Package ‘netcom’

June 4, 2024

Type Package

Title NETwork COMparison Inference

Version 2.1.7

Date 2024-5-28

Description Infer system functioning with empirical NETwork COMparisons. These methods are part of a growing paradigm in network science that uses relative comparisons of networks to infer mechanistic classifications and predict systemic interventions. They have been developed and applied in Langendorf and Burgess (2021) <doi:10.1038/s41598-021-99251-7>, Langendorf (2020) <doi:10.1201/9781351190831-6>, and Langendorf and Goldberg (2019) <doi:10.48550/arXiv.1912.12551>.

URL `https://github.com/langendorfr/netcom`

Repository CRAN

License GPL-3

Encoding UTF-8

Depends R (>= 3.1.0)

Imports stats, dplyr, tibble, clue, expm, igraph, Matrix, pdist, pracma, vegan, magrittr, foreach, parallel, doParallel, optimx, GenSA, rlang, ggfortify, ggplot2, ggraph, reshape2

RoxygenNote 7.3.1

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Ryan Langendorf [aut, cre],
Debra Goldberg [ctb],
Matthew Burgess [ctb]

Maintainer Ryan Langendorf <`ryan.langendorf@colorado.edu`>

Date/Publication 2024-06-04 17:50:05 UTC

Contents

`align` *Network Alignment*

Description

Network alignment by comparing the entropies of diffusion kernels simulated on two networks. `align` takes two networks stored as matrices and returns a node-level alignment between them.

Usage

```
align(
  network_1_input,
  network_2_input,
  base = 2,
  max_duration,
  characterization = "entropy",
  normalization = FALSE,
  unit_test = FALSE
)
```

Arguments

`network_1_input`	The first network being aligned, which must be in matrix form. If the two networks are of different sizes, it will be easier to interpret the output if this is the smaller one.
`network_2_input`	The second network, which also must be a matrix.
`base`	Defaults to 1. The base in the series of time steps to sample the diffusion kernels at. If base = 1 every time step is sampled. If base = 2, only time steps that are powers of 2 are sampled, etc. Larger values place more emphasis on earlier time steps. This can be helpful if the diffusion kernel quickly converges to an equilibrium, and also runs faster.
`max_duration`	Defaults to twice the diameter of the larger network. Sets the number of time steps to allow the diffusion kernel to spread for, which is the smallest power of base that is at least as large as max_duration.
`characterization`	Defaults to "entropy". Determines how the diffusion kernels are characterized. Either "entropy" or "gini". "entropy" is a size-normalized version of Shannon's entropy with base e (Euler's number). This is also known as interaction or species evenness in ecology. "gini" is the Gini coefficient.
`normalization`	Defaults to FALSE. Determines if self-loops should be augmented such that edge weights are proportional to those in network_1_input and network_2_input. FALSE by default because this is inappropriate for unweighted binary/logical networks where edges indicate only the presence of an interaction.
`unit_test`	Defaults to FALSE. Saves the following intermediate steps to help with general troubleshooting: post-processing matrix representations of both networks, time steps at which the diffusion kernels were sampled, the diffusion kernels at those time steps, the characterizations of the diffusion kernels at those time steps, and the cost matrix fed into the Hungarian algorithm where the ij element is the difference between the characterization-over-time curves for node i in the first network and node j in the second network.

Details

Network alignment pairs nodes between two networks so as to maximize similarities in their edge structures. This allows information from well-studied systems to be used in poorly studied ones, such as to identify unknown protein functions or ecosystems that will respond similarly to a given disturbance. Most network alignment algorithms focus on measures of topological overlap between edges of the two networks. The method implemented here compares nodes using the predictability of dynamics originating from each node in each network. Consider network alignment as trying to compare two hypothetical cities of houses connected by roads. The approach implemented here is to pairwise compare each house with those in the other city by creating a house-specific signature. This is accomplished by quantifying the predictability of the location of a person at various times after they left their house, assuming they were moving randomly. This predictability across all houses captures much of the way each city is organized and functions. `align` uses this conceptual rationale to align two networks, with nodes as houses, edges as roads, and random diffusion representing people leaving their houses and walking around the city to other houses. The mechanics of this, which are conceptually akin to flow algorithms and Laplacian dynamics, can be analytically expressed as a Markov chain raised to successive powers which are the durations of diffusion.

Note that the novel part of `align` lies in creating a matrix where the ij entry is a measure of similarity between node i in the first network and node j in the second. The final alignment is found using `solve_LSAP` in the package `clue`, which uses the Hungarian algorithm to solve the assignment problem optimally.

Value

`score`	Mean of all alignment scores between nodes in both original networks network_1_input and network_2_input.
`alignment`	Data frame of the nodes in both networks, sorted numerically by the first network (why it helps to make the smaller network the first one), and the corresponding alignment score.
`score_with_padding`	Same as score but includes the padding nodes in the smaller network, which can be thought of as a size gap penalty for aligning differently sized networks. Only included if the input networks are different sizes.
`alignment_with_padding`	Same as alignment but includes the padding nodes in the smaller network. Only included if the input networks are different sizes.

References

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. Naval Research Logistics (NRL), 2(1-2), 83-97.

Langendorf, R. E., & Goldberg, D. S. (2019). Aligning statistical dynamics captures biological network functioning. arXiv preprint arXiv:1912.12551.

C. Papadimitriou and K. Steiglitz (1982), Combinatorial Optimization: Algorithms and Complexity. Englewood Cliffs: Prentice Hall.

Examples

```
# The two networks to be aligned
net_one <- matrix(stats::runif(25,0,1), nrow=5, ncol=5)
net_two <- matrix(stats::runif(25,0,1), nrow=5, ncol=5)
```

```
align(net_one, net_two)
align(net_one, net_two, base = 1, characterization = "gini", normalization = TRUE)
```

`best_fit_optim` *Empirical parameterization*

Description

Helper function to find the best fitting version of a mechanism by searching across its parameter space

Usage

```
best_fit_optim(
  parameter,
  process,
  network,
  net_size,
  net_kind,
  mechanism_kind,
  resolution,
  resolution_min,
  resolution_max,
  reps,
  power_max,
  connectance_max,
  divergence_max,
  mutation_max,
  cores,
  directed,
  method,
  cause_orientation,
  DD_kind,
  DD_weight,
  max_norm,
  best_fit_kind = "avg",
  verbose = FALSE
)
```

Arguments

`parameter`	The parameter being tested for its ability to generate networks alike the input ‘network‘.
`process`	Name of mechanism. Currently only "ER", "PA", "DD", "DM" "SW", and "NM" are supported. Future versions will accept user-defined network-generating

	functions and associated parameters. ER = Erdos-Renyi random. PA = Preferential Attachment. DD = Duplication and Divergence. DM = Duplication and Mutation. SW = Small World. NM = Niche Model.
`network`	The network being compared to a hypothesized 'process' with a given 'parameter' value.
`net_size`	Number of nodes in the network.
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list").
`mechanism_kind`	Either "canonical" or "grow" can be used to simulate networks. If "grow" is used, note that here it will only simulate pure mixtures made of a single mechanism.
`resolution`	The first step is to find the version of each process most similar to the target network. This parameter sets the number of parameter values to search across. Decrease to improve performance, but at the cost of accuracy.
`resolution_min`	= The minimum parameter value to consider. Zero is not used because in many processes it results in degenerate systems (e.g. entirely unconnected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process.
`resolution_max`	The maximum parameter value to consider. One is not used because in many processes it results in degenerate systems (e.g. entirely connected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process.
`reps`	The number of networks to simulate for each parameter. More replicates increases accuracy by making the estimation of the parameter that produces networks most similar to the target network less idiosyncratic.
`power_max`	The maximum power of attachment in the Preferential Attachment process (PA).
`connectance_max`	The maximum connectance parameter for the Niche Model.
`divergence_max`	The maximum divergence parameter for the Duplication and Divergence/Mutation mechanisms.
`mutation_max`	The maximum mutation parameter for the Duplication and Mutation mechanism.
`cores`	The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`directed`	Whether the target network is directed.
`method`	This determines the method used to compare networks at the heart of the classification. Currently "DD" (Degree Distribution) and "align" (the align function which compares networks by the entropy of diffusion on them) are supported. Future versions will allow user-defined methods.
`cause_orientation`	The orientation of directed adjacency matrices.
`DD_kind`	A vector of network properties to be used to compare networks.
`DD_weight`	Weights of each network property in DD_kind. Defaults to 1, which is equal weighting for each property.

`max_norm`	Binary variable indicating if each network property should be normalized so its max value (if a node-level property) is one.
`best_fit_kind`	How to aggregate the stochastic replicates of the process + parameter combination.
`verbose`	Defaults to TRUE. Whether to print all messages.

Details

Note: Currently each process is assumed to have a single governing parameter.

Value

A number measuring how different the input network is from the parameter + process combination.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

# Adjacency matrix
size <- 10
network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)

# Calculate how similar the input network is to Small-World networks with
# a rewiring probability of 0.28.
best_fit_optim(
     parameter = 0.28,
     process = "SW",
     network = network,
     net_size = 12,
     net_kind = "matrix",
     mechanism_kind = "grow",
     resolution = 100,
     resolution_min = 0.01,
     resolution_max = 0.99,
     reps = 3,
     power_max = 5,
     connectance_max = 0.5,
     divergence_max = 0.5,
     mutation_max = 0.5,
     cores = 1,
     directed = TRUE,
     method = "DD",
     cause_orientation = "row",
     DD_kind = c(
         "in", "out", "entropy_in", "entropy_out",
         "clustering_coefficient", "page_rank", "communities"
     ),
     DD_weight = 1,
     max_norm = FALSE,
     verbose = FALSE
)
```

`classify` *Mechanistic Network Classification*

Description

Tests a network against hypothetical generating processes using a comparative network inference.

Usage

```
classify(
  network,
  directed,
  method = "DD",
  net_kind = "matrix",
  mechanism_kind = "canonical",
  DD_kind = c("in", "out", "entropy_in", "entropy_out", "clustering_coefficient",
    "page_rank", "communities", "motifs_3", "motifs_4", "eq_in", "eq_out",
   "eq_entropy_in", "eq_entropy_out", "eq_clustering_coefficient", "eq_page_rank",
    "eq_communities", "eq_motifs_3", "eq_motifs_4"),
  DD_weight = c(0.0735367966, 0.0739940162, 0.0714523761, 0.0708156931, 0.0601296752,
   0.0448072016, 0.0249793608, 0.0733125084, 0.0697029389, 0.0504358835, 0.0004016029,
   0.0563752664, 0.0561878218, 0.0540490099, 0.0504347104, 0.0558106667, 0.0568270319,
    0.0567474398),
  cause_orientation = "row",
  max_norm = FALSE,
  resolution = 100,
  resolution_min = 0.01,
  resolution_max = 0.99,
  reps = 3,
  processes = c("ER", "PA", "DM", "SW", "NM"),
  test = "empirical",
  best_fit_finder = "systematic",
  power_max = 5,
  connectance_max = 0.5,
  divergence_max = 0.5,
  mutation_max = 0.5,
  null_reps = 50,
  best_fit_kind = "avg",
  best_fit_sd = 0,
  ks_dither = 0,
  ks_alternative = "two.sided",
  cores = 1,
  size_different = FALSE,
  null_dist_trim = 1,
  verbose = FALSE
)
```

Arguments

`network`	The network to be classified.
`directed`	Whether the target network is directed. If missing this will be inferred by the symmetry of the input network.
`method`	This determines the method used to compare networks at the heart of the classification. Currently "DD" (Degree Distribution) and "align" (the align function which compares networks by the entropy of diffusion on them) are supported. Future versions will allow user-defined methods. Defaults to "DD".
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list"). Defaults to "matrix".
`mechanism_kind`	Either "canonical" or "grow" can be used to simulate networks. If "grow" is used, note that here it will only simulate pure mixtures made of a single mechanism. Defaults to "canonical".
`DD_kind`	= A vector of network properties to be used to compare networks. Defaults to "all", which is the average of the in- and out-degrees.
`DD_weight`	= Weights of each network property in DD_kind. Defaults to 1, which is equal weighting for each property.
`cause_orientation`	= The orientation of directed adjacency matrices. Defaults to "row".
`max_norm`	Binary variable indicating if each network property should be normalized so its max value (if a node-level property) is one. Defaults to FALSE.
`resolution`	Defaults to 100. The first step is to find the version of each process most similar to the target network. This parameter sets the number of parameter values to search across. Decrease to improve performance, but at the cost of accuracy.
`resolution_min`	Defaults to 0.01. The minimum parameter value to consider. Zero is not used because in many processes it results in degenerate systems (e.g. entirely unconnected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process.
`resolution_max`	Defaults to 0.99. The maximum parameter value to consider. One is not used because in many processes it results in degenerate systems (e.g. entirely connected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process.
`reps`	Defaults to 3. The number of networks to simulate for each parameter. More replicates increases accuracy by making the estimation of the parameter that produces networks most similar to the target network less idiosyncratic.

`processes`	Defaults to c("ER", "PA", "DD", "SW", "NM"). Vector of process abbreviations. Currently only the default five are supported. Future versions will accept user-defined network-generating functions and associated parameters. ER = Erdos-Renyi random. PA = Preferential Attachment. DD = Duplication and Divergence. SW = Small World. NM = Niche Model.
`test`	Defaults to "empirical". The test used to distinguish the null distribution of comparisons between the network being classified and the networks simulated according to a hypothesized mechanism(s), with a particular best-fitting parameter. "empirical" finds how many simulated networks were on average farther from each other than the network being classified is. "KS" uses a KS test. "WMWU" uses a Wilcoxon-Mann-Whitney-U test.
`best_fit_finder`	Defaults to "systematic". Determines how the best-fitting parameter of each mechanism specified in processes is found. "systematic" tries every parameter value from resolution_min to resolution_max with a step size of resolution_max - resolution_min / resolution. "optim_L-BFGS-B" uses the L-BFGS-B optimizer in the optimx package. "optim_GenSA" uses the GenSA optimizer in the GenSA package.
`power_max`	Defaults to 5. The maximum power of attachment in the Preferential Attachment process (PA).
`connectance_max`	= Defaults to 0.5. The maximum connectance parameter for the Niche Model.
`divergence_max`	= Defaults to 0.5. The maximum divergence parameter for the Duplication and Divergence/Mutation mechanisms.
`mutation_max`	= Defaults to 0.5. The maximum mutation parameter for the Duplication and Mutation mechanism.
`null_reps`	Defaults to 50. The number of best fit networks to simulate that will be used to create a null distribution of distances between networks within the given process, which will then be used to test if the target network appears unusually distant from them and therefore likely not governed by that process.
`best_fit_kind`	Defaults to "avg". If null_reps is more than 1, the fit of each parameter has to be an aggregate statistic of the fit of all the null_reps networks. Must be ‘avg‘, ‘median‘, ‘min‘, or ‘max‘.
`best_fit_sd`	Defaults to 0. Standard Deviation used to simulate networks with a similar but not identical best fit parameter. This is important because simulating networks with the identical parameter can artificially inflate the false negative rate by assuming the best fit parameter is the true parameter. For large resolution and reps values this will become true, but can be computationally intractable for realistically large systems.
`ks_dither`	Defaults to 0. The KS test cannot compute exact p-values when every pairwise network distance is not unique. Adding small amounts of noise makes each distance unique. We are not aware of a study on the impacts this has on accuracy so it is set to zero by default.
`ks_alternative`	Defaults to "two.sided". Governs the KS test. Assuming best_fit_sd is not too large, this can be set to "greater" because the target network cannot be more

	alike identically simulated networks than they are to each other. In practice we have found "greater" and "less" produce numerical errors. Only "two.sided", "less", and "greater" are supported through stats::ks.test().
`cores`	Defaults to 1. The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`size_different`	= If there is a difference in the size of the networks used in the null distribution. Defaults to FALSE.
`null_dist_trim`	= Number between zero and one that determines how much of each network comparison distribution (unknown network compared to simulated networks, simulated networks compared to each other) should be used. Prevents p-value convergence with large sample sizes. Defaults to 1, which means all comparisons are used (no trimming).
`verbose`	Defaults to FALSE. Whether to print all messages.

Details

Note: Currently each process is assumed to have a single governing parameter.

Value

A dataframe with 3 columns and as many rows as processes being tested (5 by default). The first column lists the processes. The second lists the p-value on the null hypothesis that the target network did come from that row's process. The third column gives the estimated parameter for that particular process.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

# Adjacency matrix
size <- 10
network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)

# Classify this network
# This can take several minutes to run
classify(network, processes = c("ER", "PA", "DM", "SW", "NM"))
```

classify_Systematic *Mechanistic Network Classification*

Description

Tests a network against hypothetical generating processes using a comparative network inference.

Usage

```
classify_Systematic(
  network,
  directed = FALSE,
  method = "DD",
  net_kind = "matrix",
  DD_kind = c("in", "out", "entropy_in", "entropy_out", "clustering_coefficient",
    "page_rank", "communities", "motifs_3", "motifs_4", "eq_in", "eq_out",
   "eq_entropy_in", "eq_entropy_out", "eq_clustering_coefficient", "eq_page_rank",
    "eq_communities", "eq_motifs_3", "eq_motifs_4"),
  DD_weight = c(0.0735367966, 0.0739940162, 0.0714523761, 0.0708156931, 0.0601296752,
   0.0448072016, 0.0249793608, 0.0733125084, 0.0697029389, 0.0504358835, 0.0004016029,
   0.0563752664, 0.0561878218, 0.0540490099, 0.0504347104, 0.0558106667, 0.0568270319,
    0.0567474398),
  cause_orientation = "row",
  max_norm = FALSE,
  resolution = 100,
  resolution_min = 0.01,
  resolution_max = 0.99,
  reps = 3,
  processes = c("ER", "PA", "DM", "SW", "NM"),
  power_max = 5,
  connectance_max = 0.5,
  divergence_max = 0.5,
  mutation_max = 0.5,
  null_reps = 50,
  best_fit_kind = "avg",
  best_fit_sd = 0.01,
  ks_dither = 0,
  ks_alternative = "two.sided",
  cores = 1,
  size_different = FALSE,
  null_dist_trim = 1,
  verbose = TRUE
)
```

Arguments

`network`	The network to be classified.

`directed`	Defaults to TRUE. Whether the target network is directed.
`method`	This determines the method used to compare networks at the heart of the classification. Currently "DD" (Degree Distribution) and "align" (the align function which compares networks by the entropy of diffusion on them) are supported. Future versions will allow user-defined methods. Defaults to "DD".
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list"). Defaults to "matrix".
`DD_kind`	= A vector of network properties to be used to compare networks. Defaults to "all", which is the average of the in- and out-degrees.
`DD_weight`	= Weights of each network property in DD_kind. Defaults to 1, which is equal weighting for each property.
`cause_orientation`	= The orientation of directed adjacency matrices. Defaults to "row".
`max_norm`	Binary variable indicating if each network property should be normalized so its max value (if a node-level property) is one. Defaults to FALSE.
`resolution`	Defaults to 100. The first step is to find the version of each process most similar to the target network. This parameter sets the number of parameter values to search across. Decrease to improve performance, but at the cost of accuracy.
`resolution_min`	Defaults to 0.01. The minimum parameter value to consider. Zero is not used because in many processes it results in degenerate systems (e.g. entirely unconnected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process.
`resolution_max`	Defaults to 0.99. The maximum parameter value to consider. One is not used because in many processes it results in degenerate systems (e.g. entirely connected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process.
`reps`	Defaults to 3. The number of networks to simulate for each parameter. More replicates increases accuracy by making the estimation of the parameter that produces networks most similar to the target network less idiosyncratic.
`processes`	Defaults to c("ER", "PA", "DD", "SW", "NM"). Vector of process abbreviations. Currently only the default five are supported. Future versions will accept user-defined network-generating functions and associated parameters. ER = Erdos-Renyi random. PA = Preferential Attachment. DD = Duplication and Divergence. SW = Small World. NM = Niche Model.
`power_max`	Defaults to 5. The maximum power of attachment in the Preferential Attachment process (PA).
`connectance_max`	= Defaults to 0.5. The maximum connectance parameter for the Niche Model.
`divergence_max`	= Defaults to 0.5. The maximum divergence parameter for the Duplication and Divergence/Mutation mechanisms.
`mutation_max`	= Defaults to 0.5. The maximum mutation parameter for the Duplication and Mutation mechanism.
`null_reps`	Defaults to 50. The number of best fit networks to simulate that will be used to create a null distribution of distances between networks within the given process,

	which will then be used to test if the target network appears unusually distant from them and therefore likely not governed by that process.
`best_fit_kind`	Defaults to "avg". If null_reps is more than 1, the fit of each parameter has to be an aggregate statistic of the fit of all the null_reps networks. Must be ‘avg‘, ‘median‘, ‘min‘, or ‘max‘.
`best_fit_sd`	Defaults to 0.01. Standard Deviation used to simulate networks with a similar but not identical best fit parameter. This is important because simulating networks with the identical parameter artificially inflates the false negative rate by assuming the best fit parameter is the true parameter. For large resolution and reps values this will become true, but also computationally intractable for realistically large systems.
`ks_dither`	Defaults to 0. The KS test cannot compute exact p-values when every pairwise network distance is not unique. Adding small amounts of noise makes each distance unique. We are not aware of a study on the impacts this has on accuracy so it is set to zero by default.
`ks_alternative`	Defaults to "two.sided". Governs the KS test. Assuming best_fit_sd is not too large, this can be set to "greater" because the target network cannot be more alike identically simulated networks than they are to each other. In practice we have found "greater" and "less" produce numerical errors. Only "two.sided", "less", and "greater" are supported through stats::ks.test().
`cores`	Defaults to 1. The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`size_different`	= If there is a difference in the size of the networks used in the null distribution. Defaults to FALSE.
`null_dist_trim`	= Number between zero and one that determines how much of each network comparison distribution (unknown network compared to simulated networks, simulated networks compared to each other) should be used. Prevents p-value convergence with large sample sizes. Defaults to 1, which means all comparisons are used (no trimming).
`verbose`	Defaults to TRUE. Whether to print all messages.

Details

Note: Currently each process is assumed to have a single governing parameter.

Value

A dataframe with 3 columns and as many rows as processes being tested (5 by default). The first column lists the processes. The second lists the p-value on the null hypothesis that the target network did come from that row’s process. The third column gives the estimated parameter for that particular process.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

# Adjacency matrix
size <- 10
network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)

# Classify this network
# This can take several minutes to run
classify(network, processes = c("ER", "PA", "DM", "SW", "NM"))
```

---

compare *Compare Networks Many-to-Many*

Description

Compares one network to a list of many networks.

Usage

```
compare(
  networks,
  net_kind = "matrix",
  method = "DD",
  cause_orientation = "row",
  DD_kind = "all",
  DD_weight = 1,
  max_norm = FALSE,
  size_different = FALSE,
  cores = 1,
  diffusion_sampling = 2,
  diffusion_limit = 10,
  verbose = FALSE
)
```

Arguments

`networks`	The networks being compared to the target network
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list"). Defaults to "matrix".
`method`	This determines the method used to compare networks at the heart of the classification. Currently "DD" (Degree Distribution) and "align" (the align function which compares networks by the entropy of diffusion on them) are supported. Future versions will allow user-defined methods. Defaults to "DD".

`cause_orientation`	= The orientation of directed adjacency matrices. Defaults to "row".
`DD_kind`	= A vector of network properties to be used to compare networks. Defaults to "all", which is the average of the in- and out-degrees.
`DD_weight`	= Weights of each network property in DD_kind. Defaults to 1, which is equal weighting for each property.
`max_norm`	Binary variable indicating if each network property should be normalized so its max value (if a node-level property) is one. Defaults to FALSE.
`size_different`	Defaults to FALSE. If TRUE, will ensure the node-level properties being compared are vectors of the same length, which is accomplished using splines.
`cores`	Defaults to 1. The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`diffusion_sampling`	Base of the power to use to nonlinearly sample the diffusion kernels if method = "align". Defaults to 2.
`diffusion_limit`	Number of markov steps in the diffusion kernels if method = "align". Defaults to 10.
`verbose`	Defaults to TRUE. Whether to print all messages.

Details

Note: Currently each process is assumed to have a single governing parameter.

Value

A square matrix with dimensions equal to the number of networks being compared, where the ij element is the comparison of networks i and j.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

# Adjacency matrix
size <- 10
comparisons <- 50
networks <- list()
for (net in 1:comparisons) {
     networks[[net]] = matrix(
         sample(
             c(0,1),
             size = size^2,
             replace = TRUE),
         nrow = size,
         ncol = size)
}
compare(networks = networks)
```

`compare_Target` *Compare Networks One-to-Many*

Description

Compares one network to a list of many networks.

Usage

```
compare_Target(
  target,
  networks,
  net_size,
  net_kind = "matrix",
  method = "DD",
  cause_orientation = "row",
  DD_kind = "all",
  DD_weight = 1,
  max_norm = FALSE,
  cores = 1,
  verbose = FALSE
)
```

Arguments

`target`	The network be compared.
`networks`	The networks being compared to the target network
`net_size`	Size
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list"). Defaults to "matrix".
`method`	This determines the method used to compare networks at the heart of the classification. Currently "DD" (Degree Distribution) and "align" (the align function which compares networks by the entropy of diffusion on them) are supported. Future versions will allow user-defined methods. Defaults to "DD".
`cause_orientation`	= The orientation of directed adjacency matrices. Defaults to "row".
`DD_kind`	= A vector of network properties to be used to compare networks. Defaults to "all", which is the average of the in- and out-degrees.

`DD_weight`	= Weights of each network property in DD_kind. Defaults to 1, which is equal weighting for each property.
`max_norm`	Binary variable indicating if each network property should be normalized so its max value (if a node-level property) is one. Defaults to FALSE.
`cores`	Defaults to 1. The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`verbose`	Defaults to TRUE. Whether to print all messages.

Details

Note: Currently each process is assumed to have a single governing parameter.

Value

A pseudo-distance vector where the i-element is the comparison between the target network and the ith network being compared to.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

# Adjacency matrix
size <- 10
comparisons <- 50
network_target <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
network_others <- list()
for (net in 1:comparisons) {
     network_others[[net]] = matrix(
         sample(
             c(0,1),
             size = size^2,
             replace = TRUE),
         nrow = size,
         ncol = size)
}
compare_Target(target = network_target, networks = network_others, net_size = size, method = "DD")
```

gini — *Gini coefficient*

Description

Takes a matrix and returns the Gini coefficient of each column.

Usage

```
gini(input, byrow = FALSE)
```

Arguments

`input`	A matrix where the Gini coefficient will be calculated on each column. Note that vector data must be converted to a single-column matrix.
`byrow`	Defaults to FALSE. Set to TRUE to calculate the Gini coefficient of each row.

Value

A vector of the Gini coefficients of each column.

References

Gini, C. (1912). Variabilita e mutabilita. Reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi.

Examples

```
# Vectors are not supported. First convert to a single-column matrix.
sample_data <- runif(20, 0, 1)
gini(matrix(sample_data, ncol = 1))

# Multiple Gini coefficients can be calculated simultaneously
gini(matrix(sample_data, ncol = 2))
```

grow_DD — *Grow a Duplication and Divergence Network*

Description

Grows an already existing network by adding a node according to the Duplication and Divergence mechanism. Nodes can only attach to previously grown nodes.

Usage

```
grow_DD(
  matrix,
  x,
  divergence,
  link = 0,
  connected = FALSE,
  retcon = FALSE,
  directed = TRUE
)
```

Arguments

`matrix`	Existing network to experience growth.
`x`	The ID of the node to be grown.
`divergence`	Probability that the new node loses edges associated with the node it duplicates. Needs to be between zero and one.
`link`	Probability that the new node attaches to the node it duplicates. Defaults to 0.
`connected`	Binary argument determining if the newly grown node has to be connected to the existing network. Defaults to FALSE, to prevent rare computational slow-downs when it is unlikely to create a connected network. Defaults to FALSE.
`retcon`	Binary variable determining if already existing nodes can attach to new nodes. Defaults to FALSE.
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix. Defaults to TRUE.

Details

Different from Duplication & Mutation models in that edges can only be lost.

Value

An adjacency matrix.

References

Ispolatov, I., Krapivsky, P. L., & Yuryev, A. (2005). Duplication-divergence model of protein interaction network. Physical review E, 71(6), 061911.

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
new_network_prep[1:size, 1:size] = existing_network
```

```
new_network <- grow_DD(matrix = new_network_prep, x = size + 1, divergence = 0.5)
```

---

grow_DM *Grow a Duplication and Mutation Network*

Description

Grows an already existing network by adding a node according to the Duplication and Mutation mechanism. Nodes can only attach to previously grown nodes.

Usage

```
grow_DM(
  matrix,
  x,
  divergence,
  mutation = 0,
  link = 0,
  connected = FALSE,
  retcon = FALSE,
  directed = TRUE
)
```

Arguments

matrix	Existing network to experience growth.
x	The ID of the node to be grown.
divergence	Probability that the new node loses edges associated with the node it duplicates. Needs to be between zero and one.
mutation	Probability that the new node gains edges not associated with the node it duplicates. Needs to be between zero and one.
link	Probability that the new node attaches to the node it duplicates. Defaults to 0.
connected	Binary argument determining if the newly grown node has to be connected to the existing network. Defaults to FALSE, to prevent rare computational slow-downs when it is unlikely to create a connected network. Defaults to FALSE.
retcon	Binary variable determining if already existing nodes can attach to new nodes. Defaults to FALSE.
directed	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix. Defaults to TRUE.

Details

Different from Duplication & Mutation models in that edges can only be lost.

Value

An adjacency matrix.

References

Ispolatov, I., Krapivsky, P. L., & Yuryev, A. (2005). Duplication-divergence model of protein interaction network. Physical review E, 71(6), 061911.

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
new_network_prep[1:size, 1:size] = existing_network
new_network <- grow_DM(matrix = new_network_prep, x = size + 1, divergence = 0.5)
```

---

grow_ER *Grow an Erdos-Renyi Random Network*

Description

Grows an already existing network by adding a node according to the Erdos-Renyi random mechanism. Nodes can only attach to previously grown nodes.

Usage

```
grow_ER(matrix, x, p, retcon = FALSE, directed = TRUE)
```

Arguments

`matrix`	Existing network to experience growth.
`x`	The ID of the node to be grown.
`p`	Probability possible edges exist. Needs to be between zero and one.
`retcon`	Binary variable determining if already existing nodes can attach to new nodes. Defaults to FALSE.
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix. Defaults to TRUE.

Details

Different from Duplication & Mutation models in that edges can only be lost.

Value

An adjacency matrix.

References

Erdos, P. and Renyi, A., On random graphs, Publicationes Mathematicae 6, 290–297 (1959).

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
new_network_prep[1:size, 1:size] = existing_network
new_network <- grow_ER(matrix = new_network_prep, x = size + 1, p = 0.5)
```

`grow_NM` *Grow a Niche Model Network*

Description

Grows an already existing network by adding a node according to the Niche Model mechanism. Nodes can only attach to previously grown nodes.

Usage

```
grow_NM(matrix, x, niches, connectance = 0.2, directed = TRUE, retcon = FALSE)
```

Arguments

`matrix`	Existing network to experience growth.
`x`	The ID of the node to be grown.
`niches`	Vector of length x, with values between zero and one corresponding to each node's niche.
`connectance`	Niche Model parameter specifying the expected connectivity of the network, which determines for a given node the niche space window within which it attaches to every other node. Defaults to 0.2.
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix. Defaults to TRUE.
`retcon`	Binary variable determining if already existing nodes can attach to new nodes. Defaults to FALSE.

Details

Stirs a node in a Niche Model network.

Value

An adjacency matrix.

References

Williams, R. J., & Martinez, N. D. (2000). Simple rules yield complex food webs. Nature, 404(6774), 180-183.

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
new_network_prep[1:size, 1:size] = existing_network
new_network <- grow_NM(matrix = new_network_prep, x = size + 1, niches = stats::runif(size))
```

---

grow_PA *Grow a Preferential Attachment Network*

Description

Grows an already existing network by adding a node according to the Preferential Attachment mechanism. Nodes can only attach to previously grown nodes.

Usage

```
grow_PA(
  matrix,
  x,
  power,
  sum_v_max = "sum",
  nascent_help = TRUE,
  retcon = FALSE,
  directed = TRUE
)
```

Arguments

`matrix`	Existing network to experience growth.
`x`	The ID of the node to be grown.
`power`	Power of attachment, which determines how much new nodes prefer to attach to nodes that have many edges compared to few. Needs to be positive.
`sum_v_max`	Degree distributions must be normalized, either by their "max" or "sum". Defaults to "max".
`nascent_help`	Should a single edge be added to the degree distribution of all nodes so that nodes with a zero in-degree can still have a chance of being attached to by new nodes. Defaults to TRUE.
`retcon`	Binary variable determining if already existing nodes can attach to new nodes. Defaults to FALSE.
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix. Defaults to TRUE.

Details

Adds a node in a network according to the Preferential Attachment mechanism.

Value

An adjacency matrix.

References

Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. science, 286(5439), 509-512.

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
new_network_prep[1:size, 1:size] = existing_network
new_network <- grow_PA(matrix = new_network_prep, x = size + 1, power = 2.15)
```

grow_SW *Grow a Small-World Network*

Description

Grows an already existing network by adding a node according to the Small-World mechanism. Nodes can only attach to previously grown nodes.

Usage

```
grow_SW(matrix, x, rewire, connected = FALSE, retcon = FALSE, directed = TRUE)
```

Arguments

`matrix`	Existing network to experience growth.
`x`	The ID of the node to be grown.
`rewire`	Small-World parameter specifying the probability each edge is randomly rewired, allowing for the possiblity of bridges between connected communities.
`connected`	Binary argument determining if the newly grown node has to be connected to the existing network. Defaults to FALSE, to prevent rare computational slow-downs when it is unlikely to create a connected network. Defaults to False.
`retcon`	Binary variable determining if already existing nodes can attach to new nodes. Defaults to FALSE.
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix. Defaults to TRUE.

Details

Grows a node in a network according to the Small-World mechanism.

Value

An adjacency matrix.

References

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of ‘small-world’networks. nature, 393(6684), 440-442.

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
```

```
new_network_prep[1:size, 1:size] = existing_network
new_network <- grow_SW(matrix = new_network_prep, x = size + 1, rewire = 0.213)
```

`ics` *Induced Conserved Structure (ICS)*

Description

Calculates the Induced Conserved Structure proposed by Patro and Kingsford (2012) of an alignment between two networks.

Usage

```
ics(network_1_input, network_2_input, alignment, flip = FALSE)
```

Arguments

`network_1_input`	The first network being aligned, which must be in matrix form. If the two networks are of different sizes, it will be easier to interpret the output if this is the smaller one.
`network_2_input`	The second network, which also must be a matrix.
`alignment`	A matrix, such as is output by the function NetCom, where the first two columns contain corresponding node IDs for the two networks that were aligned.
`flip`	Defaults to FALSE. Set to TRUE if the first network is larger than the second. This is necessary because ICS is not a symmetric measure of alignment quality.

Value

A number ranging between 0 and 1. If the Induced Conserved Structure is 1, the two networks are isomorphic (identical) under the given alignment.

References

Patro, R., & Kingsford, C. (2012). Global network alignment using multiscale spectral signatures. Bioinformatics, 28(23), 3105-3114.

Examples

```
# Note that ICS is only defined on unweighted networks.
net_one <- round(matrix(runif(25,0,1), nrow=5, ncol=5))
net_two <- round(matrix(runif(25,0,1), nrow=5, ncol=5))

ics(net_two, net_two, align(net_one, net_two)$alignment)
```

make_DD *Makes a Duplication and Divergence Network*

Description

Makes a network according to the Duplication and Divergence mechanism.

Usage

```
make_DD(size, net_kind, divergence, directed = TRUE)
```

Arguments

`size`	Number of nodes in the network.
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list").
`divergence`	Probability that the new node loses edges associated with the node it duplicates. Needs to be between zero and one.
`directed`	Whether the target network is directed. Defaults to TRUE.

Details

Different from Duplication & Mutation models in that edges can only be lost.

Value

An adjacency matrix.

References

Ispolatov, I., Krapivsky, P. L., & Yuryev, A. (2005). Duplication-divergence model of protein interaction network. Physical review E, 71(6), 061911.

Examples

```
# Import netcom
library(netcom)

# Network size (number of nodes)
size <- 10

# Divergence parameter
divergence <- 0.237

# Make network according to the Duplication & Divergence mechanism
make_DD(size = size, net_kind = "matrix", divergence = divergence)
```

make_DM *Make a Duplication and Mutation Network*

Description

Make an already existing network according to the Duplication and Mutation mechanism.

Usage

```
make_DM(size, net_kind, divergence, mutation, directed = FALSE)
```

Arguments

`size`	Number of nodes in the network.
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list").
`divergence`	Probability that the new node loses edges associated with the node it duplicates. Needs to be between zero and one.
`mutation`	Probability that the new node gains edges not associated with the node it duplicates. Needs to be between zero and one.
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix. Defaults to TRUE.

Details

Different from Duplication & Mutation models in that edges can only be lost.

Value

An adjacency matrix.

References

Ispolatov, I., Krapivsky, P. L., & Yuryev, A. (2005). Duplication-divergence model of protein interaction network. Physical review E, 71(6), 061911.

Examples

```
# Import netcom
library(netcom)

# Network size (number of nodes)
size <- 10

# Divergence parameter
divergence <- 0.237

# Mutation parameter
```

```
mutation <- 0.1

# Make network according to the Duplication & Mutation mechanism
make_DM(size = size, net_kind = "matrix", divergence = divergence, mutation = mutation)
```

make_Mixture *Make a Mixture Mechanism Network*

Description

Creates a network by iteratively adding or rewiring nodes, each capable of attaching to existing nodes according to a user-specified mechanism.

Usage

```
make_Mixture(
  mechanism,
  directed,
  parameter,
  kind,
  size,
  niches,
  retcon = FALSE,
  link_DD = 0,
  link_DM = 0,
  force_connected = FALSE
)
```

Arguments

`mechanism`	A vector of mechanism names corresponding to the mechanisms each node acts in accordance with. Note that the first two mechanisms are irrelevant because the first two nodes default to connecting to each other. Currently supported mechanisms: "ER" (Erdos-Renyi random), "PA", (Preferential Attachment), "DD", (Duplication and Divergence), "DM" (Duplication and Mutation), "SW", (Small-World), and "NM" (Niche Model).
`directed`	A binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix. Either a single value or a vector of values the same length as the mechanism input vector.
`parameter`	Parameter of each node's mechanism. Either a single value or a vector of values the same length as the mechanism input vector.
`kind`	Either 'grow' or 'rewire', and determines if the nodes specified in the mechanism input vector are to be rewired or grown. Either a single value or a vector of values the same length as the mechanism input vector. The number of 'grow' nodes, excluding the first two which are always a pair of bidirectionally connected nodes, is the size of the final network.

`size`	Typically not specified. The size of the network depends on how many ‘grow‘ events are part of the ‘kind‘ input sequence. This should only be used when all four components of the network evolution (‘mechanism‘, ‘kind‘, ‘parameter‘, and ‘directed‘) are single name inputs instead of vectors.
`niches`	Used by the Niche Model to determine which nodes interact. Needs to be a vector of the same length as the number of nodes, and range between zero and one.
`retcon`	Binary variable determining if already existing nodes can attach to new nodes. Defaults to FALSE.
`link_DD`	Defaults to 0. A second parameter in the DD (Duplication & Divergence). Currently only one parameter per mechanism can be specified.
`link_DM`	Defaults to 0. A second parameter in the DM (Duplication & Mutation). Currently only one parameter per mechanism can be specified.
`force_connected`	Defaults to FALSE. Determines if nodes can be added to the growing network that are disconnected. If TRUE, this is prevented by re-determining the offending node’s edges until the network is connected.

Details

This function grows, one node at a time, a mixture mechanism network. As each node is added to the growing network it can attach to existing nodes by its own node-specific mechanism. A sequence of mechanism names must be provided. Note: Currently each mechanism is assumed to have a single governing parameter.

Value

An unweighted mixture mechanism adjacency matrix.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

# Start by creating a sequence of network evolutions.
# There are four components to this sequence that can each be defined for every step
# in the network's evolution. Or, you can also specify a component once which will
# be used for every step in the newtwork's evolution.

mechanism <- c(
    rep("ER", 7),
    rep("PA", 2),
    rep("ER", 3)
)
```

```
kind <- c(
    rep("grow", 7),
    rep("rewire", 2),
    rep("grow", 3)
)

parameter <- c(
    rep(0.3, 7),
    rep(2, 2),
    rep(0.3, 3)
)
directed <- c(
    rep(TRUE, 7),
    rep(FALSE, 2),
    rep(TRUE, 3)
)

# Simulate a network according to the rules of this system evolution.
network <- make_Mixture(
     mechanism = mechanism,
     kind = kind,
     parameter = parameter,
     directed = directed
)
```

make_NM *Make a Niche Model network*

Description

Creates a single network according to the Niche Model. Can be directed or undirected, but is always unweighted.

Usage

```
make_NM(
  size,
  niches,
  net_kind = "matrix",
  connectance = 0.1,
  directed = TRUE,
  grow = FALSE
)
```

Arguments

`size`	The number of nodes in the network. Must be a positive integer.

`niches`	A vector of numbers specifying the niche of each member of the system (node). Each niche value must be element of [0,1].
`net_kind`	The format of the network. Currently must be either 'matrix' or 'list'.
`connectance`	Defaults to 0.5. The ratio of actual interactions to possible interactions. Effects the beta distributed width of niche values each member of the system (node) interacts with.
`directed`	If FALSE all interactions will be made symmetric. Note that the process of creating interactions is unaffected by this choice. Defaults to TRUE.
`grow`	Binary argument that determines if the network should be made in a growing fashion, where nodes' edges are added in order of their niches and can only attach to previously considered nodes. Defaults to FALSE.

Value

An interaction matrix format of a Niche Model network.

References

Williams, R. J., & Martinez, N. D. (2000). Simple rules yield complex food webs. Nature, 404(6774), 180-183.

Examples

```
# Import netcom
library(netcom)

# Network size (number of nodes)
size <- 10

# Create niche values for each member of the system (node)
niches <- stats::runif(n = size)

# Make network according to the Niche Model
make_NM(size = size, niches = niches)
```

`make_Null` *Mechanism Null Distributions*

Description

Creates a null distribution for a mechanism and parameter combination.

Usage

```
make_Null(
  input_network,
  net_kind,
  mechanism_kind,
  process,
  parameter,
  net_size,
  iters,
  method,
  neighborhood,
  DD_kind,
  DD_weight,
  directed,
  resolution_min = 0.01,
  resolution_max = 0.99,
  power_max = 5,
  connectance_max = 0.5,
  divergence_max = 0.5,
  best_fit_sd = 0,
  cores = 1,
  size_different = FALSE,
  cause_orientation = "row",
  max_norm = FALSE,
  verbose = FALSE
)
```

Arguments

`input_network`	The network for which to create a null distribution.
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list"). Defaults to "matrix".
`mechanism_kind`	Either "canonical" or "grow" can be used to simulate networks. If "grow" is used, note that here it will only simulate pure mixtures made of a single mechanism.
`process`	Name of mechanism. Currently only "ER", "PA", "DD", "DM" "SW", and "NM" are supported. Future versions will accept user-defined network-generating functions and associated parameters. ER = Erdos-Renyi random. PA = Preferential Attachment. DD = Duplication and Divergence. DM = Duplication and Mutation. SW = Small World. NM = Niche Model.
`parameter`	Parameter in the governing mechanism.
`net_size`	Number of nodes in the network.
`iters`	Number of replicates in the null distribution. Note that length(null_dist) = ((iters^2)-iters)/2.
`method`	This determines the method used to compare networks at the heart of the classification. Currently "DD" (Degree Distribution) and "align" (the align function

	which compares networks by the entropy of diffusion on them) are supported. Future versions will allow user-defined methods.
`neighborhood`	The range of nodes that form connected communities. Note: This implementation results in overlap of communities.
`DD_kind`	= A vector of network properties to be used to compare networks.
`DD_weight`	= A vector of weights for the relative importance of the network properties in DD_kind being used to compare networks. Should be the same length as DD_kind.
`directed`	Whether the target network is directed.
`resolution_min`	The minimum parameter value to consider. Zero is not used because in many processes it results in degenerate systems (e.g. entirely unconnected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.01.
`resolution_max`	The maximum parameter value to consider. One is not used because in many processes it results in degenerate systems (e.g. entirely connected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.99.
`power_max`	Defaults to 5. The maximum power of attachment in the Preferential Attachment process (PA).
`connectance_max`	Defaults to 0.5. The maximum connectance parameter for the Niche Model.
`divergence_max`	Defaults to 0.5. The maximum divergence parameter for the Duplication and Divergence/Mutation mechanisms.
`best_fit_sd`	Defaults to 0.01. Standard Deviation used to simulate networks with a similar but not identical best fit parameter. This is important because simulating networks with the identical parameter artificially inflates the false negative rate by assuming the best fit parameter is the true parameter. For large resolution and reps values this will become true, but also computationally intractable for realistically large systems.
`cores`	Defaults to 1. The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`size_different`	If there is a difference in the size of the networks used in the null distribution. Defaults to FALSE.
`cause_orientation`	The orientation of directed adjacency matrices. Defaults to "row".
`max_norm`	Binary variable indicating if each network property should be normalized so its max value (if a node-level property) is one. Defaults to FALSE.
`verbose`	Defaults to FALSE. Whether to print all messages.

Details

Produces ground-truthing network data.

Value

A list. The first element contains the networks. The second contains their corresponding parameters.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

make_Systematic(net_size = 10)
```

---

make_Null_canonical *Mechanism Null Distributions*

Description

Creates a null distribution for a mechanism and parameter combination.

Usage

```
make_Null_canonical(
  input_network,
  net_kind,
  process,
  parameter,
  net_size,
  iters,
  method,
  neighborhood,
  DD_kind,
  DD_weight,
  directed,
  resolution_min = 0.01,
  resolution_max = 0.99,
  power_max = 5,
  connectance_max = 0.5,
  divergence_max = 0.5,
  best_fit_sd = 0,
  cores = 1,
  size_different = FALSE,
  cause_orientation = "row",
  max_norm = FALSE,
  verbose = FALSE
)
```

Arguments

`input_network`	The network for which to create a null distribution.
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list"). Defaults to "matrix".
`process`	Name of mechanism. Currently only "ER", "PA", "DD", "DM" "SW", and "NM" are supported. Future versions will accept user-defined network-generating functions and associated parameters. ER = Erdos-Renyi random. PA = Preferential Attachment. DD = Duplication and Divergence. DM = Duplication and Mutation. SW = Small World. NM = Niche Model.
`parameter`	Parameter in the governing mechanism.
`net_size`	Number of nodes in the network.
`iters`	Number of replicates in the null distribution. Note that length(null_dist) = ((iters^2)-iters)/2.
`method`	This determines the method used to compare networks at the heart of the classification. Currently "DD" (Degree Distribution) and "align" (the align function which compares networks by the entropy of diffusion on them) are supported. Future versions will allow user-defined methods.
`neighborhood`	The range of nodes that form connected communities. Note: This implementation results in overlap of communities.
`DD_kind`	A vector of network properties to be used to compare networks.
`DD_weight`	A vector of weights for the relative importance of the network properties in DD_kind being used to compare networks. Should be the same length as DD_kind.
`directed`	Whether the target network is directed.
`resolution_min`	The minimum parameter value to consider. Zero is not used because in many processes it results in degenerate systems (e.g. entirely unconnected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.01.
`resolution_max`	The maximum parameter value to consider. One is not used because in many processes it results in degenerate systems (e.g. entirely connected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.99.
`power_max`	Defaults to 5. The maximum power of attachment in the Preferential Attachment process (PA).
`connectance_max`	Defaults to 0.5. The maximum connectance parameter for the Niche Model.
`divergence_max`	Defaults to 0.5. The maximum divergence parameter for the Duplication and Divergence/Mutation mechanisms.
`best_fit_sd`	Defaults to 0.01. Standard Deviation used to simulate networks with a similar but not identical best fit parameter. This is important because simulating networks with the identical parameter artificially inflates the false negative rate by assuming the best fit parameter is the true parameter. For large resolution and reps values this will become true, but also computationally intractable for realistically large systems.

`cores`	Defaults to 1. The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`size_different`	If there is a difference in the size of the networks used in the null distribution. Defaults to FALSE.
`cause_orientation`	The orientation of directed adjacency matrices. Defaults to "row".
`max_norm`	Binary variable indicating if each network property should be normalized so its max value (if a node-level property) is one. Defaults to FALSE.
`verbose`	Defaults to FALSE. Whether to print all messages.

Details

Produces ground-truthing network data.

Value

A list. The first element contains the networks. The second contains their corresponding parameters.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

make_Systematic(net_size = 10)
```

`make_Null_mixture` *Mechanism Null Distributions*

Description

Creates a null distribution for a mechanism and parameter combination.

Usage

```
make_Null_mixture(
  input_network,
  net_kind,
  process,
  parameter,
  net_size,
  iters,
  method,
  neighborhood,
  DD_kind,
  DD_weight,
  directed,
  resolution_min = 0.01,
  resolution_max = 0.99,
  power_max = 5,
  connectance_max = 0.5,
  divergence_max = 0.5,
  best_fit_sd = 0,
  cores = 1,
  size_different = FALSE,
  cause_orientation = "row",
  max_norm = FALSE,
  verbose = FALSE
)
```

Arguments

`input_network`	The network for which to create a null distribution.
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list"). Defaults to "matrix".
`process`	Name of mechanism. Currently only "ER", "PA", "DD", "DM" "SW", and "NM" are supported. Future versions will accept user-defined network-generating functions and associated parameters. ER = Erdos-Renyi random. PA = Preferential Attachment. DD = Duplication and Divergence. DM = Duplication and Mutation. SW = Small World. NM = Niche Model.
`parameter`	Parameter in the governing mechanism.
`net_size`	Number of nodes in the network.
`iters`	Number of replicates in the null distribution. Note that length(null_dist) = ((iters^2)-iters)/2.
`method`	This determines the method used to compare networks at the heart of the classification. Currently "DD" (Degree Distribution) and "align" (the align function which compares networks by the entropy of diffusion on them) are supported. Future versions will allow user-defined methods.
`neighborhood`	The range of nodes that form connected communities. Note: This implementation results in overlap of communities.
`DD_kind`	A vector of network properties to be used to compare networks.
`DD_weight`	A vector of weights for the relative importance of the network properties in DD_kind being used to compare networks. Should be the same length as DD_kind.
`directed`	Whether the target network is directed.
`resolution_min`	The minimum parameter value to consider. Zero is not used because in many processes it results in degenerate systems (e.g. entirely unconnected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.01.

`resolution_max`	The maximum parameter value to consider. One is not used because in many processes it results in degenerate systems (e.g. entirely connected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.99.
`power_max`	Defaults to 5. The maximum power of attachment in the Preferential Attachment process (PA).
`connectance_max`	Defaults to 0.5. The maximum connectance parameter for the Niche Model.
`divergence_max`	Defaults to 0.5. The maximum divergence parameter for the Duplication and Divergence/Mutation mechanisms.
`best_fit_sd`	Defaults to 0.01. Standard Deviation used to simulate networks with a similar but not identical best fit parameter. This is important because simulating networks with the identical parameter artificially inflates the false negative rate by assuming the best fit parameter is the true parameter. For large resolution and reps values this will become true, but also computationally intractable for realistically large systems.
`cores`	Defaults to 1. The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`size_different`	If there is a difference in the size of the networks used in the null distribution. Defaults to FALSE.
`cause_orientation`	The orientation of directed adjacency matrices. Defaults to "row".
`max_norm`	Binary variable indicating if each network property should be normalized so its max value (if a node-level property) is one. Defaults to FALSE.
`verbose`	Defaults to FALSE. Whether to print all messages.

Details

Produces ground-truthing network data.

Value

A list. The first element contains the networks. The second contains their corresponding parameters.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

make_Systematic(net_size = 10)
```

make_SW *Makes a Small-World Network*

Description

Make an already existing network according to the Small-World mechanism.

Usage

```
make_SW(size, rewire, neighborhood, net_kind = "matrix", directed = FALSE)
```

Arguments

`size`	The number of nodes in the network. Must be a positive integer.
`rewire`	Small-World parameter specifying the probability each edge is randomly rewired, allowing for the possiblity of bridges between connected communities.
`neighborhood`	The range of nodes that form connected communities. Note: This implementation results in overlap of communities.
`net_kind`	The format of the network. Currently must be either 'matrix' or 'list'.x
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix. Defaults to TRUE.

Details

Rewires a node in a network according to the Small-World mechanism.

Value

An adjacency matrix.

References

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. nature, 393(6684), 440-442.

Examples

```
# Import netcom
library(netcom)

# Network size (number of nodes)
size <- 10

# Rewiring parameter
rewire <- 0.2

# Make network according to the Small-World mechanism
make_SW(size = size, net_kind = "matrix", rewire = rewire)
```

make_Systematic *Systematically Make Networks*

Description

Creates a list of networks that systematically spans mechanisms and their respective parameters.

Usage

```
make_Systematic(
  net_size,
  neighborhood,
  directed = TRUE,
  net_kind = "matrix",
  mechanism_kind = "canonical",
  resolution = 100,
  resolution_min = 0.01,
  resolution_max = 0.99,
  reps = 3,
  processes = c("ER", "PA", "DM", "SW", "NM"),
  power_max = 5,
  connectance_max = 0.5,
  divergence_max = 0.5,
  mutation_max = 0.5,
  canonical = FALSE,
  cores = 1,
  verbose = TRUE
)
```

Arguments

`net_size`	Number of nodes in the network.
`neighborhood`	The range of nodes that form connected communities. Note: This implementation results in overlap of communities.
`directed`	Whether the target network is directed. Defaults to TRUE.
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list"). Defaults to "matrix".
`mechanism_kind`	Either "canonical" or "grow" can be used to simulate networks. If "grow" is used, note that here it will only simulate pure mixtures made of a single mechanism. Defaults to "canonical".
`resolution`	The first step is to find the version of each process most similar to the target network. This parameter sets the number of parameter values to search across. Decrease to improve performance, but at the cost of accuracy. Defaults to 100.

`resolution_min`	= The minimum parameter value to consider. Zero is not used because in many processes it results in degenerate systems (e.g. entirely unconnected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.01.
`resolution_max`	The maximum parameter value to consider. One is not used because in many processes it results in degenerate systems (e.g. entirely connected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.99.
`reps`	Defaults to 3. The number of networks to simulate for each parameter. More replicates increases accuracy by making the estimation of the parameter that produces networks most similar to the target network less idiosyncratic.
`processes`	Defaults to c("ER", "PA", "DD", "SW", "NM"). Vector of process abbreviations. Currently only the default five are supported. Future versions will accept user-defined network-generating functions and associated parameters. ER = Erdos-Renyi random. PA = Preferential Attachment. DD = Duplication and Divergence. SW = Small World. NM = Niche Model.
`power_max`	Defaults to 5. The maximum power of attachment in the Preferential Attachment process (PA).
`connectance_max`	Defaults to 0.5. The maximum connectance parameter for the Niche Model.
`divergence_max`	Defaults to 0.5. The maximum divergence parameter for the Duplication and Divergence/Mutation mechanisms.
`mutation_max`	Defaults to 0.5. The maximum mutation parameter for the Duplication and Mutation mechanism.
`canonical`	Defautls to FALSE. If TRUE the mechanisms are directed or undirected in accordance with their canonical forms. This negates the value of ‘directed‘.
`cores`	Defaults to 1. The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`verbose`	Defaults to TRUE. Whether to print all messages.

Details

Produces ground-truthing network data.

Value

A list. The first element contains the networks. The second contains their corresponding parameters.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

make_Systematic(net_size = 10)
```

`make_Systematic_canonical` *Systematically Make Networks*

Description

Creates a list of networks that systematically spans mechanisms and their respective parameters.

Usage

```
make_Systematic_canonical(
  net_size,
  neighborhood,
  directed = TRUE,
  net_kind = "matrix",
  resolution = 100,
  resolution_min = 0.01,
  resolution_max = 0.99,
  reps = 3,
  processes = c("ER", "PA", "DM", "SW", "NM"),
  power_max = 5,
  connectance_max = 0.5,
  divergence_max = 0.5,
  mutation_max = 0.5,
  cores = 1,
  verbose = TRUE
)
```

Arguments

`net_size`	Number of nodes in the network.
`neighborhood`	The range of nodes that form connected communities. Note: This implementation results in overlap of communities.
`directed`	Whether the target network is directed. Defaults to TRUE.
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list"). Defaults to "matrix".
`resolution`	The first step is to find the version of each process most similar to the target network. This parameter sets the number of parameter values to search across. Decrease to improve performance, but at the cost of accuracy. Defaults to 100.

`resolution_min`	= The minimum parameter value to consider. Zero is not used because in many processes it results in degenerate systems (e.g. entirely unconnected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.01.
`resolution_max`	The maximum parameter value to consider. One is not used because in many processes it results in degenerate systems (e.g. entirely connected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.99.
`reps`	Defaults to 3. The number of networks to simulate for each parameter. More replicates increases accuracy by making the estimation of the parameter that produces networks most similar to the target network less idiosyncratic.
`processes`	Defaults to c("ER", "PA", "DD", "SW", "NM"). Vector of process abbreviations. Currently only the default five are supported. Future versions will accept user-defined network-generating functions and associated parameters. ER = Erdos-Renyi random. PA = Preferential Attachment. DD = Duplication and Divergence. SW = Small World. NM = Niche Model.
`power_max`	= Defaults to 5. The maximum power of attachment in the Preferential Attachment process (PA).
`connectance_max`	= Defaults to 0.5. The maximum connectance parameter for the Niche Model.
`divergence_max`	= Defaults to 0.5. The maximum divergence parameter for the Duplication and Divergence/Mutation mechanisms.
`mutation_max`	= Defaults to 0.5. The maximum mutation parameter for the Duplication and Mutation mechanism.
`cores`	= Defaults to 1. The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`verbose`	= Defaults to TRUE. Whether to print all messages.

Details

Produces ground-truthing network data.

Value

A list. The first element contains the networks. The second contains their corresponding parameters.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

make_Systematic(net_size = 10)
```

make_Systematic_directedCanonicalLike *Systematically Make Networks*

Description

Creates a list of networks that systematically spans mechanisms and their respective parameters.

Usage

```
make_Systematic_directedCanonicalLike(
  net_size,
  directed = TRUE,
  net_kind = "matrix",
  resolution = 100,
  resolution_min = 0.01,
  resolution_max = 0.99,
  reps = 3,
  processes = c("ER", "PA", "DM", "SW", "NM"),
  power_max = 5,
  connectance_max = 0.5,
  divergence_max = 0.5,
  mutation_max = 0.5,
  cores = 1,
  verbose = TRUE
)
```

Arguments

`net_size`	Number of nodes in the network.
`directed`	Whether the target network is directed. Defaults to TRUE.
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list"). Defaults to "matrix".
`resolution`	The first step is to find the version of each process most similar to the target network. This parameter sets the number of parameter values to search across. Decrease to improve performance, but at the cost of accuracy. Defaults to 100.
`resolution_min`	= The minimum parameter value to consider. Zero is not used because in many processes it results in degenerate systems (e.g. entirely unconnected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.01.
`resolution_max`	The maximum parameter value to consider. One is not used because in many processes it results in degenerate systems (e.g. entirely connected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.99.

`reps`	Defaults to 3. The number of networks to simulate for each parameter. More replicates increases accuracy by making the estimation of the parameter that produces networks most similar to the target network less idiosyncratic.
`processes`	Defaults to c("ER", "PA", "DD", "SW", "NM"). Vector of process abbreviations. Currently only the default five are supported. Future versions will accept user-defined network-generating functions and associated parameters. ER = Erdos-Renyi random. PA = Preferential Attachment. DD = Duplication and Divergence. SW = Small World. NM = Niche Model.
`power_max`	= Defaults to 5. The maximum power of attachment in the Preferential Attachment process (PA).
`connectance_max`	= Defaults to 0.5. The maximum connectance parameter for the Niche Model.
`divergence_max`	= Defaults to 0.5. The maximum divergence parameter for the Duplication and Divergence/Mutation mechanisms.
`mutation_max`	= Defaults to 0.5. The maximum mutation parameter for the Duplication and Mutation mechanism.
`cores`	= Defaults to 1. The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`verbose`	= Defaults to TRUE. Whether to print all messages.

Details

Produces ground-truthing network data.

Value

A list. The first element contains the networks. The second contains their corresponding parameters.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

make_Systematic(net_size = 10)
```

make_Systematic_mixture

Systematically Make Networks

Description

Creates a list of networks that systematically spans mechanisms and their respective parameters.

Usage

```
make_Systematic_mixture(
  net_size,
  neighborhood,
  directed = TRUE,
  net_kind = "matrix",
  resolution = 100,
  resolution_min = 0.01,
  resolution_max = 0.99,
  reps = 3,
  processes = c("ER", "PA", "DM", "SW", "NM"),
  power_max = 5,
  connectance_max = 0.5,
  divergence_max = 0.5,
  mutation_max = 0.5,
  canonical = FALSE,
  cores = 1,
  verbose = TRUE
)
```

Arguments

`net_size`	Number of nodes in the network.
`neighborhood`	The range of nodes that form connected communities. Note: This implementation results in overlap of communities.
`directed`	Whether the target network is directed. Defaults to TRUE.
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list"). Defaults to "matrix".
`resolution`	The first step is to find the version of each process most similar to the target network. This parameter sets the number of parameter values to search across. Decrease to improve performance, but at the cost of accuracy. Defaults to 100.
`resolution_min`	= The minimum parameter value to consider. Zero is not used because in many processes it results in degenerate systems (e.g. entirely unconnected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.01.

`resolution_max`	The maximum parameter value to consider. One is not used because in many processes it results in degenerate systems (e.g. entirely connected networks). Currently process agnostic. Future versions will accept a vector of values, one for each process. Defaults to 0.99.
`reps`	Defaults to 3. The number of networks to simulate for each parameter. More replicates increases accuracy by making the estimation of the parameter that produces networks most similar to the target network less idiosyncratic.
`processes`	Defaults to c("ER", "PA", "DD", "SW", "NM"). Vector of process abbreviations. Currently only the default five are supported. Future versions will accept user-defined network-generating functions and associated parameters. ER = Erdos-Renyi random. PA = Preferential Attachment. DD = Duplication and Divergence. SW = Small World. NM = Niche Model.
`power_max`	Defaults to 5. The maximum power of attachment in the Preferential Attachment process (PA).
`connectance_max`	Defaults to 0.5. The maximum connectance parameter for the Niche Model.
`divergence_max`	Defaults to 0.5. The maximum divergence parameter for the Duplication and Divergence/Mutation mechanisms.
`mutation_max`	Defaults to 0.5. The maximum mutation parameter for the Duplication and Mutation mechanism.
`canonical`	Defautls to FALSE. If TRUE the mechanisms are directed or undirected in accordance with their canonical forms. This negates the value of ‘directed‘.
`cores`	= Defaults to 1. The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`verbose`	= Defaults to TRUE. Whether to print all messages.

Details

Produces ground-truthing network data.

Value

A list. The first element contains the networks. The second contains their corresponding parameters.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

make_Systematic(net_size = 10)
```

null_fit_optim *Empirical parameterization via null distributions*

Description

Helper function to find the best fitting version of a mechanism by searching across the null distributions associated with a process + parameter combination.

Usage

```
null_fit_optim(
  parameter,
  process,
  network,
  net_size,
  iters,
  neighborhood,
  directed,
  DD_kind,
  DD_weight,
  net_kind,
  mechanism_kind,
  method,
  size_different,
  power_max,
  connectance_max,
  divergence_max,
  best_fit_sd,
  max_norm,
  cause_orientation,
  cores,
  null_dist_trim,
  ks_dither,
  ks_alternative,
  verbose = FALSE
)
```

Arguments

`parameter`	The parameter being tested for its ability to generate networks alike the input ‘network‘.
`process`	Name of mechanism. Currently only "ER", "PA", "DD", "DM" "SW", and "NM" are supported. Future versions will accept user-defined network-generating functions and associated parameters. ER = Erdos-Renyi random. PA = Preferential Attachment. DD = Duplication and Divergence. DM = Duplication and Mutation. SW = Small World. NM = Niche Model.

`network`	The network being compared to a hypothesized ‘process‘ with a given ‘parameter‘ value.
`net_size`	Number of nodes in the network.
`iters`	Number of replicates in the null distribution. Note that length(null_dist) = ((iters^2)-iters)/2.
`neighborhood`	The range of nodes that form connected communities. Note: This implementation results in overlap of communities.
`directed`	Whether the target network is directed.
`DD_kind`	A vector of network properties to be used to compare networks.
`DD_weight`	A vector of weights for the relative importance of the network properties in DD_kind being used to compare networks. Should be the same length as DD_kind.
`net_kind`	If the network is an adjacency matrix ("matrix") or an edge list ("list").
`mechanism_kind`	Either "canonical" or "grow" can be used to simulate networks. If "grow" is used, note that here it will only simulate pure mixtures made of a single mechanism.
`method`	This determines the method used to compare networks at the heart of the classification. Currently "DD" (Degree Distribution) and "align" (the align function which compares networks by the entropy of diffusion on them) are supported. Future versions will allow user-defined methods.
`size_different`	If there is a difference in the size of the networks used in the null distribution.
`power_max`	The maximum power of attachment in the Preferential Attachment process (PA).
`connectance_max`	The maximum connectance parameter for the Niche Model.
`divergence_max`	The maximum divergence parameter for the Duplication and Divergence/Mutation mechanisms.
`best_fit_sd`	Standard Deviation used to simulate networks with a similar but not identical best fit parameter. This is important because simulating networks with the identical parameter artificially inflates the false negative rate by assuming the best fit parameter is the true parameter. For large resolution and reps values this will become true, but also computationally intractable for realistically large systems.
`max_norm`	Binary variable indicating if each network property should be normalized so its max value (if a node-level property) is one.
`cause_orientation`	The orientation of directed adjacency matrices.
`cores`	The number of cores to run the classification on. When set to 1 parallelization will be ignored.
`null_dist_trim`	= Number between zero and one that determines how much of each network comparison distribution (unknown network compared to simulated networks, simulated networks compared to each other) should be used. Prevents p-value convergence with large sample sizes. Defaults to 1, which means all comparisons are used (no trimming).
`ks_dither`	The KS test cannot compute exact p-values when every pairwise network distance is not unique. Adding small amounts of noise makes each distance unique. We are not aware of a study on the impacts this has on accuracy so it is set to zero by default.

`ks_alternative`	Governs the KS test. Assuming best_fit_sd is not too large, this can be set to "greater" because the target network cannot be more alike identically simulated networks than they are to each other. In practice we have found "greater" and "less" produce numerical errors. Only "two.sided", "less", and "greater" are supported through stats::ks.test().
`verbose`	Defaults to TRUE. Whether to print all messages.

Details

Note: Currently each process is assumed to have a single governing parameter.

Value

A number measuring how different the input network is from the parameter + process combination.

References

Langendorf, R. E., & Burgess, M. G. (2020). Empirically Classifying Network Mechanisms. arXiv preprint arXiv:2012.15863.

Examples

```
# Import netcom
library(netcom)

# Adjacency matrix
size <- 10
network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)

# Calculate how similar the input network is to Small-World networks with
# a rewiring probability of 0.28.
null_fit_optim(
     parameter = 0.28,
     process = "SW",
     network = network,
     net_size = 12,
     iters = 20,
     neighborhood = max(1, round(0.1 * net_size)),
     net_kind = "matrix",
     mechanism_kind = "grow",
     power_max = 5,
     connectance_max = 0.5,
     divergence_max = 0.5,
     cores = 1,
     directed = TRUE,
     method = "DD",
     size_different = FALSE,
     cause_orientation = "row",
     DD_kind = c(
         "in", "out", "entropy_in", "entropy_out",
         "clustering_coefficient", "page_rank", "communities"
     ),
     DD_weight = 1,
     best_fit_sd = 0,
     max_norm = FALSE,
     null_dist_trim = 0,
     ks_dither = 0,
     ks_alternative = "two.sided",
     verbose = FALSE
)
```

stir_DD *Sitrs a Duplication and Divergence Network*

Description

Stirs an already existing network by rewiring a node according to the Duplication and Divergence mechanism.

Usage

```
stir_DD(
  matrix,
  x,
  divergence,
  directed = TRUE,
  link = 0,
  force_connected = FALSE
)
```

Arguments

`matrix`	Existing network to be rewired (stirred).
`x`	The ID of the node to be grown.
`divergence`	Probability that the new node loses edges associated with the node it duplicates. Needs to be between zero and one.
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix.
`link`	Probability that the new node attaches to the node it duplicates. Defaults to 0.
`force_connected`	Binary argument determining if the newly grown node has to be connected to the existing network. Defaults to FALSE, to prevent rare computational slow-downs when it is unlikely to create a connected network. Defaults to FALSE.

Details

Different from Duplication & Mutation models in that edges can only be lost.

Value

An adjacency matrix.

References

Ispolatov, I., Krapivsky, P. L., & Yuryev, A. (2005). Duplication-divergence model of protein interaction network. Physical review E, 71(6), 061911.

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
new_network_prep[1:size, 1:size] = existing_network
new_network <- stir_DD(matrix = new_network_prep, x = size + 1, divergence = 0.5)
```

---

stir_DM *Stirs a Duplication and Mutation Network*

Description

Stirs an already existing network by rewiring a node according to the Duplication and Mutation mechanism.

Usage

```
stir_DM(
  matrix,
  x,
  divergence,
  mutation,
  directed = TRUE,
  link = 0,
  force_connected = FALSE
)
```

Arguments

matrix	Existing network to experience growth.
x	The ID of the node to be rewired (stirred).
divergence	Probability that the new node loses edges associated with the node it duplicates. Needs to be between zero and one.

`mutation`	Probability that the new node gains edges not associated with the node it duplicates. Needs to be between zero and one.
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix.
`link`	Probability that the new node attaches to the node it duplicates. Defaults to 0.
`force_connected`	Binary argument determining if the newly grown node has to be connected to the existing network. Defaults to FALSE, to prevent rare computational slow-downs when it is unlikely to create a connected network. Defaults to FALSE.

Details

Different from Duplication & Mutation models in that edges can only be lost.

Value

An adjacency matrix.

References

Ispolatov, I., Krapivsky, P. L., & Yuryev, A. (2005). Duplication-divergence model of protein interaction network. Physical review E, 71(6), 061911.

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
new_network_prep[1:size, 1:size] = existing_network
new_network <- stir_DM(matrix = new_network_prep, x = size + 1, divergence = 0.5, mutation = 0.21)
```

`stir_ER` *Stir an Erdos-Renyi Random Network*

Description

Stirs an already existing network by rewiring a node according to the Erdos-Renyi random mechanism.

Usage

```
stir_ER(matrix, x, p, directed = TRUE, retcon = FALSE)
```

Arguments

`matrix`	Existing network to experience growth.
`x`	The ID of the node to be rewired (stirred).
`p`	Probability possible edges exist. Needs to be between zero and one.
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix.
`retcon`	Binary variable determining if already existing nodes can attach to new nodes. Defaults to FALSE.

Details

Different from Duplication & Mutation models in that edges can only be lost.

Value

An adjacency matrix.

References

Erdos, P. and Renyi, A., On random graphs, Publicationes Mathematicae 6, 290–297 (1959).

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
new_network_prep[1:size, 1:size] = existing_network
new_network <- stir_ER(matrix = new_network_prep, x = size + 1, p = 0.5)
```

`stir_NM` *Stirs a Niche Model Network*

Description

Stirs an already existing network by rewiring a node according to the Niche Model mechanism.

Usage

```
stir_NM(matrix, x, niches, directed = TRUE, connectance = 0.2)
```

Arguments

`matrix`	Existing network to experience rewiring (stirring).
`x`	The ID of the node to be grown.
`niches`	Vector of length x, with values between zero and one corresponding to each node's niche.
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix. Defaults to TRUE.
`connectance`	Niche Model parameter specifying the expected connectivity of the network, which determines for a given node the niche space window within which it attaches to every other node. Defaults to 0.2.

Details

Stirs a node in a Niche Model network.

Value

An adjacency matrix.

References

Williams, R. J., & Martinez, N. D. (2000). Simple rules yield complex food webs. Nature, 404(6774), 180-183.

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
new_network_prep[1:size, 1:size] = existing_network
new_network <- stir_NM(
    matrix = new_network_prep,
    x = size + 1,
    connectance = 0.1,
    niches = runif(size + 1)
)
```

stir_PA *Stirs a Preferential Attachment Network*

Description

Stirs an already existing network by rewiring a node according to the Preferential Attachment mechanism.

Usage

```
stir_PA(
  matrix,
  x,
  power,
  directed = TRUE,
  retcon = FALSE,
  sum_v_max = "max",
  nascent_help = TRUE
)
```

Arguments

`matrix`	Existing network to experience growth.
`x`	The ID of the node to be rewired (stirred).
`power`	Power of attachment, which determines how much new nodes prefer to attach to nodes that have many edges compared to few. Needs to be positive.
`directed`	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix.
`retcon`	Binary variable determining if already existing nodes can attach to new nodes. Defaults to FALSE.
`sum_v_max`	Degree distributions must be normalized, either by their "max" or "sum". Defaults to "max".
`nascent_help`	Should a single edge be added to the degree distribution of all nodes so that nodes with a zero in-degree can still have a chance of being attached to by new nodes. Defaults to TRUE.

Details

Rewires a node in a network according to the Preferential Attachment mechanism.

Value

An adjacency matrix.

References

Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. science, 286(5439), 509-512.

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
new_network_prep[1:size, 1:size] = existing_network
new_network <- stir_PA(matrix = new_network_prep, x = size + 1, power = 2.15)
```

---

stir_SW *Stirs a Small-World Network*

Description

Stirs an already existing network by rewiring a node according to the Small-World mechanism.

Usage

```
stir_SW(matrix, x, rewire, directed = TRUE)
```

Arguments

matrix	Existing network to experience growth.
x	The ID of the node to be grown.
rewire	Small-World parameter specifying the probability each edge is randomly rewired, allowing for the possiblity of bridges between connected communities.
directed	Binary variable determining if the network is directed, resulting in off-diagonal asymmetry in the adjacency matrix.

Details

Rewires a node in a network according to the Small-World mechanism.

Value

An adjacency matrix.

References

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of ‘small-world’networks. nature, 393(6684), 440-442.

Examples

```
# Import netcom
library(netcom)

size <- 10
existing_network <- matrix(sample(c(0,1), size = size^2, replace = TRUE), nrow = size, ncol = size)
new_network_prep <- matrix(0, nrow = size + 1, ncol = size + 1)
new_network_prep[1:size, 1:size] = existing_network
new_network <- stir_SW(matrix = new_network_prep, x = size + 1, rewire = 0.213)
```

Index