), 43
exclusiveNodeStats(), 12, 14, 40
equivalence, 51
filterInputData, 52
filterInputData(), 39, 47, 55, 60
findAssociatedClones, 53
findAssociatedClones(), 23–25, 61
findAssociatedSeqs, 57
findAssociatedSeqs(), 25, 53, 56
findAssociatedSeqs2
  (findAssociatedSeqs), 57
findPublicClusters, 63
findPublicClusters(), 26, 29, 31, 36
fisher.test(), 60
generateAdjacencyMatrix, 68
generateNetworkFromAdjacencyMat
generateNetworkGraph, 71
generateNetworkGraph, 71
generateNetworkGraphPlots (addPlots), 16
generateNetworkGraphPlots(), 82
generateNetworkObjects, 72, 89
generateNetworkObjects(), 3, 4, 6–8, 12, 14, 15, 17, 19, 81–84, 89, 90
generateNetworkGraphPlots, 16
geom_edge_link0(), 18, 88
geom_node_point(), 18, 87, 88

98
geom_node_text(), 82, 84
get_diameter(), 9, 76
getClusterStats, 32, 35, 74
getClusterStats(), 10, 82
getNeighborhood, 77
ggplot(), 18, 87
graph, 16, 18, 42, 51, 82–84, 88, 90, 92
graph(), 88
graph_from_adjacency_matrix(), 71
grep(), 39, 52, 55, 60, 65
hamDistBounded, 69, 79, 86
hamDistBounded(), 39, 55, 72, 78, 85
hcl.pals(), 18, 87
igraph, 3, 4, 8, 12, 15–17, 19, 41, 71, 73, 82, 84, 87, 90
labelClusters, 81
labelClusters(), 6, 10, 20, 77
labelNodes, 83
labelNodes(), 20
labs(), 87
levDistBounded, 69, 85
levDistBounded(), 39, 55, 72, 78, 80
list, 3, 7, 14, 17, 81, 83
load(), 47
loadDataFromFileList (combineSamples), 46
loadDataFromFileList(), 23, 27, 32, 54, 59, 64
matrix, 4, 8, 15, 19, 82, 84, 90
NAIR (NAIR-package), 2
NAIR-package, 2
node_stat_settings (chooseNodeStats), 43
page_rank(), 44
pdf, 90
pdf(), 88, 92
plotNetworkGraph, 87
read.csv(), 24, 27, 28, 32, 47, 54, 59, 64
read.csv2(), 47
read.delim(), 48
read.table(), 24, 27, 28, 32, 47, 48, 54, 59, 64
readRDS(), 47
regex, 39, 52, 55, 60, 65
saveNetwork, 89
saveNetwork(), 33, 41
saveNetworkPlots, 91
saveNetworkPlots(), 19, 20
scale_color_viridis(), 18, 87
simulateToyData, 93
sparseAdjacencyMatFromSeqs
  (generateAdjacencyMatrix), 68
stderr, 69
stderr(), 4, 8, 12, 15, 18, 21, 24, 28, 33, 41,
  47, 52, 55, 60, 66, 73, 75, 90, 92
transitivity(), 9, 44, 76
write(), 90, 92
write.csv(), 60, 90
write.table(), 55
write_graph(), 90
writeMM(), 90