

Package: hypergraphs (via r-universe)

October 12, 2026

Title Higher-Order Network Analysis

Version 0.7.5

Description Build and analyze higher-order networks, in which a relation can involve more than two nodes or can depend on more than the current node. Covers the three structure families of the higher-order network literature Battiston et al. (2020) [10.1016/j.physrep.2020.05.004](https://doi.org/10.1016/j.physrep.2020.05.004), plus a text family that reads a corpus as a hypergraph. Every object is read through one accessor, and every sequence-taking verb reads long, wide, list or model input. Memory networks from categorical sequences: higher-order networks with rule extraction Xu, Wickramaratne, and Chawla (2016) [10.1126/sciadv.1600028](https://doi.org/10.1126/sciadv.1600028), higher-order network embedding Saebi et al. (2020) [10.1089/big.2019.0169](https://doi.org/10.1089/big.2019.0169), hypergeometric path anomaly detection LaRock et al. (2020) [10.1137/1.9781611976236.52](https://doi.org/10.1137/1.9781611976236.52), the multi-order generative model Scholtes (2017) [10.1145/3097983.3098145](https://doi.org/10.1145/3097983.3098145), higher-order centralities Scholtes, Wider, and Garas (2016) [10.1140/epjb/e2016-60663-0](https://doi.org/10.1140/epjb/e2016-60663-0), a permutation-based Markov order test, a path-dependence diagnostic, and bootstrap and permutation inference for extracted rules. Simplicial complexes: clique, Vietoris-Rips and pathway complexes, Betti numbers, persistent homology, persistence landscapes, bottleneck and Wasserstein distances, and q-analysis. Hypergraphs: construction from network cliques, sliding windows over sequences, bipartite group membership, or embedding nearest neighbours; structural measures; tensor eigenvector centralities Benson (2019) [10.1137/18M1203031](https://doi.org/10.1137/18M1203031); PageRank with edge-dependent vertex weights Chitra and Raphael (2019) [10.48550/arXiv.1905.08287](https://doi.org/10.48550/arXiv.1905.08287); the weighted normalized hypergraph Laplacian of Zhou, Huang, and Scholkopf (2006) [10.7551/mitpress/7503.003.0205](https://doi.org/10.7551/mitpress/7503.003.0205) with spectral, symmetric non-negative matrix factorization (NMF), and joint-NMF clustering Hayashi et al. (2020) [10.1145/3340531.3412034](https://doi.org/10.1145/3340531.3412034) and transductive label spreading; projections to weighted graphs and s-line graphs; Bernoulli-incidence,

stochastic-block, uniform and regular random generators; temporal snapshots; hyperedge betweenness and closeness; Infomap ensembles with adjusted- mutual-information medoid selection; Y/T/O motif profiles and null models by degree-preserving swaps and the configuration model; and hypergraph neural networks in native 'torch': the hypergraph neural network (HGNN), hypergraph attention networks (HyperGAT and its heterogeneous variant), the hypergraph convolutional network (HyperGCN), hypergraph networks with hyperedge neurons (HNNH), AllDeepSets and AllSetTransformer. Text hypergraphs: a corpus becomes a weighted document-word hypergraph in either orientation (bag of words with smoothed term frequency-inverse document frequency (tf-idf), token windows, or embedding nearest neighbours), analysed with tidy verbs for clustering, keyword extraction, few-label classification, and null testing.

License MIT + file LICENSE

URL <https://github.com/mohsaqr/hypergraphs>,
<https://pak.dynasite.org/hypergraphs/>

BugReports <https://github.com/mohsaqr/hypergraphs/issues>

Language en-US

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.4)

Imports cograph (>= 2.7.2), ggplot2, grDevices, grid, Matrix, methods, Nestimate (>= 0.9.26), parallel, RSpectra, stats, tools, utils

Suggests gridExtra, igraph, knitr, png, rmarkdown, sbert, testthat (>= 3.0.0), torch

Additional_repositories <https://mohsaqr.r-universe.dev>

VignetteBuilder knitr

Config/testthat/edition 3

LazyData true

Repository <https://mohsaqr.r-universe.dev>

Date/Publication 2026-10-12 13:21:40 UTC

RemoteUrl <https://github.com/mohsaqr/hypergraphs>

RemoteRef HEAD

RemoteSha 9ec1d27f7ab7ced4ef4e15664ecb46a23450eac3

Contents

clean_text	6
covid_abstracts	8
covid_embeddings	9
covid_sample	9
dual_hypergraph	10
forum_posts	11
gfcc_citations	12
gfcc_decisions	13
group_hypergraph	14
heterogeneous_hgat	18
hg_agreement	19
hg_allset	21
hg_assortativity	22
hg_betti	24
hg_bootstrap	25
hg_bottleneck	27
hg_centrality	28
hg_centrality.net_hg	29
hg_centrality.net_hon	32
hg_classify	34
hg_cluster	36
hg_coccluster	38
hg_communities	40
hg_communities.net_hg	41
hg_communities.net_hon	43
hg_community_quality	45
hg_compare	46
hg_compare_communities	48
hg_degree	50
hg_degree_correlation	51
hg_dictionary	52
hg_edge_centrality	53
hg_edges	54
hg_embed	56
hg_euler	57
hg_get	58
hg_get.hg_classification	60
hg_get.hg_hypergat	62
hg_get.hypergraphs_bootstrap	63
hg_get.hypergraphs_comparison	64
hg_get.hypergraphs_hypa	65
hg_get.hypergraphs_memory_communities	66
hg_get.hypergraphs_result	68
hg_get.hypergraphs_summary	69
hg_get.net_hg	70
hg_get.net_hg_cluster	72

hg_get.net_hg_measures	72
hg_get.net_hg_transduction	73
hg_get.net_hon	74
hg_get.net_honem	75
hg_get.net_markov_order	76
hg_get.net_markov_order_group	77
hg_get.net_markov_stability	77
hg_get.net_mogen	79
hg_get.net_path_dependence	80
hg_get.net_temporal_hypergraph	81
hg_get.persistence_landscape	82
hg_get.persistent_homology	83
hg_get.q_analysis	84
hg_get.simplicial_complex	84
hg_get.text_hypergraph	86
hg_growth	87
hg_hnhn	89
hg_homology	90
hg_hypergat	91
hg_hypergc	95
hg_joint_cluster	96
hg_keywords	97
hg_landscape	100
hg_laplacian	101
hg_line_graph	102
hg_markov_stability	104
hg_membership	105
hg_mmsbm	106
hg_modularity	110
hg_motifs	112
hg_neural	113
hg_null_test	115
hg_pagerank	117
hg_qanalysis	119
hg_representations	120
hg_seeds	121
hg_sequences	121
hg_snapshot	124
hg_snapshots	125
hg_stability	126
hg_subset	128
hg_topic_quality	130
hg_topic_search	133
hg_topic_sizes	135
hg_topics	136
hg_transitivity	139
hg_wasserstein	141
hon	142

honem	144
hypo	145
hypo.default	146
hypo.net_hg	148
hypergraph	150
icsid_tribunals	152
knn_hypergraph	153
long-data	154
markov_order	155
memory	156
mogen	158
network_hypergraph	159
pairwise_network	161
plot.hypergraphs_bootstrap	163
plot.hypergraphs_comparison	164
plot.hypergraphs_memory_communities	165
plot.net_hg	166
plot.net_hg_cluster	173
plot.net_hg_pairwise	174
plot.net_hg_transduction	175
plot.net_temporal_hypergraph	175
predict.hg_hypergat	177
print.hg_communities	179
print.hypergraphs_bootstrap	180
print.hypergraphs_comparison	181
print.hypergraphs_memory_communities	181
print.hypergraphs_memory_group	182
print.net_hg_cluster	183
print.net_hg_measures	183
print.net_hg_transduction	185
print.text_hypergraph	185
random_hypergraph	186
ring_communities	187
ring_sequences	188
simplicial	189
stop_words_en	193
summary.net_hg_cluster	194
summary.net_hg_transduction	195
temporal_hypergraph	195
text_hypergraph	198
topic_network	202
tutoring_events	204
window_hypergraph	205

clean_text

*Clean a corpus before building a text hypergraph***Description**

Repairs and strips the non-content a bibliographic or web export leaves in a text: HTML tags and entities, mojibake and typographic characters, bracketed citation numbers and list or section numbering, URLs and DOIs, trailing copyright notices, bare numbers, and optionally stop words. Each step is a switch. The result has the same length (or the same rows) as the input; a text that falls below min_content becomes the empty string, which `text_hypergraph()` then drops with a `hypergraphs_dropped_documents` warning, so nothing leaves the corpus silently.

Usage

```
clean_text(
  x,
  column = NULL,
  html = TRUE,
  encoding = TRUE,
  citations = TRUE,
  urls = TRUE,
  copyright = TRUE,
  copyright_max = 300L,
  boilerplate = FALSE,
  numbers = TRUE,
  remove = NULL,
  stop_words = NULL,
  min_chars = 0L,
  min_content = 0
)
```

Arguments

<code>x</code>	A character vector, or a data.frame with a text column.
<code>column</code>	When <code>x</code> is a data.frame, the name of its text column.
<code>html</code>	Decode HTML entities (<code>&amp;</code> , <code>&nbsp;</code> , <code>&#8217;</code> , ...) and strip tags (default TRUE). A numeric entity that names no character (zero, a UTF-16 surrogate in U+D800-U+DFFF, or a value above U+10FFFF) becomes the replacement character U+FFFD, as the HTML standard decodes it.
<code>encoding</code>	Repair UTF-8-read-as-Latin-1 mojibake (the three-character garble of a curly apostrophe, the four-character garble of an emoji) and normalise typographic quotes, dashes and non-breaking spaces to their ASCII forms (default TRUE).
<code>citations</code>	Remove bracketed reference numbers (<code>[12]</code> , <code>[3, 4]</code> , <code>[1-5]</code>), leading list markers (<code>1.</code> , <code>(3)</code> , <code>a)</code>) and dotted section numbers (<code>3.2.1</code>) (default TRUE).
<code>urls</code>	Remove URLs, <code>www.</code> addresses, <code>doi:</code> strings and bare DOIs (default TRUE).

copyright	Remove a trailing copyright notice: everything from the last copyright sign, (c) or Copyright to the end when that tail is under copyright_max characters, plus "All rights reserved" (default TRUE).
copyright_max	Longest tail treated as a notice (default 300).
boilerplate	Remove publisher boilerplate wherever it occurs in a text, not only at the end (default FALSE): publisher names (IEEE, Elsevier, Springer, Wiley, Taylor & Francis, Informa, Emerald, Routledge, SAGE, MDPI, Frontiers, Oxford and Cambridge University Press, the American Chemical Society and its Division of Chemical Education), company suffixes (Inc, Ltd, LLC, B.V., GmbH) and the phrases of licence and rights notices ("all rights reserved", "under exclusive licence", "the author(s)", "published by", "open access", "Creative Commons", CC licence codes, "copyright"). Publishers change the wording of their notices over time, so a classifier trained on abstracts with boilerplate learns the publisher and the year.
numbers	Remove bare numbers, percentages and years (default TRUE); a number joined to letters, such as covid19, p53 or covid-19, is a word and is kept whole. Numbers never enter a text hypergraph's vocabulary anyway (the tokeniser keeps alphabetic tokens), so this matters for display and for min_content.
remove	Extra patterns to remove (Perl regular expressions, case-insensitive), applied after the repairs and before the content floor: a database's placeholder for a missing abstract, a publisher's own boilerplate. Default NULL.
stop_words	Words to remove, case-insensitively at word boundaries (default NULL, none). Usually left to <code>text_hypergraph()</code> 's own stop_words; use it here when the cleaned text itself is shown.
min_chars	Minimum number of characters for a word to be kept (default 0L, keep everything). Raise it to drop the single letters and short fragments left by initials, enumerations and hyphenated line breaks; min_chars = 3 keeps words of three characters or more.
min_content	Minimum share of alphabetic characters among non-space characters, judged on the repaired text before numbers are removed (default 0, keep everything). A reference list or a table fragment scores low; 0.5 is a sensible floor for abstracts.

Value

x with the text cleaned: a character vector of the same length (names kept), or the same data.frame with column replaced. NA and low-content texts become "". Raises hypergraphs_bad_input for a malformed x, column or switch.

References

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. doi:10.1017/CBO9780511809071

Examples

```
messy <- c(
  "1. The programme raised attainment in 2020, see section 3.2.1.",
```

```

    "Peer&nbsp;support <b>improved</b> outcomes [12] across schools.",
    "Full report at https://example.org/study.pdf and doi:10.1/x.",
    "Students' views shifted. (c) 2021 Informa UK Limited.",
    "(3) OJ No L 297, 24.11.1979, p. 1."
  )
  clean_text(messy)
  clean_text(messy, min_content = 0.5)
  clean_text(c("[No abstract available]", "A real abstract."),
    remove = "no abstract available")

```

covid_abstracts

COVID-19 education research abstracts

Description

A deterministic stratified sample of 165 abstracts (up to 40 per year, 2020–2024) from a Scopus export of COVID-19 education research, the same source corpus used by the `sbert` package's topic-modeling articles. Rebuilt by `data-raw/covid_abstracts.R`.

Usage

```
covid_abstracts
```

Format

A data frame with 165 rows and 4 columns:

doc Scopus EID, the unique document identifier.

title Article title.

abstract Abstract text (each at least 400 characters).

year Publication year (integer, 2020–2024).

Source

Scopus export of COVID-19 education research, 2020–2024.

Examples

```

hg <- text_hypergraph(covid_abstracts, column = "abstract", id = "doc")
hg

```

covid_embeddings	<i>Sentence embeddings of the COVID-19 abstracts</i>
------------------	--

Description

Sentence embeddings of [covid_abstracts](#)' abstract texts, computed with the `sbert` package's pinned `all-MiniLM-L6-v2` model (L2-normalized rows). Bundled so that `text_hypergraph(construction = "knn")` runs offline; rebuilt by `data-raw/covid_embeddings.R`.

Usage

```
covid_embeddings
```

Format

A numeric matrix with 165 rows (`rownames = covid_abstracts$doc`) and 384 columns.

Source

Computed from [covid_abstracts](#) with `sbert::encode()`.

Examples

```
hg <- text_hypergraph(covid_abstracts, column = "abstract", id = "doc",
  construction = "knn", k = 10,
  embeddings = covid_embeddings)

hg
```

covid_sample	<i>A random sample of 1,000 COVID-19 education research abstracts</i>
--------------	---

Description

A simple random sample of 1,000 abstracts from the Scopus export of COVID-19 education research that the `sbert` package also uses (4,187 records, 2020–2024). Records without an abstract of at least 400 characters are excluded before sampling, and an abstract that appears twice is kept once. The years are represented in the proportions of the source. Rebuilt by `data-raw/covid_sample.R`.

Usage

```
covid_sample
```

Format

A data frame with 1,000 rows and 4 columns:

doc Scopus EID, the unique document identifier.

title Article title.

abstract Abstract text (each at least 400 characters).

year Publication year (integer, 2020–2024).

Source

Scopus export of COVID-19 education research, 2020–2024.

Examples

```
hg <- text_hypergraph(covid_sample, column = "abstract", id = "doc",
                      stop_words = stop_words_en(), min_count = 5)
hg
```

dual_hypergraph	<i>Dual of a hypergraph</i>
-----------------	-----------------------------

Description

Returns the dual hypergraph: every hyperedge becomes a vertex and every vertex becomes a hyperedge, with the transposed weighted incidence. For a bag-construction `text_hypergraph()`, the dual is identical to rebuilding with the opposite nodes orientation (tested), so document-level and word-level analyses can share one constructed object. Duals of windowed and kNN hypergraphs are returned as plain `net_hg` objects (their corpus bookkeeping does not transpose meaningfully).

Usage

```
dual_hypergraph(hg)
```

Arguments

hg A `text_hypergraph()`, `knn_hypergraph()`, or any hypergraphs `net_hg`.

Value

A hypergraph whose incidence is exactly the transpose of `hg`'s, sparse when `hg` is sparse, with every row and column kept: an isolated vertex of `hg` becomes an empty hyperedge of the dual and an empty hyperedge an isolated vertex, so the dual of the dual is `hg`'s incidence again. A `text_hypergraph` with flipped nodes for bag constructions (dense or sparse), otherwise a `net_hg`. Hyperedge metadata (`edge_data`, window counts) describes hyperedges that become vertices, so it is not carried over. Accepted by all `hg_*` verbs.

References

Berge, C. (1989). *Hypergraphs: Combinatorics of Finite Sets*. North-Holland Mathematical Library 45. North-Holland.

Examples

```
hg <- text_hypergraph(c(a = "salt and soup", b = "soup and stars"))
dual <- dual_hypergraph(hg)
dual
hg_measures(dual, what = "edges")
```

forum_posts

Simulated forum posts whose topics follow second-order memory

Description

A corpus whose sequence of topics can only be described with memory. Thirty students each write twelve posts. Each post is about one of three topics, cooking, astronomy or gardening, and holds seven words drawn with replacement from the vocabulary of its topic and one everyday word (night, today, week, friends) shared by all topics. The first two topics of a student are drawn at random. After two different topics, the next topic is the third one with probability 0.8 and otherwise drawn at random; after a repeated topic, it is drawn at random. A first-order model of the topic sequence cannot represent this rule.

Usage

```
forum_posts
```

Format

A data frame with 360 rows (one per post) and 5 columns:

post Character. The post identifier, st01_t01 to st30_t12.

student Character. The author, st01 to st30.

turn Integer. The position of the post among the author's posts, 1 to 12.

topic Character. The topic drawn by the simulation.

text Character. The text of the post, eight words.

Source

Simulated with seed 11 by data-raw/forum_posts.R.

See Also

[hg_sequences\(\)](#), which turns a clustering of these posts into one sequence of topics per student.

Examples

```
head(forum_posts)
forum_hg <- text_hypergraph(forum_posts, column = "text", id = "post")
forum_hg
```

gfcc_citations

Citation blocks of the German Federal Constitutional Court

Description

Every citation between the decisions in [gfcc_decisions](#), grouped into citation blocks: an uninterrupted run of cited decisions inside one decision, the hyperedge of Coupette, Hartung and Katz (2024). One row per citation, dated by both decisions, so backward citations are the rows with `date_citing > date_cited`.

Usage

```
gfcc_citations
```

Format

A data frame with 77,284 rows and 5 columns:

citing Key of the citing decision, the source of the block.

cited Key of the cited decision, the node.

block Citation block identifier, `citing:index`, the hyperedge.

date_citing Date of the citing decision, the time of the block.

date_cited Date of the cited decision.

Source

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs (GFCC dataset). Zenodo. [doi:10.5281/zenodo.8081511](https://doi.org/10.5281/zenodo.8081511), via the replication archive [doi:10.5281/zenodo.8081507](https://doi.org/10.5281/zenodo.8081507). Licence CC BY-NC 4.0. Built by `data-raw/legal_hypergraphs.R`.

References

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. [doi:10.1098/rsta.2023.0141](https://doi.org/10.1098/rsta.2023.0141)

Examples

```
head(gfcc_citations)
blocks <- temporal_hypergraph(gfcc_citations, node = "cited",
                              hyperedge = "block", time = "date_citing",
                              nodes = gfcc_decisions, sparse = TRUE)
summary(blocks)
```

gfcc_decisions*Decisions of the German Federal Constitutional Court*

Description

The 3,618 decisions in volumes 1 to 160 of the official collection of the German Federal Constitutional Court (BVerfGE), 1951 to 2022, as released by Coupette, Hartung and Katz (2024). The node table of the citation-block hypergraph in [gfcc_citations](#): every decision is a node from its own date, whether or not it is ever cited.

Usage

```
gfcc_decisions
```

Format

A data frame with 3,618 rows and 5 columns:

decision Decision key, volume-page.

date Date of the decision.

citation Official citation, e.g. "BVerfGE 1, 1-3".

type Decision type ("Urteil" or "Beschluss").

title Title of the decision, in German.

Source

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs (GFCC dataset). Zenodo. [doi:10.5281/zenodo.8081511](#), via the replication archive [doi:10.5281/zenodo.8081507](#). Licence CC BY-NC 4.0. Built by `data-raw/legal_hypergraphs.R`.

References

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. [doi:10.1098/rsta.2023.0141](#)

Examples

```
head(gfcc_decisions)
```

Description

Constructs a [net_hg](#) the way a network is defined from data. **Membership data** name a node and a hyperedge: every node sharing one value of hyperedge (a session, a team, a citation block) belongs to one hyperedge. An **edge list** names from and to, and every row is a hyperedge of size two. An optional weight column produces a weighted incidence matrix.

Usage

```
group_hypergraph(
  data,
  node = NULL,
  hyperedge = NULL,
  weight = NULL,
  nodes = NULL,
  sparse = FALSE,
  separator = NULL,
  from = NULL,
  to = NULL,
  member = NULL,
  cooccur_by = NULL,
  top = NULL,
  states = NULL,
  threshold = NULL,
  min_size = 1L,
  group = NULL,
  min_share = NULL
)
```

Arguments

data	Data frame in long format, one row per node in a hyperedge or per edge; or a clustering of sequences – a mixture Markov fit (<code>net_mmm</code>), a distance clustering (<code>net_clustering</code>), or the per-cluster networks built from either (<code>netobject_group</code>). See "Clustered sequences" below.
node	Character. Name of the column whose values become the hypergraph's nodes (people, events, words, items).
hyperedge	Character. Name of the column whose shared values bind nodes into one hyperedge (sessions, trials, teams, citation blocks). When neither pair of columns is named, <code>from</code> and <code>to</code> are detected case-insensitively from the alias table <code>temporal_hypergraph()</code> uses (<code>source/target</code> , <code>sender/receiver</code> , ...), and failing that node and hyperedge.
weight	Character or NULL. If supplied, the column is summed per (node, hyperedge) pair to produce a weighted incidence matrix. Default NULL produces a 0/1 binary incidence matrix.

nodes	Optional vector giving the complete node universe. This keeps nodes with no observed group memberships as zero-incidence rows, which is needed for representations such as citation hypergraphs where every decision is a node but some decisions are never cited.
sparse	Logical. Store incidence as a sparse Matrix? Use this for large, sparse event data such as the full GFCC citation-block corpus.
separator	Split the node column on this string, one row per member, before building. Bibliographic exports ship a hyperedge's members as a single delimited cell – EUR-Lex citationcelex and eurovoc, Scopus and Web of Science reference and keyword fields – so separator = ";" replaces the caller's own split, trim and drop-empties. Members empty after trimming are dropped, and a row left with no member contributes no hyperedge.
from, to member, cooccur_by	Column names of a pairwise edge list, as an alternative to node and hyperedge. Deprecated names of node and hyperedge; using them warns with a hypergraphs_deprecated condition.
top	The number of most frequent sets kept as hyperedges, in each value of group or over all hyperedges; Inf keeps every set. For a data frame, giving top, group or min_share counts sets (see "Frequent sets within groups"), and with none of them every value of hyperedge becomes one hyperedge. Clustered sequences and topic models keep 8 by default.
states	For counted sets: the states to keep, such as the events of interest. Every other state is removed from each sequence's set before counting, and a sequence left with no state is not counted. NULL (default) keeps every state. For a topic model, the topics to keep.
threshold	Topic model only: the share at which a topic counts as present in a document, one number in (0, 1]. Required for a topic model and refused for any other input.
min_size	For counted sets (group, top, min_share or a topic model): the smallest set counted (default 1); 2 keeps only the sets of two or more. The share of a set is still taken over all sets of its group.
group	The comparison variable: count frequent sets separately within each value of this column, which must be constant within each hyperedge (an outcome, a cluster, a year). Each value of hyperedge is then one set, the node values it holds, and the top most frequent sets within each value of group become hyperedges (see "Frequent sets within groups"). For a topic model, a column of the documents table of the fitted hypergraph.
min_share	For counted sets: the minimum support, the smallest share of a group's sets that a set must reach to be kept, one number in (0, 1]. Giving it counts sets like top; with no top, every set that reaches the share is kept (Agrawal & Srikant 1994). NULL (default) applies no minimum.

Details

The bipartite representation preserves the full group structure without projecting to a pairwise network. A group of three members A, B, C produces a single 3-hyperedge containing all three, not

three pairwise edges AB, AC, BC. This avoids information loss when group interactions are the primary unit of analysis (Perc et al. 2013).

Unlike `network_hypergraph()` (which derives hyperedges from a network's clique structure), `group_hypergraph()` takes group memberships directly. The two functions are complementary:

- `group_hypergraph()` - when group membership is observed (sessions, transactions, co-authorships).
- `network_hypergraph()` - when only pairwise interactions are observed and triadic structure must be inferred from triangles.

Rows with NA in the node, hyperedge or weight column are dropped silently.

Value

A `net_hg` object with the same structure produced by `network_hypergraph()` (hyperedges, incidence, nodes, `n_nodes`, `n_hyperedges`, `size_distribution`, `params`), plus `edge_data` when data carries hyperedge attributes (see Details). The `params` list records `source = "group_hypergraph"` and the original column names. For clustered sequences the object also has `group_sizes` and `state_counts` (read them with `hg_get()`) and `params` records `source = "clustered_sequences"`, `top`, `states` and `unit = "sequences"`.

Frequent sets within groups

With `group`, `top` or `min_share`, a data frame is read as transactions. Each value of hyperedge (a trial, a session, a basket) is one set, the distinct node values it holds, and the sets are counted within each value of group, which must be constant within a hyperedge. The top most frequent sets of each value of group become hyperedges, counted exactly as for clustered sequences below, and `hg_get(hg, what = "sets")` reads them with their count and share of the value's sets. Without `group`, the sets are counted over all hyperedges together, under one value named after hyperedge ("All sessions"). `plot(hg, group = )` draws the sets of one value, every node sized by the sets that contain it. `states` keeps only the listed node values, and `min_size` the sets of at least that many.

Topic combinations

Given a mixed-membership topic model fitted by `hg_topics()`, every document is reduced to the set of topics whose share in it is at least threshold, the thresholded topic presence used to build topic co-occurrence networks (Abuhay et al. 2017; Cassi et al. 2017). A document on love, hate and romance gives the set of those three topics. The top most frequent sets become hyperedges, counted exactly as for clustered sequences below, with each document as one transaction, and the topics are the nodes. `top = Inf` keeps every set, and `min_size = 2` keeps only the sets that combine two or more topics. `group` names a column of the documents table of the fitted hypergraph, such as a publication year or an author, and the sets are then counted within each of its values; `NULL` (default) counts them over all documents. A document with no topic at the threshold is not counted. Read the sets with `hg_get(hg, what = "sets")`; count is the number of documents with exactly that set and share its proportion of the group's documents.

Clustered sequences

Given a clustering of sequences, every sequence of every group is reduced to the set of its distinct states (order and repetition dropped; NA and empty cells ignored), and the top most frequent sets

of each group become hyperedges – frequent-itemset support counting with each sequence as one transaction (Agrawal & Srikant 1994), restricted to the sets that occur exactly. Sets of equal count are ranked by their name (states sorted and joined by " + "). Groups are named "Cluster 1", "Cluster 2", ... for a `net_mmm` or `net_clustering`, and by the list names of a `netobject_group`, so renamed groups carry through. Integer-coded states are decoded to their labels. The objects are read by their structure; the package that fitted them is not needed.

The hyperedges are named "<group>: <set>" and carry group, set and count (sequences with exactly that set) as hyperedge attributes, so `plot()` colours and titles them by count in "sequences", and `plot(hg, group = "Cluster 1")` draws one group's sets with each node sized by the sequences of that group containing the state. Read the tables with `hg_get(hg, what = "sets")` (one row per hyperedge: its group, set, size, count and share of the group's sequences) and `hg_get(hg, what = "state_counts")` (one row per group and state). A malformed clustering (no \$data, assignments that do not match it, unnamed networks) raises `hypergraphs_bad_input`.

Every other column of data that is constant within a hyperedge (a session's date, a team's department) is kept as a hyperedge attribute in the result's `edge_data` table, one row per hyperedge, where `hg_subset()`'s `where` argument and `pairwise_network()`'s `edge_source` can use it. Columns that vary within a hyperedge describe memberships, not hyperedges, and are left out.

Note

Dense and sparse paths are tested for exact equality. The sparse path is additionally exercised by the full GFCC reproduction from the Legal Hypergraphs Zenodo archive (3,618 nodes, 46,165 hyperedges and 77,187 nonzero incidences).

A dense incidence with more than `.Machine$integer.max` cells cannot be addressed by the flat cell index, and would exhaust memory well before that. It raises the classed error `hypergraphs_dense_too_large` rather than attempting the allocation; pass `sparse = TRUE` for data at that scale.

References

- Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)* (pp. 487-499). Morgan Kaufmann.
- Abuhay, T. M., Kovalchuk, S. V., Bochenina, K., Kamps, G., Krzhizhanovskaya, V. V., & Lees, M. H. (2017). Analysis of computational science papers from ICCS 2001-2016 using topic modeling and graph theory. *Procedia Computer Science*, 108, 7-17. doi:10.1016/j.procs.2017.05.183
- Cassi, L., Lahatte, A., Rafols, I., Sautier, P., & de Turckheim, E. (2017). Improving fitness: Mapping research priorities against societal needs on obesity. *Journal of Informetrics*, 11(4), 1095-1113. doi:10.1016/j.joi.2017.09.010
- Perc, M., Gomez-Gardenes, J., Szolnoki, A., Floria, L. M., & Moreno, Y. (2013). Evolutionary dynamics of group interactions on structured populations: a review. *Journal of the Royal Society Interface* 10(80), 20120997. doi:10.1098/rsif.2012.0997

See Also

`network_hypergraph()` for the clique-based constructor, `temporal_hypergraph()` for the same inputs with a clock.

Examples

```
df <- data.frame(
  person = c("Alice", "Bob", "Carol", "Alice", "Bob",
             "Dave", "Carol", "Dave", "Eve"),
  session = c("S1", "S1", "S1", "S2", "S2",
              "S3", "S3", "S3", "S3")
)
hg <- group_hypergraph(df, node = "person", hyperedge = "session")
print(hg)
summary(hg)

contacts <- data.frame(from = c("a", "b"), to = c("b", "c"))
group_hypergraph(contacts, from = "from", to = "to")

# a clustering of sequences, shaped as a mixture Markov fit (net_mmm)
fit <- structure(list(
  data = data.frame(V1 = c("a", "a", "b", "a", "c"),
                    V2 = c("b", "b", "c", "c", "a"),
                    V3 = c("a", NA, "a", "b", "b")),
  assignments = c(1L, 1L, 2L, 1L, 2L), k = 2L
), class = "net_mmm")
sets <- group_hypergraph(fit, top = 3)
hg_get(sets, what = "sets")
plot(sets, group = "Cluster 1")
```

heterogeneous_hgat	<i>Heterogeneous graph attention classifier (HGAT)</i>
--------------------	--

Description

Implements Linmei et al.'s (2019) heterogeneous graph convolution and dual-level attention. Node types retain separate input feature spaces and transformation matrices. For every target node, type-level attention first weighs the available neighboring information types; node-level attention then weighs individual neighbors before their type-specific transformed representations are combined. This is HGAT for heterogeneous information networks, distinct from Ding et al.'s document-hypergraph [text_hypergat\(\)](#).

Usage

```
heterogeneous_hgat(
  adjacency,
  node_types,
  features,
  labels,
  hidden = 128L,
  layers = 2L,
  epochs = 200L,
```

```

    lr = 0.005,
    weight_decay = 5e-04,
    dropout = 0.5,
    validation = 0.1,
    seed = 1L,
    verbose = FALSE
  )

```

Arguments

adjacency	Symmetric non-negative adjacency matrix with identical unique row and column names.
node_types	Named character vector assigning every node a type.
features	Named list with one numeric feature matrix per type. Rows are named nodes of that type; feature widths may differ across types.
labels	Named character labels for a subset of nodes.
hidden	Common hidden dimension.
layers	Number of dual-attention layers (paper default 2).
epochs, lr, weight_decay, dropout, validation, seed, verbose	Training controls as in hg_neural() .

Value

A prediction data.frame for every heterogeneous-network node, with training history and node types attached as attributes.

References

Linmei, H., Yang, T., Shi, C., Ji, H., & Li, X. (2019). Heterogeneous graph attention networks for semi-supervised short text classification. *EMNLP-IJCNLP 2019*, 4821–4830.

hg_agreement

Agreement between two labelings of the same nodes

Description

Compares any two tidy labelings – partitions from [hg_cluster\(\)](#), predictions from [hg_classify\(\)](#), [hg_neural\(\)](#) or [hg_hypergat\(\)](#) – joined on their shared node column. agreement is the share of nodes with literally equal labels (meaningful when both labelings use the same label set, e.g. a classifier scored against the clustering that produced its seeds); ari is the adjusted Rand index (Hubert & Arabie 1985), which is label-permutation invariant and the right statistic when the two label sets are arbitrary (e.g. two independent clusterings). Adjusted mutual information (ami) and normalized mutual information (nmi) are also available; the legal-hypergraphs workflow uses AMI to select the medoid of repeated Infomap partitions.

Usage

```
hg_agreement(
  x,
  y,
  what = c("summary", "table", "mapping"),
  method = "ari",
  node = NULL,
  label = NULL
)
```

Arguments

x, y	Tidy labelings: data.frames with a node (or, in the text family's document tables, doc) column and a predicted, cluster, community, topic, block or label column (first match in that order wins), one row per node. Nodes are matched by name; nodes present in only one labeling are dropped. A fitted model is read through its one-label-per-node table: a fit of hg_communities() on a hypergraph through its medoid partition (<code>hg_get(fit, what = "medoid")</code>), a hg_mmsbm() fit through each node's strongest community (<code>what = "nodes"</code>), a hg_topics() model through each document's dominant topic (<code>what = "documents"</code>), and a text_hypergraph() through its document table, so a corpus column is scored by naming it in label.
what	"summary" (default) for the one-row comparison, "table" for the tidy contingency table of the joined labels, or "mapping" for one row per label of x naming the label of y that holds most of its nodes – how a partition survives a reweighting or a change of method, topic by topic.
method	One or more label-permutation-invariant measures: "ari" (default), "ami", or "nmi". Ignored for <code>what = "table"</code> .
node, label	Column names, one name for both labelings or two names for x and y in turn, that override the defaults above – <code>hg_agreement(predictions, corpus, label = c("predicted", "year"))</code> scores a classifier against a column of the corpus. <code>node = NULL</code> (default) uses node, else doc; <code>label = NULL</code> (default) keeps the predicted / cluster / community / topic / block / label lookup.

Value

A base data.frame. For `what = "summary"`: one row with columns n (nodes compared), agreement (share of equal labels), aligned (nodes that stay with the majority of their x label in y – the sum of overlap over `what = "mapping"`, so a label-name-free count of how many nodes a re-fit keeps together) and the requested measure columns. For `what = "table"`: one row per label pair with columns label_x, label_y and n. For `what = "mapping"`: one row per label of x with columns label_x, n (its nodes), label_y (the label of y holding most of them), overlap (how many) and share (overlap / n), in the natural order of label_x.

References

Hubert, L., & Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2, 193–218.

Vinh, N. X., Epps, J., & Bailey, J. (2010). Information theoretic measures for clusterings comparison. *Journal of Machine Learning Research*, 11, 2837–2854.

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots",
  cooking_2 = "this soup recipe needs salt on a cold night",
  space_1 = "the telescope revealed a distant galaxy and stars",
  space_2 = "astronomers aimed the telescope at the stars all night"
), stop_words = c("the", "with", "and", "a", "this", "at", "on", "all"))
topics <- hg_cluster(hg, k = 2, seed = 1)
fit <- hg_classify(hg, labels = c(cooking_1 = "Cluster 1",
                                space_1 = "Cluster 2"))

hg_agreement(fit, topics)
hg_agreement(fit, topics, what = "table")
hg_agreement(fit, topics, what = "mapping")
# a labeling read from any table: name its label column
known <- data.frame(doc = c("cooking_1", "space_2"), theme = c("cooking", "space"))
hg_agreement(topics, known, label = c("cluster", "theme"), what = "table")
```

hg_allset

AllSet hypergraph node classifier

Description

Implements the AllSet framework of Chien et al. (2022) as one complete node-to-edge-to-node propagation layer. `model = "deepsets"` uses Eq. 7, an MLP before and after each multiset sum. `model = "transformer"` uses Eq. 8: pooling by multihead attention from a learned query, followed by residual MLPs and layer normalization. Both directions have independent learned multiset functions and are permutation invariant.

Usage

```
hg_allset(
  hg,
  labels,
  features = "incidence",
  model = c("deepsets", "transformer"),
  hidden = 128L,
  heads = 4L,
  epochs = 200L,
  lr = 0.001,
  weight_decay = 5e-04,
  dropout = 0.5,
  validation = 0.1,
  seed = 1L,
  verbose = FALSE
)
```

Arguments

hg	A <code>text_hypergraph()</code> (or any hypergraphs <code>net_hg</code>), dense or sparse.
labels	The known labels: a named character vector (names are node identifiers, values class labels) or a tidy data.frame with a node column and a label, cluster or predicted column. At least two classes.
features	Vertex feature matrix, one row per node in hg node order (rownames must match the node names), e.g. <code>sbert</code> embeddings. The default "incidence" uses the hypergraph's own weighted incidence rows (tf-idf bag-of-words features for a nodes = "doc" text hypergraph).
model	"deepsets" for AllDeepSets or "transformer" for AllSetTransformer.
hidden	Hidden width (default 128).
heads	Number of attention heads for AllSetTransformer. hidden must be divisible by heads.
epochs	Training epochs (default 200).
lr, weight_decay	Adam learning rate and L2 penalty.
dropout	Hidden-layer dropout.
validation	Fraction of labeled nodes held out to select an epoch.
seed	Reproducible R and torch seed.
verbose	Message loss every 20 epochs.

Value

A prediction data.frame in the same format as `hg_neural()`, with training history and the AllSet model name attached.

References

Chien, E., Pan, C., Peng, J., & Milenkovic, O. (2022). You are AllSet: A multiset function framework for hypergraph neural networks. *ICLR 2022*.

hg_assortativity	<i>Degree assortativity of a hypergraph</i>
------------------	---

Description

Measures whether nodes of similar hyperdegree share hyperedges, with the three choice functions of Chodrow (2020, Sec. 4.2). From every hyperedge with at least two members a pair of nodes is chosen, and the coefficient is the correlation of their scores over hyperedges, each hyperedge counting once:

Usage

`hg_assortativity(hg, type = "uniform", scale = c("rank", "degree"))`

Arguments

hg	A hypergraphs net_hg, dense or sparse.
type	One or more of "uniform" (default), "top_2", "top_bottom".
scale	"rank" (default; Chodrow's generalized Spearman) or "degree" (Pearson on raw hyperdegrees).

Details

- "uniform" – a uniformly random pair of distinct members (computed exactly, as the expectation over the random choice; the pair is unordered, so both orders enter with equal weight);
- "top_2" – the two members of largest degree (unordered);
- "top_bottom" – the ordered pair (largest-degree member, smallest-degree member), measuring whether the least-connected member of a group is better connected when its best-connected member is.

The score of a node is its hyperdegree (the number of hyperedges containing it). With scale = "rank" (default) the score is the node's rank by hyperdegree among all nodes (ties averaged), which gives Chodrow's "generalized Spearman" coefficient; scale = "degree" uses the raw hyperdegree (a Pearson coefficient, as XGI does).

On a graph (every hyperedge of size two) the three choice functions give the same pairs, and type = "uniform", scale = "degree" is Newman's degree assortativity (tested against `igraph::assortativity_degree`).

Convention differences with XGI 0.10.2 (`degree_assortativity(exact = TRUE)`), documented and tested: XGI's exact "uniform" weights every hyperedge by its number of member pairs $m(m - 1)$ instead of counting each hyperedge once, so the two agree only on uniform hypergraphs (XGI's Monte Carlo sampler, `exact = FALSE`, samples Chodrow's edge-weighted quantity and agrees within sampling error); XGI symmetrizes "top-bottom" (both orders), whereas Chodrow's definition keeps the order (largest, smallest); XGI returns 0 where the coefficient is undefined. "top_2" agrees with XGI exactly.

The coefficient is undefined (NA) when either chosen score has zero variance across hyperedges, e.g. on a regular hypergraph.

Zero is not the null value for "top_2" and "top_bottom". Both pair order statistics of a hyperedge (its two largest scores, or its largest and smallest), which are correlated even when members are drawn at random, so random hypergraphs give positive values (Chodrow 2020 compares against a null model for this reason). Judge them against coefficients on degree-matched random hypergraphs, not against 0; "uniform" is centred near 0 under random mixing.

Value

A base data.frame, one row per type, with columns type, scale, assortativity (in $[-1, 1]$, NA when undefined) and n_edges (the hyperedges of size at least two that entered).

Conditions

Raises `hypergraphs_bad_input` when hg is not a net_hg or has no hyperedge with at least two members.

References

- Chodrow, P. S. (2020). Configuration models of random hypergraphs. *Journal of Complex Networks*, 8(3), cnaa018. doi:10.1093/comnet/cnaa018
- Battiston, F., Cencetti, G., Iacopini, I., Latora, V., Lucas, M., Patania, A., Young, J.-G., & Petri, G. (2020). Networks beyond pairwise interactions: Structure and dynamics. *Physics Reports*, 874, 1-92. doi:10.1016/j.physrep.2020.05.004

See Also

[hg_degree_correlation\(\)](#), [hg_transitivity\(\)](#).

Examples

```
groups <- data.frame(
  member = c("a", "b", "c", "a", "d", "b", "d", "e", "c", "d", "e", "f"),
  group = c("g1", "g1", "g1", "g2", "g2", "g3", "g3", "g3",
            "g4", "g4", "g4", "g4")
)
hg <- group_hypergraph(groups, node = "member", hyperedge = "group")
hg_assortativity(hg, type = c("uniform", "top_2", "top_bottom"))
```

hg_betti

Betti numbers of a simplicial complex

Description

The ranks of the homology groups with rational coefficients: β_0 counts connected components, β_1 independent loops, β_2 voids, and so on.

Usage

```
hg_betti(sc)
```

Arguments

sc A simplicial_complex from [simplicial\(\)](#).

Details

The coefficients are rational, not $\mathbb{Z}/2$. The two agree on most complexes but not on one with torsion: on the six-vertex real projective plane (RP2) [hg_betti\(\)](#) gives (1, 0, 0), while [hg_homology\(\)](#), which reduces over $\mathbb{Z}/2$ as the persistence literature does, ends its Betti curve at (1, 1, 1). Use [hg_homology\(\)](#) when $\mathbb{Z}/2$ Betti numbers are meant.

Value

A named integer vector c(b0 = , b1 = , ...).

References

Hatcher, A. (2002). *Algebraic Topology*. Cambridge University Press.

See Also

[hg_euler\(\)](#), [hg_homology\(\)](#)

Examples

```
adj <- matrix(c(0, 1, 1, 0,
               1, 0, 1, 1,
               1, 1, 0, 1,
               0, 1, 1, 0), 4, 4, dimnames = list(letters[1:4], letters[1:4]))
hg_betti(simplicial(adj))
```

hg_bootstrap

Bootstrap inference for higher-order network rules

Description

Nonparametric bootstrap over sequences for the rules of a higher-order network (see [hon\(\)](#)): sequences are resampled with replacement, and for every rule edge of the observed network the replicate distribution yields a percentile confidence interval for its conditional probability and a *support* - the fraction of replicates in which the rule's context is extracted as a rule at all. Support close to 1 for a higher-order context means the order elevation is stable under resampling, not an artifact of the particular sample; first-order contexts have support 1 by construction.

Usage

```
hg_bootstrap(
  data,
  n_boot = 500L,
  level = 0.95,
  max_order = 5L,
  min_freq = 1L,
  collapse_repeats = FALSE,
  action = NULL,
  actor = NULL,
  time = NULL,
  parallel = FALSE,
  n_cores = 2L,
  seed = NULL,
  session = NULL,
  time_threshold = 900,
  timezone = "UTC"
)
```

Arguments

data	Sequences in any form described in sequence-input : a long event table (with action), a wide data.frame (one sequence per row), a list of vectors, or a model object carrying its sequences. A group model from hon() with group is resampled group by group with its own settings, and the result is a hypergraphs_bootstrap_group (one bootstrap per group; summary() and hg_get() stack them with a group column).
n_boot	Integer ≥ 2 . Bootstrap replicates. Default 500.
level	Confidence level in (0, 1). Default 0.95 (percentile interval).
max_order, min_freq, collapse_repeats	As in hon() .
action, actor, time, session, time_threshold, timezone	Long-format arguments (see sequence-input); leave the column names NULL for wide or list input.
parallel	Logical. Use <code>parallel::mclapply</code> for the replicates (not on Windows). Results are identical to the serial run.
n_cores	Integer. Cores when <code>parallel = TRUE</code> .
seed	Optional integer seed.

Details

Per-sequence observation counts are computed once; every replicate is a weighted aggregation of those counts followed by counts-based rule extraction (never a re-scan of the data). All randomness is drawn before any parallel work, so `parallel = TRUE` reproduces the serial result under the same seed.

Value

An object of class `hypergraphs_bootstrap`: a list with edges (the tidy inference table, one row per rule edge of the observed network: from, to, order, count, probability, ci_lower, ci_upper, support, n_boot_used), `n_boot`, `level`, `max_order`, `min_freq`, `n_trajectories`, and `seed`. Has `print`, `summary` and `plot` methods; [hg_get\(\)](#) returns the inference table (optionally filtered with `min_support =` or restricted with `order_min =`).

References

- Xu, J., Wickramaratne, T. L., & Chawla, N. V. (2016). Representing higher-order dependencies in networks. *Science Advances* 2(5), e1600028. doi:[10.1126/sciadv.1600028](https://doi.org/10.1126/sciadv.1600028)
- Efron, B., & Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman & Hall.

See Also

[hon\(\)](#), [hg_compare\(\)](#), [markov_order\(\)](#)

Examples

```

hg_seqs <- list(
  c("a", "b", "c", "a", "b", "c"),
  c("x", "b", "d", "x", "b", "d"),
  c("a", "b", "c", "a", "b", "c"),
  c("x", "b", "d", "x", "b", "d")
)
bs <- hg_bootstrap(hg_seqs, n_boot = 50, max_order = 2, seed = 1)
bs
hg_get(bs, sort_by = "support", top = 6)

```

hg_bottleneck

*Bottleneck distance between persistence diagrams***Description**

The largest displacement of the best matching between two persistence diagrams, where a point may also match the diagonal. Essential classes are matched only to essential classes; a dimension whose essential counts differ is Inf apart.

Usage

```
hg_bottleneck(d1, d2, dimension = NULL, tol = .Machine$double.eps^0.5)
```

Arguments

d1, d2	hg_homology() results, or data.frames with columns dimension, birth, death.
dimension	Integer vector of dimensions to compare. NULL compares every dimension present in either diagram.
tol	Tolerance of the binary search. Default <code>.Machine\$double.eps^0.5</code> .

Value

A named numeric vector, one distance per dimension, named "dim_<k>".

References

Cohen-Steiner, D., Edelsbrunner, H., & Harer, J. (2007). Stability of persistence diagrams. *Discrete & Computational Geometry*, 37, 103–120. doi:10.1007/s0045400612765

Edelsbrunner, H., & Harer, J. (2010). *Computational Topology: An Introduction*. American Mathematical Society.

See Also

[hg_wasserstein\(\)](#)

Examples

```
d1 <- data.frame(dimension = 0L, birth = 0, death = 2)
d2 <- data.frame(dimension = 0L, birth = 0, death = 3)
hg_bottleneck(d1, d2)
```

hg_centrality

Centralities of a hypergraph or a memory network

Description

One verb, two estimators, chosen by the class of x:

a hypergraph (net_hg, e.g. from `text_hypergraph()`, `window_hypergraph()`) eigenvector-style hypergraph centralities – clique-motif, Z- and H-eigenvector, EDVW PageRank, subhypergraph and Katz; see `hg_centrality.net_hg()`.

a memory network (net_hon, from `hon()`) PageRank, betweenness and closeness of the higher-order topology, projected onto the first-order states; see `hg_centrality.net_hon()`.

Each method keeps its own arguments; passing an argument that only the other method takes raises `hypergraphs_bad_input`.

Usage

```
hg_centrality(x, ...)

## Default S3 method:
hg_centrality(x, ...)
```

Arguments

```
x          A net_hg or a net_hon.
...        Arguments of the method for class(x).
```

Value

A base `data.frame`, one row per node (or per state), with one column per requested centrality. Any other input raises `hypergraphs_bad_input`.

Examples

```
hg <- text_hypergraph(c(a = "salt and soup", b = "soup and stars"))
hg_centrality(hg, type = "clique")

seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
hg_centrality(hon(seqs, max_order = 2), type = "pagerank")
```

hg_centrality.net_hg *Hypergraph eigenvector centralities*

Description

The hypergraph method of `hg_centrality()`. Computes one or more eigenvector-style centralities on a `net_hg`: *clique-motif* (CEC), *Z-eigenvector* (ZEC), and *H-eigenvector* (HEC). Each variant captures influence differently - CEC flattens group structure via clique expansion, while ZEC and HEC propagate through the higher-order groups directly.

Usage

```
## S3 method for class 'net_hg'
hg_centrality(
  x,
  type = c("clique", "Z", "H"),
  sort_by = NULL,
  n = Inf,
  max_iter = 1000L,
  tol = 1e-08,
  normalize = TRUE,
  damping = 0.85,
  edge_weights = NULL,
  alpha = NULL,
  ...
)
```

Arguments

<code>x</code>	A <code>text_hypergraph()</code> (or any hypergraphs <code>net_hg</code>).
<code>type</code>	Centralities to compute; any of "clique", "Z", "H" (default: all three), "pagerank", "subhypergraph", "katz".
<code>sort_by</code>	Optional centrality name to sort by, descending (ties broken by node name); default keeps node order.
<code>n</code>	Return only the first <code>n</code> rows after sorting (default all) – e.g. <code>sort_by = "clique", n = 10</code> for the ten most central nodes.
<code>max_iter</code>	Maximum number of power-iteration steps of the "Z" and "H" tensor centralities. Default 1000. The "clique" centrality is solved directly (the Perron vector of each connected component) and does not iterate.
<code>tol</code>	Convergence tolerance of the "Z" and "H" iterations, on the L1 change between successive iterates. Default 1e-8. An iteration that does not reach it within <code>max_iter</code> steps warns with class <code>hypergraphs_no_converge</code> .
<code>normalize</code>	Logical. If TRUE (default), each returned centrality vector is L2-normalized to unit norm (compatible with <code>igraph::eigen_centrality()</code> 's scale for type "clique"). Does not apply to "pagerank", which always sums to 1, or "subhypergraph", which is returned on its natural log scale.

damping	Single numeric in (0, 1). PageRank damping factor (probability of following the walk rather than teleporting). Default 0.85. Only used by type = "pagerank".
edge_weights	NULL, a single positive number (recycled to every hyperedge, e.g. 1 for unit weights), or a positive numeric vector, one per hyperedge. Hyperedge weights of the EDVW random walk behind type = "pagerank"; NULL defaults to the window counts for hypergraphs built by <code>window_hypergraph()</code> , else the Hayashi et al. dispersion heuristic (unit weights on a binary incidence). Only used by type = "pagerank".
alpha	Attenuation factor of type = "katz": a single number in $(0, 1/\lambda_{max})$, where λ_{max} is the largest eigenvalue of the hypergraph adjacency. There is no correct default; type = "katz" without alpha raises <code>hypergraphs_bad_input</code> , and so does an alpha at or above the bound (the message reports it).
...	Must be empty: an argument that only the memory-network method takes raises <code>hypergraphs_bad_input</code> .

Details

type also accepts "pagerank" (EDVW hypergraph PageRank), "subhypergraph" and "katz" (Katz centrality, which needs alpha); the result is one row per node.

Clique-motif eigenvector centrality (CEC): forms the clique-expanded pairwise graph W where $W_{ij} = |\{e : i, j \in e\}|$ and returns the leading eigenvector of W . Equivalent to running `igraph::eigen_centrality()` on `pairwise_network()` output.

Z-eigenvector centrality (ZEC): solves the linear eigen-equation on the hyperedge tensor,

$$\lambda x_i = \sum_{e \ni i} \prod_{j \in e, j \neq i} x_j,$$

via power iteration. Works for hypergraphs with mixed edge sizes.

H-eigenvector centrality (HEC): solves the power-k-1 eigen-equation,

$$\lambda x_i^{k-1} = \sum_{e \ni i} \prod_{j \in e, j \neq i} x_j.$$

For uniform hypergraphs (all hyperedges of size k), this is equivalent to normalizing the ZEC update by the geometric-mean exponent $1/(k-1)$. For mixed sizes, the effective exponent is taken from the largest hyperedge; expect slightly different rankings from ZEC in the mixed case.

Hypergraph PageRank ("pagerank"): the stationary distribution of the damped EDVW random walk of Chitra & Raphael (2019): from node v , pick a hyperedge $e \ni v$ with probability proportional to its weight $w(e)$, then a node $u \in e$ with probability proportional to its edge-dependent vertex weight $\gamma_e(u)$ (the incidence cell, i.e. occurrence totals for `window_hypergraph()`); with probability $1 - \text{damping}$ teleport uniformly. Their collapse theorem: with edge-independent vertex weights (a binary incidence) the walk is equivalent to PageRank on the weighted clique expansion with edge weights $\sum_{e \ni u, v} w(e)/\delta(e)$ – the hypergraph adds information exactly when γ is edge-dependent. Nodes left in no hyperedge (possible after `min_weight/min_size` filtering) teleport from every step and receive only teleportation mass. The undamped stationary distribution of the same walk is the `pi` column reported by `hg_cluster()`.

Subhypergraph centrality ("subhypergraph"): the logarithm of the diagonal of the matrix exponential of the clique adjacency derived from binary incidence (so entries count shared hyperedges),

$\log[\exp(W)]_{ii}$. It counts closed walks based at each node with a factorial penalty for length and matches the implementation used by HypergraphX 1.5 in the legal-hypergraphs analysis.

Katz centrality ("katz"): Katz's (1953) status index on the hypergraph adjacency of Estrada & Rodriguez-Velazquez (2006), A_{ij} = the number of hyperedges containing both i and j (the weighted clique expansion),

$$x = \sum_{k \geq 1} \alpha^k A^k \mathbf{1} = (I - \alpha A)^{-1} \alpha A \mathbf{1}, \quad 0 < \alpha < 1/\lambda_{\max}(A),$$

the α -discounted number of walks leaving each node (Battiston et al. 2020, Sec. III.B.3). Returned on this natural scale, not normalized; the alpha centrality of Bonacich & Lloyd (2001) with unit exogenous status, which also counts the length-0 walk, is $\text{katz} + 1$ (tested against `igraph::alpha_centrality()` on the clique expansion). XGI's `katz_centrality()` is the same vector for its fixed $\alpha = 2^{-n}$, rescaled to sum to one (tested). It is the only type that also runs on a sparse incidence.

Value

A base data.frame, one row per node (or the n requested rows), with one column per requested centrality.

References

- Benson, A. R. (2019). Three hypergraph eigenvector centralities. *SIAM Journal on Mathematics of Data Science* 1(2), 293-312. [arXiv:1807.09644](https://arxiv.org/abs/1807.09644).
- Chitra, U., & Raphael, B. J. (2019). Random walks on hypergraphs with edge-dependent vertex weights. *Proceedings of the 36th International Conference on Machine Learning*, PMLR 97, 1172-1181.
- Hayashi, K., Aksoy, S. G., Park, C. H., & Park, H. (2020). Hypergraph random walks, Laplacians, and clustering. *Proceedings of CIKM 2020*, 495-504. [doi:10.1145/3340531.3412034](https://doi.org/10.1145/3340531.3412034)
- Estrada, E., & Rodriguez-Velazquez, J. A. (2005). Complex networks as hypergraphs. *arXiv preprint physics/0505137*.
- Estrada, E., & Rodriguez-Velazquez, J. A. (2006). Subgraph centrality and clustering in complex hyper-networks. *Physica A*, 364, 581-594. [doi:10.1016/j.physa.2005.12.002](https://doi.org/10.1016/j.physa.2005.12.002)
- Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, 18(1), 39-43. [doi:10.1007/BF02289026](https://doi.org/10.1007/BF02289026)
- Bonacich, P., & Lloyd, P. (2001). Eigenvector-like measures of centrality for asymmetric relations. *Social Networks*, 23(3), 191-201. [doi:10.1016/S03788733\(01\)000387](https://doi.org/10.1016/S03788733(01)000387)
- Battiston, F., Cencetti, G., Iacopini, I., Latora, V., Lucas, M., Patania, A., Young, J.-G., & Petri, G. (2020). Networks beyond pairwise interactions: Structure and dynamics. *Physics Reports*, 874, 1-92. [doi:10.1016/j.physrep.2020.05.004](https://doi.org/10.1016/j.physrep.2020.05.004)

See Also

[network_hypergraph\(\)](#), [pairwise_network\(\)](#), [hg_measures\(\)](#).

Examples

```
hg <- text_hypergraph(c(
  a = "salt and soup and onions",
  b = "soup and salt",
  c = "stars and salt"
))
hg_centrality(hg, type = "clique")
hg_centrality(hg, type = c("clique", "katz"), alpha = 0.1)
```

hg_centrality.net_hon *Higher-order centralities with first-order projection*

Description

The memory-network method of `hg_centrality()`. Computes centralities on the higher-order topology of a `hon()` network and, by default, projects them back onto the first-order states, following Scholtes, Wider & Garas (2016). Because a higher-order node carries the memory of how a state was reached, these centralities can rank states differently from the same measures on the first-order transition network - and that difference is the evidence that memory matters for flow, not just for prediction.

Usage

```
## S3 method for class 'net_hon'
hg_centrality(
  x,
  type = c("pagerank", "betweenness", "closeness"),
  project = TRUE,
  projection = c("scaled", "last", "first", "all"),
  damping = 0.85,
  weighted = FALSE,
  max_iter = 1000L,
  tol = 1e-12,
  max_paths = 1e+06,
  sort_by = NULL,
  top = NULL,
  ...
)
```

Arguments

<code>x</code>	A <code>net_hon</code> object from <code>hon()</code> .
<code>type</code>	Character vector, any subset of <code>c("pagerank", "betweenness", "closeness")</code> . Default computes all three.
<code>project</code>	Logical. TRUE (default) returns one row per first-order state; FALSE returns one row per higher-order node with the ordinary directed-graph centralities of the higher-order topology (useful to see which <i>contexts</i> , not states, dominate).

projection	How a higher-order node's PageRank is distributed over the states of its path when <code>project = TRUE</code> : "scaled" (default, pathpy's default - each of the k states receives a k -th of the value, so the projected PageRank still sums to 1), "last" (all of it to the state actually occupied), "first", or "all" (every state on the path receives the full value; the result no longer sums to 1). Ignored by betweenness and closeness, which are first-order by construction.
damping	PageRank damping factor in $(0, 1)$. Default 0.85.
weighted	Logical. FALSE (default, pathpy's convention) runs PageRank on the binary higher-order topology; TRUE uses the edge weights of <code>hon\$matrix</code> .
max_iter, tol	Power-iteration controls for PageRank.
max_paths	Integer cap on the shortest paths enumerated for betweenness. Exceeding it raises a <code>hypergraphs_too_many_paths</code> error rather than silently truncating. Default $1e6$.
sort_by	NULL (alphabetical by state or node, the default) or one of the requested types - sort the table by that centrality, largest first, with ties broken by state/node so the order stays deterministic.
top	Integer or NULL. Return only the first top rows, applied after any filter and after <code>sort_by</code> , so <code>sort_by</code> and <code>top</code> compose. Default NULL returns every row.
...	Must be empty: an argument that only the hypergraph method takes raises <code>hypergraphs_bad_input</code> .

Details

Semantics match pathpy 2.2.0 (the reference implementation by the method's author), generalized from fixed-order to the variable-order networks `hon()` produces: a higher-order node's order is the number of states in the path it represents, and the first-order distance implied by a higher-order hop count adds that node's order minus one. With a uniform order the generalization reduces exactly to pathpy's formulas.

Value

A `data.frame`, one row per first-order state (or per higher-order node when `project = FALSE`), with the requested centralities as columns:

state First-order state (when `project = TRUE`).

node, order Higher-order node and its order (when `project = FALSE`).

pagerank Projected PageRank of the higher-order topology.

betweenness Share of first-order shortest paths (implied by the higher-order topology) passing through the state.

closeness Harmonic closeness on the implied first-order distances: the sum of inverse distances *into* the state.

Rows are ordered by state (or by node) so the result is deterministic.

References

- Scholtes, I., Wider, N., & Garas, A. (2016). Higher-order aggregate networks in the analysis of temporal networks: path structures and centralities. *The European Physical Journal B* 89, 61. doi:10.1140/epjb/e2016606630
- Xu, J., Wickramaratne, T. L., & Chawla, N. V. (2016). Representing higher-order dependencies in networks. *Science Advances* 2(5), e1600028. doi:10.1126/sciadv.1600028

See Also

`hon()`, `hg_bootstrap()`, `memory()`

Examples

```
seqs <- c(
  replicate(6, rep(c("a", "b", "c"), 4), simplify = FALSE),
  replicate(6, rep(c("x", "b", "d"), 4), simplify = FALSE)
)
hon <- hon(seqs, max_order = 2)
hg_centrality(hon)

# Which contexts, rather than which states, carry the flow?
hg_centrality(hon, type = "pagerank", project = FALSE)
```

hg_classify

Transductive label spreading on a hypergraph

Description

Semi-supervised classification of hypergraph nodes by the regularization framework of Zhou et al. (2006): given labels for a subset of nodes, the scores $F = (I - \xi) * (I - \xi * S)^{-1} Y$ spread the labels over the hypergraph, where $S = I - L$ is the normalized similarity operator of the chosen Laplacian and Y is the label indicator matrix. Each node is assigned the class with the highest score. This is the non-neural ancestor of hypergraph-attention text classifiers: with documents as hyperedges over words (or vice versa) it classifies unlabeled nodes from a handful of labeled ones.

Usage

```
hg_classify(
  hg,
  labels,
  xi = 0.99,
  type = c("zhou", "random_walk"),
  normalization = c("none", "class_mass"),
  edge_weights = NULL,
  holdout = NULL,
  seed = 1L
)
```

Arguments

hg	A <code>text_hypergraph()</code> (or any hypergraphs <code>net_hg</code>).
labels	The known labels: a named character vector (names are node identifiers – documents under <code>nodes = "doc"</code> – values their class labels), a tidy <code>data.frame</code> with a node column and a <code>label</code> , <code>cluster</code> or <code>predicted</code> column, or the name of a column of the hypergraph's document table, which a <code>text_hypergraph()</code> fills with the input's other columns (<code>labels = "period"</code>).
xi	Numeric in $(0, 1)$. Spreading coefficient (default 0.99); larger values weight the hypergraph structure more relative to the initial labels.
type, edge_weights	Passed to <code>hg_laplacian()</code> .
normalization	Decision rule for turning spread scores into predictions: "none" (default, the raw Zhou 2006 argmax) or "class_mass" (class-mass normalization, Zhu et al. 2003). Use "class_mass" when the labeled seeds are class-imbalanced – the raw rule can collapse every prediction onto the majority class.
holdout	NULL (default) uses every given label and returns the predictions. A share in $(0, 1)$ hides that share of the labels, drawn within each class, predicts them from the rest, and returns an <code>hg_classification</code> that prints the held-out accuracy and balanced accuracy.
seed	Seed of the held-out draw (default 1); the caller's random-number stream is restored.

Details

The result is one row per node with its given and predicted label.

Value

A base `data.frame`, one row per node, with columns `node`, `label` (the given label or NA), `predicted`, `score`, and `margin`. With `holdout`, an `hg_classification`: the same table with `label` holding the true label, `split` ("train" or "test") and correct for the held-out documents; read its evaluation with `hg_get()` (what = "accuracy", "classes", "confusion").

References

- Zhou, D., Huang, J., & Scholkopf, B. (2006). Learning with hypergraphs: Clustering, classification, and embedding. *NeurIPS 19*. doi:10.7551/mitpress/7503.003.0205
- Zhu, X., Ghahramani, Z., & Lafferty, J. (2003). Semi-supervised learning using Gaussian fields and harmonic functions. *ICML 20*.

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots",
  cooking_2 = "this soup recipe needs salt on a cold night",
  space_1 = "the telescope revealed a distant galaxy and stars",
  space_2 = "astronomers aimed the telescope at the stars all night"
```

```
), stop_words = c("the", "with", "and", "a", "this", "at", "on", "all"))
hg_classify(hg, labels = c(cooking_1 = "cooking", space_1 = "space"))
```

hg_cluster

Spectral or symmetric-NMF clustering of hypergraph vertices

Description

Partitions the nodes of a hypergraph into k clusters with either of Hayashi et al.'s (2020) representative-digraph algorithms. `algorithm = "spectral"` (RDC-Spec) row-normalizes the k smallest Laplacian eigenvectors and applies k -means. `algorithm = "symnmf"` (RDC-Sym) computes a rank- k non-negative factorization $T \approx U U'$ of the normalized similarity $T = I - L$, then assigns each vertex to the largest entry in its row of U , exactly as Algorithm 2 specifies. T is dense whatever the incidence, so on a sparse hypergraph it is built from the sparse Laplacian and factorised dense ($8 n^2$ bytes: 350 MB for 6,630 documents); beyond 15,000 nodes this raises `hypergraphs_sparse_too_large`. With `type = "random_walk"` and a weighted incidence (e.g. from `group_hypergraph()` with `weight =`), the edge-dependent vertex weights genuinely change the partition - with edge-independent weights the walk collapses to a graph random walk (Chitra & Raphael 2019).

Usage

```
hg_cluster(
  hg,
  k,
  type = c("zhou", "random_walk"),
  edge_weights = NULL,
  seed = NULL,
  nstart = 25L,
  what = c("clusters", "embedding", "eigenvalues", "membership"),
  n = Inf,
  algorithm = c("spectral", "symnmf"),
  max_iter = 500L,
  tol = 1e-06,
  parallel = FALSE,
  n_cores = 2L
)
```

Arguments

hg	A <code>text_hypergraph()</code> (or any hypergraphs <code>net_hg</code>).
k	Number of clusters (explicit by design; there is no correct default).
type	The Laplacian, as in <code>hg_laplacian()</code> : "zhou" (Zhou et al. 2006) or "random_walk" (Hayashi et al. 2020).
edge_weights	Hyperedge weights for the Laplacian: NULL (default, every hyperedge counts once), a positive numeric vector with one entry per hyperedge, or "idf" to weight each word hyperedge by its smoothed inverse document frequency (needs

	weight = "tfidf"), so a word used in most documents binds them less than a rare shared word. Note that the default type = "zhou" reads only which documents a hyperedge binds; the incidence cells (weight =) enter the "random_walk" type only.
seed	Random seed passed to the selected solver; set it for a reproducible partition.
nstart	Number of solver starts (default 25L).
what	What to return: "clusters" (default) for the partition, "embedding" for the partition plus the row-normalized spectral embedding used by k-means (dim1..dimk – plot these to map the corpus) and the stationary weight pi, or "eigenvalues" for the Laplacian spectrum (dense engines return all n values; the sparse engine returns the k + 1 it computed – raise k for an eigengap scan), or, with algorithm = "symnmf", "membership" for the graded membership of every node in every cluster: the node's row of the non-negative factor, normalised to sum to one (Kuang, Ding & Park 2012).
n	For what = "eigenvalues", how many rows to keep, leading first (default Inf, all of them). The spectrum carries one eigenvalue per node, so a corpus of a few thousand documents returns a few thousand rows, and only the leading ones carry the gap that decides how many groups the structure supports. Ignored for the other values of what.
algorithm	Character. "spectral" (default; RDC-Spec) or "symnmf" (RDC-Sym).
max_iter	Maximum multiplicative-update iterations for algorithm = "symnmf".
tol	Relative objective tolerance for algorithm = "symnmf".
parallel	For algorithm = "symnmf": run the nstart starts with parallel::mclapply() (not on Windows, where they run serially). Default FALSE. Every start's initial factor is drawn first, in order, so the result equals the serial one. Supplying it (or n_cores) with algorithm = "spectral" raises hypergraphs_bad_input.
n_cores	Cores when parallel = TRUE (default 2).

Details

Both solvers are stochastic: nstart initializations are used and a seed fixes the result. Report stability across seeds for consequential results.

type = "random_walk" (Hayashi et al. 2020) is the natural choice for tf-idf-weighted text hypergraphs. what = returns the cluster table, the embedding or the leading eigenvalues as a tidy data.frame.

Value

A base data.frame. For what = "clusters": one row per node, columns node and cluster. For what = "embedding": node, cluster, pi, dim1..dimk. For what = "eigenvalues": one row per eigenvalue, columns index, value (ascending) and gap (the distance to the next eigenvalue – large gaps indicate supported cluster counts; NA on the last row). For what = "membership": one row per node and cluster, columns node, cluster and membership (summing to one over a node's clusters); a node's largest membership is its cluster in what = "clusters".

References

- Kuang, D., Ding, C., & Park, H. (2012). Symmetric nonnegative matrix factorization for graph clustering. *Proceedings of the 2012 SIAM International Conference on Data Mining*, 106-117. doi:10.1137/1.9781611972825.10
- Hayashi, K., Aksoy, S. G., Park, C. H., & Park, H. (2020). Hypergraph random walks, Laplacians, and clustering. *CIKM 2020*, 495-504. doi:10.1145/3340531.3412034
- Chitra, U., & Raphael, B. J. (2019). Random walks on hypergraphs with edge-dependent vertex weights. *ICML 2019*.

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots",
  cooking_2 = "this soup recipe needs salt on a cold night",
  space_1 = "the telescope revealed a distant galaxy and stars",
  space_2 = "astronomers aimed the telescope at the stars all night"
), stop_words = c("the", "with", "and", "a", "this", "at", "on", "all"))
hg_cluster(hg, k = 2, type = "random_walk", seed = 1)
hg_cluster(hg, k = 2, seed = 1, what = "embedding")
hg_cluster(hg, k = 2, seed = 1, what = "eigenvalues", n = 5)
```

hg_coccluster

Co-clustering of nodes and hyperedges (documents and words)

Description

Clusters the nodes and the hyperedges of a hypergraph together, so that each cluster is a pair: a set of documents and the set of words that characterises them, from one spectral cut of the document-word bipartite graph (Dhillon 2001). With A the incidence matrix (nodes by hyperedges, the stored weights: counts or tf-idf on a `text_hypergraph()`), D_1 and D_2 its row and column sums, the scaled matrix $A_n = D_1^{-1/2} A D_2^{-1/2}$ is decomposed by SVD; its leading singular pair is trivial, the next $\ell = \lceil \log_2 k \rceil$ left and right singular vectors U, V give the embedding $Z = [D_1^{-1/2} U; D_2^{-1/2} V]$, one row per node and per hyperedge, and k-means on the rows of Z gives the k co-clusters (Dhillon 2001, Algorithm Multipartition).

Usage

```
hg_coccluster(
  hg,
  k,
  seed = NULL,
  nstart = 25L,
  what = c("clusters", "embedding"),
  role = c("all", "node", "hyperedge")
)
```

Arguments

hg	A <code>text_hypergraph()</code> (or any hypergraphs <code>net_hg</code>) with no empty node or hyperedge.
k	Number of co-clusters, at least 2.
seed	Seed for the k-means starts; the caller's random number stream is restored on exit. NULL (default) uses the current stream.
nstart	Number of k-means starts (default 25L).
what	"clusters" (default) for the assignment, or "embedding" for the assignment plus the embedding coordinates <code>dim1..dimL</code> .
role	Which rows to return: "all" (default), "node" (on a document hypergraph, the documents) or "hyperedge" (the words).

Details

For a binary incidence with unit hyperedge weights, the node half of the embedding spans the same space as the Zhou et al. (2006) hypergraph Laplacian eigenvectors that `hg_cluster()` cuts (the squared singular values are one minus the Laplacian eigenvalues), so co-clustering reads the same cut on both sides of the bipartite graph; with other weights the two differ, because the Zhou Laplacian reads only the binary pattern. A cluster may hold nodes and no hyperedges, or the reverse.

Value

A base `data.frame`, one row per node and then per hyperedge (filtered by role), with columns `node` (the node or hyperedge name), `role` ("node" or "hyperedge") and `cluster` ("Cluster 1", ..., numbered by first appearance in that row order); with `what = "embedding"` also `dim1..dimL`. Raises `hypergraphs_bad_input` for a `k` that is not a whole number from 2 to the number of rows, a `k` whose $\ell + 1$ singular pairs the incidence does not have ($\ell + 1 > \min(m, n)$ for an $m \times n$ incidence, so $k \leq 2^{\min(m, n)-1}$), an embedding with fewer distinct rows than `k`, or an empty node or hyperedge (zero degree, where the scaling is undefined).

References

- Dhillon, I. S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 269–274. doi:10.1145/502512.502550
- Zhou, D., Huang, J., & Schölkopf, B. (2006). Learning with hypergraphs: Clustering, classification, and embedding. *Advances in Neural Information Processing Systems 19*, 1601–1608. doi:10.7551/mitpress/7503.003.0205

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots",
  cooking_2 = "this soup recipe needs salt on a cold night",
  space_1 = "the telescope revealed a distant galaxy and stars",
  space_2 = "astronomers aimed the telescope at the stars all night"
), stop_words = c("the", "with", "and", "a", "this", "at", "on", "all"))
hg_cocluster(hg, k = 2, seed = 1)
```

```
hg_coclust(hg, k = 2, seed = 1, role = "hyperedge")
```

hg_communities	<i>Communities of a hypergraph or a memory network</i>
----------------	--

Description

One verb, two estimators, chosen by the class of x:

a hypergraph (net_hg) an ensemble of Infomap runs on a projection (or IRMM with type = "irmm"), with the adjusted-mutual- information medoid; see [hg_communities.net_hg\(\)](#).

a memory network (net_hon, from [hon\(\)](#)) the map equation for memory networks: overlapping modules of the physical states, compared with the first-order map; see [hg_communities.net_hon\(\)](#).

Each method keeps its own arguments; passing an argument that only the other method takes raises hypergraphs_bad_input.

Usage

```
hg_communities(x, ...)
```

```
## Default S3 method:  
hg_communities(x, ...)
```

Arguments

x	A net_hg or a net_hon.
...	Arguments of the method for class(x).

Value

An hg_communities object (hypergraph) or a hypergraphs_memory_communities object (memory network); read either with [hg_get\(\)](#). Any other input raises hypergraphs_bad_input.

Examples

```
seqs <- list(c("a", "h", "b", "a", "h", "b", "a"),  
            c("c", "h", "d", "c", "h", "d", "c"))  
comm <- hg_communities(hon(seqs, max_order = 2L), trials = 2L)  
hg_get(comm)
```

hg_communities.net_hg *Stable Infomap communities of a hypergraph projection*

Description

The hypergraph method of `hg_communities()`. Builds the normalized association graph of Coupette et al. (2024), runs Infomap repeatedly, compares every pair of partitions with AMI, ARI and NMI, and returns the run with the largest summed AMI as the medoid. The paper uses 50 seeds and 100 Infomap trials per seed, which are the defaults.

Usage

```
## S3 method for class 'net_hg'
hg_communities(
  x,
  n_runs = 50L,
  trials = 100L,
  seeds = NULL,
  method = c("association", "citation"),
  duplicate_edges = c("count", "collapse"),
  self_association = FALSE,
  edge_source = NULL,
  directed = FALSE,
  type = c("infomap", "irmm"),
  delta = 0.01,
  max_iter = 50L,
  edge_weights = NULL,
  parallel = FALSE,
  n_cores = 2L,
  ...
)
```

Arguments

<code>x</code>	A static <code>net_hg</code> .
<code>n_runs</code>	Number of independent seeded Infomap runs (default 50).
<code>trials</code>	Infomap trials within each run (default 100).
<code>seeds</code>	Integer seeds. NULL uses <code>seq_len(n_runs)</code> .
<code>method</code>	Which graph Infomap runs on: the "association" projection (default; the paper's hypergraph-derived representations <code>bh</code> , <code>bhs</code> , <code>mh</code> , <code>mhs</code>) or the "citation" projection (its classic graph representations <code>bg</code> , <code>mg</code> , and with <code>directed = FALSE</code> their undirected variants <code>bgu</code> , <code>mg</code>). See <code>pairwise_network()</code> .
<code>duplicate_edges</code> , <code>self_association</code> , <code>edge_source</code>	Projection controls passed to <code>pairwise_network()</code> . Together these reproduce the paper's binary/multi and self-association representations.

directed	For method = "citation": run Infomap with directed flow on the source-to-member graph? Default FALSE.
type	The community algorithm: "infomap" (default; Infomap on the projection chosen by method, as in Coupette et al. 2024) or "irmm" (iteratively reweighted modularity maximisation, Kumar et al. 2020: Louvain on the random-walk clique reduction $A = HW(D_e - I)^{-1}H^T$, then every hyperedge is reweighted to $w'(e) = \frac{1}{m} \sum_{i=1}^c \frac{\delta(e)+c}{k_i(e)+1}$ and averaged with its previous weight, until the largest weight change is at most delta). IRMM reads the hypergraph directly, so trials, method, duplicate_edges, self_association, edge_source and directed do not apply to it and raise hypergraphs_bad_input when supplied. Each of the n_runs runs is one full IRMM fit whose Louvain steps draw from R's RNG under that run's seed (the caller's RNG state is restored); igraph is required.
delta	For type = "irmm": stop when no hyperedge weight changes by more than delta in a pass (default 0.01, the paper's threshold).
max_iter	For type = "irmm": maximum reweighting passes (default 50). A run that hits it is flagged converged = FALSE in the "runs" table and raises one hypergraphs_no_converge warning.
edge_weights	For type = "irmm": initial positive hyperedge weights (one per hyperedge, or one value recycled). NULL uses the window counts of a window_hypergraph() , else unit weights.
parallel	Logical. Run the n_runs independent runs with parallel::mclapply (not on Windows, where they run serially). Default FALSE. Every run is seeded by its own entry of seeds, so the result is identical to the serial one.
n_cores	Integer. Cores when parallel = TRUE (default 2).
...	Must be empty: an argument that only the memory-network method takes raises hypergraphs_bad_input.

Value

An hg_communities object containing medoid (a tidy node/community table), all partitions, AMI/ARI/NMI similarity matrices, run metadata, community sizes, and the graph projection Infomap ran on. For type = "irmm" the "runs" table has run, seed, n_communities, iterations, converged, max_weight_change and modularity (the linear hypergraph modularity of [hg_modularity\(\)](#)), projection is the reweighted clique reduction of the medoid run, and hg_get(fit, what = "weights") gives one row per hyperedge with edge, size, initial_weight and the medoid run's final weight.

References

- Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. doi:10.1098/rsta.2023.0141
- Kumar, T., Vaidyanathan, S., Ananthapadmanabhan, H., Parthasarathy, S., & Ravindran, B. (2020). Hypergraph clustering by iteratively reweighted modularity maximization. *Applied Network Science*, 5, 52. doi:10.1007/s41109020003003
- Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics*, 2008(10), P10008. doi:10.1088/17425468/2008/10/P10008

See Also

[hg_modularity\(\)](#) to score any partition.

Examples

```
dat <- data.frame(
  member = c("a", "b", "c", "a", "b", "c", "x", "y", "z", "x", "y", "z"),
  edge = rep(paste0("e", 1:4), each = 3)
)
h <- group_hypergraph(dat, "member", "edge")
if (requireNamespace("igraph", quietly = TRUE)) {
  fit <- hg_communities(h, n_runs = 2, trials = 2, seeds = 1:2)
  hg_get(fit)
  irmm <- hg_communities(h, type = "irmm", n_runs = 3, seeds = 1:3)
  hg_get(irmm)
  hg_get(irmm, what = "runs")
}
```

hg_communities.net_hon

Memory-network communities with the map equation

Description

Finds modules in a higher-order network by minimising the map equation for memory networks (Rosvall et al. 2014; Edler, Bohlin & Rosvall 2017). The state nodes of the HON ("a → b" is physical node b reached from a) are clustered, but each module's codebook is written over **physical** nodes: state nodes of one physical node that land in the same module share a codeword. This is not first-order Infomap run on the state graph, and a physical node whose state nodes fall in different modules belongs to all of them, so modules of physical nodes can overlap.

The same search is run on the first-order network obtained by projecting the memory network's link flow onto physical nodes, so the result says whether memory compresses the flow better than a first-order description.

Usage

```
## S3 method for class 'net_hon'
hg_communities(
  x,
  partition = NULL,
  trials = 10L,
  teleportation = 0.15,
  seed = 1L,
  ...
)
```

Arguments

x	A net_hon from hon() .
partition	NULL (default) to search, or a given state-node partition to evaluate without searching: a named vector (names = state labels as in <code>hg_get(hon, what = "nodes")</code>) or a data.frame with columns node and community.
trials	Number of independent search trials (default 10). Ignored when partition is given, except for the first-order search.
teleportation	Teleportation probability $\tau \in [0, 1]$ (default 0.15, Infomap's default).
seed	Integer seed for the random node orders (default 1). The caller's random-number stream is restored on exit.
...	Must be empty: an argument that only the hypergraph method takes raises <code>hypergraphs_bad_input</code> .

Details

Flow. State node visit rates are the stationary distribution of the walk on the HON transition matrix $P_{\alpha\beta} = w_{\alpha\beta}/w_\alpha$ (Edler et al. 2017, Eq. 8-9). With probability teleportation (τ), and always from a state with no out-links, the walker restarts at the tail of a link drawn in proportion to its weight (teleportation to links; Lambiotte & Rosvall 2012). Teleportation is unrecorded: link flow is $f_{\alpha\beta} = \pi_\alpha P_{\alpha\beta}$ renormalised to sum to one, and a state's flow is its link in-flow. This is Infomap's directed flow model; node flows and codelengths match the Python infomap package (2.15.1). `teleportation = 0` uses the plain stationary distribution and needs an irreducible chain.

Codelength (Edler et al. 2017, Eq. 24-26). With module enter flow q_m^{in} , exit flow q_m^{out} , and physical flow inside a module $\pi_{i \cap m} = \sum_{\beta_i \in m} \pi_{\beta_i}$,

$$L(M) = q^{in} H(Q) + \sum_m p_m H(P_m),$$

where $q^{in} = \sum_m q_m^{in}$, $H(Q)$ is the entropy of the enter rates q_m^{in}/q^{in} , $p_m = q_m^{out} + \sum_i \pi_{i \cap m}$, and $H(P_m)$ is the entropy of $\{q_m^{out}, \pi_{i \cap m}\}$ normalised by p_m . The one-module codelength is the entropy of the physical-node flow.

Search (Edler et al. 2017, Algorithms 2-5): every state node starts in its own module; in random sequential order each node moves to the neighbouring or new module that most decreases L ; modules are then aggregated into nodes and the moves repeat, until L stops decreasing (repeated node aggregation, the Louvain machinery). The result is fine-tuned by restarting the aggregation from single state nodes placed in their current modules. Coarse-tuning (Algorithm 6) is not implemented. `trials` independent runs are made from one seeded random stream and the shortest codelength is kept; `hg_get(x, what = "trials")` reports every run and its agreement (adjusted Rand index) with the kept one.

First-order comparison. The link flow of the memory network is summed over state nodes into a physical-node network, $w_{ij} = \sum_{\alpha_i, \beta_j} f_{\alpha\beta}$, and searched with the same flow model and the same number of trials. Rosvall et al. (2014) compare memory and first-order maps of the same pathway data this way.

Value

A `hypergraphs_memory_communities` object. Read it with `hg_get()`: `what = "states"` (one row per state node), `"physical"` (one row per physical node x module), `"modules"`, `"trials"`, `"first_order"` and `"codelength"`; see `hg_get.hypergraphs_memory_communities()`.

Conditions

`hypergraphs_bad_input` for a non-net_hon input, invalid arguments, or a partition that does not cover every state; `hypergraphs_not_ergodic` when `teleportation = 0` and the walk has no unique stationary flow.

References

- Rosvall, M., Esquivel, A. V., Lancichinetti, A., West, J. D., & Lambiotte, R. (2014). Memory in network flows and its effects on spreading dynamics and community detection. *Nature Communications*, 5, 4630. doi:10.1038/ncomms5630
- Edler, D., Bohlin, L., & Rosvall, M. (2017). Mapping higher-order network flows in memory and multilayer networks with Infomap. *Algorithms*, 10(4), 112. doi:10.3390/a10040112
- Rosvall, M., & Bergstrom, C. T. (2008). Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4), 1118-1123. doi:10.1073/pnas.0706851105
- Lambiotte, R., & Rosvall, M. (2012). Ranking and clustering of nodes in networks with smart teleportation. *Physical Review E*, 85(5), 056107. doi:10.1103/PhysRevE.85.056107
- Hubert, L., & Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2(1), 193-218. doi:10.1007/BF01908075

Examples

```
seqs <- list(c("a", "h", "b", "a", "h", "b", "a"),
             c("c", "h", "d", "c", "h", "d", "c"),
             c("a", "h", "b", "a", "h", "b"),
             c("c", "h", "d", "c", "h", "d"))
hon <- hon(seqs, max_order = 2L)
comm <- hg_communities(hon, trials = 3L)
comm
summary(comm)
hg_get(comm)
hg_get(comm, what = "physical")
```

hg_community_quality *Quality of a projected-hypergraph partition*

Description

Reports graph-partition diagnostics on the association projection: unweighted coverage, weighted coverage, performance, weighted modularity, and conductance. Conductance is the maximum (worst) community conductance $cut(S, \bar{S}) / \min(vol(S), vol(\bar{S}))$; lower is better.

Usage

```
hg_community_quality(  
  hg,  
  partition,  
  method = c("association", "citation"),  
  duplicate_edges = c("count", "collapse"),  
  self_association = FALSE,  
  edge_source = NULL  
)
```

Arguments

hg	A static net_hg.
partition	An <code>hg_communities()</code> result, a tidy node/label table, or a named label vector. Every projected node needs one non-missing label; a node repeated with different labels raises <code>hypergraphs_bad_input</code> .
method	The projection the partition is scored on: the "association" graph (default) or the undirected "citation" graph.
duplicate_edges, self_association, edge_source	Projection controls passed to <code>pairwise_network()</code> . Together these reproduce the paper's binary/multi and self-association representations.

Value

A one-row data frame.

References

Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486(3-5), 75-174. doi:10.1016/j.physrep.2009.11.002

Newman, M. E. J., & Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review E*, 69, 026113. doi:10.1103/PhysRevE.69.026113

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382, 20230141. doi:10.1098/rsta.2023.0141

hg_compare	<i>Permutation comparison of the rules of two groups</i>
------------	--

Description

Tests whether two groups of sequences differ in their higher-order rule probabilities. The rule set is extracted from the pooled data of the two groups; for every pooled rule the statistic is the absolute difference of the two groups' conditional probabilities, and its null distribution comes from permuting the group labels over sequences. Per-rule p-values are Benjamini-Hochberg adjusted; a global test aggregates the rule differences weighted by pooled counts.

Usage

```
hg_compare(
  x,
  groups = NULL,
  n_perm = 1000L,
  alpha = 0.05,
  parallel = FALSE,
  n_cores = 2L,
  seed = NULL
)
```

Arguments

<code>x</code>	A group model from hon() with group. The estimation settings (<code>max_order</code> , <code>min_freq</code> , <code>collapse_repeats</code>) and the data of each group are taken from it.
<code>groups</code>	The two groups to compare, as a character vector of length 2. May be omitted when the model has exactly two groups.
<code>n_perm</code>	Integer ≥ 2 . Label permutations. Default 1000.
<code>alpha</code>	Significance level for the significant flag on the adjusted p-values. Default 0.05.
<code>parallel</code> , <code>n_cores</code> , <code>seed</code>	As in hg_bootstrap() .

Details

A rule whose context one group never observes has no difference (`diff` is NA) and is not counted as zero. The global statistic is the pooled-count-weighted mean absolute difference over the comparable rules, those whose context both groups observe; in a permutation, a comparable rule whose context one permuted group lacks is left out of that permutation's mean. When no rule is comparable (the groups share no context), the statistic and its p-value are NA and the warning `hypergraphs_undefined_statistic` is raised.

Per-sequence counts are precomputed once and every permutation is a weighted aggregation; permutations are drawn before any parallel work, so `parallel = TRUE` reproduces the serial result under the same seed. Rules whose context is unobserved in a group under some permutation contribute only their valid permutations (`n_perm_used`).

Value

An object of class `hypergraphs_comparison`: a list with edges (one row per pooled rule: `from`, `to`, `order`, `count`, the two groups' counts and probabilities (columns named after the groups), `diff` (probability difference, first minus second), `p_value`, `p_adj` (BH), `significant`, `n_perm_used`), `global` (`statistic`, the pooled-count-weighted mean absolute difference over the comparable rules, `p_value`, `n_comparable` and `n_rules`, the comparable and pooled rules, and `n_perm_used`, the permutations with a defined statistic), `names`, `n_perm`, `alpha`, `max_order`, `min_freq`, `n_trajectories` (per group) and `seed`. Has `print`, `summary` and `plot` methods; [hg_get\(\)](#) returns the rule table (`significant = TRUE` restricts it).

References

- Xu, J., Wickramaratne, T. L., & Chawla, N. V. (2016). Representing higher-order dependencies in networks. *Science Advances* 2(5), e1600028. doi:10.1126/sciadv.1600028
- Good, P. (2005). *Permutation, Parametric and Bootstrap Tests of Hypotheses* (3rd ed.). Springer.
- Benjamini, Y., & Hochberg, Y. (1995). Controlling the false discovery rate. *Journal of the Royal Statistical Society B*, 57(1), 289-300. doi:10.1111/j.25176161.1995.tb02031.x

See Also

[hon\(\)](#), [hg_bootstrap\(\)](#), [markov_order\(\)](#)

Examples

```
set.seed(1)
first_order <- replicate(6, sample(c("a", "b", "c"), 12, replace = TRUE),
                          simplify = FALSE)
second_order <- replicate(6, rep(c("a", "b", "c", "b"), 3),
                          simplify = FALSE)
by_group <- hon(c(first_order, second_order),
               group = rep(c("first", "second"), each = 6), max_order = 2)
comparison <- hg_compare(by_group, n_perm = 99, seed = 1)
comparison
hg_get(comparison, sort_by = "p_adj", top = 6)
```

hg_compare_communities

Compare community structure across representations

Description

Sets several [hg_communities\(\)](#) fits of the same data side by side, as the paper does for its eight GFCC representations (Coupette et al. 2024, Figure 8): the cluster-size distribution of each AMI medoid, the pairwise AMI, ARI and NMI between medoids, and per-fit summaries (number of communities, singletons, the balance between the two largest clusters).

Usage

```
hg_compare_communities(..., hg = NULL, edge_source = NULL)

## S3 method for class 'hypergraphs_community_comparison'
print(x, n = 10L, ...)

## S3 method for class 'hypergraphs_community_comparison'
hg_get(x, what = c("summary", "similarity", "sizes", "quality"), ...)

## S3 method for class 'hypergraphs_community_comparison'
plot(x, what = c("sizes", "similarity"), ...)
```

Arguments

...	For plot, unused.
hg	Optional: the static net_hg the fits were computed on. When given, every medoid is scored with the measures of <code>hg_community_quality()</code> on the projection its own fit saved (for type = "irmm", the final reweighted clique reduction), and the scores are available as what = "quality". A fit run on a directed citation graph cannot be scored (the measures are defined on undirected graphs): its quality row is NA and a <code>hypergraphs_undefined_statistic</code> warning is raised.
edge_source	Deprecated and unused: the saved projections already carry the hyperedge sources. Supplying it warns with class <code>hypergraphs_deprecated</code> .
x	A <code>hypergraphs_community_comparison</code> object.
n	Number of rows of the default table to print. Default 10.
what	Which table: "summary" (default), "similarity", "sizes" or "quality".

Value

A `hypergraphs_community_comparison` object. `hg_get()` returns its "summary" (default; one row per fit with model, medoid_seed, n_communities, n_singletons, n_nontrivial, largest_size, second_size and balance = second / largest), "similarity" (one row per pair of fits with model_a, model_b, n_nodes, ami, ari, nmi, on the n_nodes nodes the two medoids share; with fewer than two shared nodes the three scores are NA and a `hypergraphs_undefined_statistic` warning is raised), "sizes" (one row per community of every medoid with model, rank, n_nodes) or, when hg was given, "quality" (one row per fit with the columns of `hg_community_quality()`). `plot()` draws the cluster-size distributions (what = "sizes", the number of communities at least as large as each size, on logarithmic axes) or the similarity matrix (what = "similarity", AMI below and ARI above the diagonal, values printed in the cells, on a diverging scale from -1 through 0 (white) to 1, since chance-corrected agreement can be negative; undefined cells are grey).

For plot, a ggplot object.

References

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. doi:10.1098/rsta.2023.0141

Examples

```
dat <- data.frame(
  member = c("a", "b", "c", "a", "b", "c", "x", "y", "z", "x", "y", "z"),
  edge = rep(paste0("e", 1:4), each = 3)
)
h <- group_hypergraph(dat, "member", "edge")
if (requireNamespace("igraph", quietly = TRUE)) {
  multi <- hg_communities(h, n_runs = 2, trials = 2, seeds = 1:2)
  binary <- hg_communities(h, n_runs = 2, trials = 2, seeds = 1:2,
    duplicate_edges = "collapse")
  comparison <- hg_compare_communities(mh = multi, bh = binary)
  hg_get(comparison)
  hg_get(comparison, what = "similarity")
}
```

```
}
```

hg_degree	<i>Simplicial degree of every node</i>
-----------	--

Description

How many simplices of each dimension contain each node.

Usage

```
hg_degree(sc, normalized = FALSE)
```

Arguments

sc	A simplicial_complex from <code>simplicial()</code> .
normalized	Logical. Divide each count by its largest possible value. Default FALSE.

Value

A base data.frame, one row per node, most connected first: node, one column per dimension (d0, d1, ...), and total (the simplices of dimension one and above). The same table is `hg_get(sc, what = "degree")`.

References

Serrano, D. H., & Gomez, D. S. (2020). Centrality measures in simplicial complexes: Applications of topological data analysis to network science. *Applied Mathematics and Computation*, 382, 125331. doi:10.1016/j.amc.2020.125331

See Also

`simplicial()`

Examples

```
adj <- matrix(c(0, 1, 1, 0,
               1, 0, 1, 1,
               1, 1, 0, 1,
               0, 1, 1, 0), 4, 4, dimnames = list(letters[1:4], letters[1:4]))
hg_degree(simplicial(adj))
```

hg_degree_correlation *Correlation of hyperdegrees across interaction orders*

Description

For every pair of hyperedge sizes $s < t$ present in the hypergraph, the Pearson correlation across nodes between the size- s hyperdegree (the number of size- s hyperedges containing the node) and the size- t hyperdegree, as reported by Lotito et al. (2023) for the high-school contact hypergraph ($\rho_{2,3}$, $\rho_{2,4}$, $\rho_{3,4}$). A high value means that nodes active in pairs are also active in larger groups.

Usage

```
hg_degree_correlation(hg)
```

Arguments

hg A hypergraphs net_hg, dense or sparse.

Details

Every node of hg enters every correlation (with degree 0 at the sizes it does not take part in). Sizes below two are ignored, as in hypergraphx. A pair is NA when one of the two sequences is constant. hypergraphx 1.8.0 degree_correlation() returns the full (max_size - 1)-square matrix, including sizes with no hyperedge (all NaN); hypergraphs returns one row per pair of sizes that occur. Tested against hypergraphx at 1e-12.

Value

A base data.frame, one row per pair of present hyperedge sizes size_1 < size_2, with columns size_1, size_2, correlation (Pearson, NA when undefined) and n_nodes.

Conditions

Raises hypergraphs_bad_input when hg is not a net_hg or has fewer than two distinct hyperedge sizes of at least two.

References

Lotito, Q. F., Contisciani, M., De Bacco, C., Di Gaetano, L., Gallo, L., Montresor, A., Musciotto, F., Ruggeri, N., & Battiston, F. (2023). Hypergraphx: a library for higher-order network analysis. *Journal of Complex Networks*, 11(3), cnad019. doi:10.1093/comnet/cnad019

See Also

[hg_assortativity\(\)](#).

Examples

```
groups <- data.frame(
  member = c("a", "b", "a", "c", "b", "c", "a", "d",
             "a", "b", "c", "a", "b", "d", "c", "d", "e"),
  group = c("p1", "p1", "p2", "p2", "p3", "p3", "p4", "p4",
            "t1", "t1", "t1", "t2", "t2", "t2", "t3", "t3", "t3")
)
hg <- group_hypergraph(groups, node = "member", hyperedge = "group")
hg_degree_correlation(hg)
```

hg_dictionary

Label nodes with a dictionary of terms

Description

Codes each node by the hyperedges it belongs to. In a text hypergraph with documents as nodes, the hyperedges of a document are its words, or with `text_hypergraph(separator = ";")` its keywords, so a dictionary that lists the terms of each category labels every document that carries one of them. This is dictionary-based content analysis (Grimmer & Stewart 2013): the categories and their terms are the analyst's, and the labelling is a count. A node takes the category with the most matched terms. A node that matches no term, or matches two categories equally, takes no label, and a message reports how many. The result is the labels input of `hg_classify()` and `hg_hypergat()`, which spread or learn the labels to the nodes the dictionary cannot code.

Usage

```
hg_dictionary(hg, dictionary)
```

Arguments

hg	A <code>net_hg</code> , typically from <code>text_hypergraph()</code> .
dictionary	A named list of character vectors, one per category, each holding the terms of that category as they appear among the hyperedge names (lowercase for a text hypergraph built with <code>lowercase = TRUE</code>). A term may belong to one category only.

Value

A base `data.frame`, one row per labelled node: `node`, `label` (the category), `n_terms` (how many of the category's terms the node carries) and `terms` (those terms, joined by ";"), ordered by `label` and `node`. Raises `hypergraphs_bad_input` for a malformed dictionary and warns with `hypergraphs_unmatched_terms` when a term of the dictionary is not a hyperedge of `hg`.

References

Grimmer, J., & Stewart, B. M. (2013). Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis*, 21(3), 267-297. doi:10.1093/pan/mps028

Examples

```
papers <- data.frame(
  keywords = c("medical education; online learning",
               "teacher education; online learning",
               "medical students; assessment",
               "online learning; assessment")
)
kw <- text_hypergraph(papers, column = "keywords", separator = ";")
hg_dictionary(kw, dictionary = list(
  health = c("medical education", "medical students"),
  teachers = "teacher education"
))
```

hg_edge_centrality *s*-betweenness and *s*-closeness centrality of hyperedges

Description

Hyperedges become vertices in the unweighted *s*-line graph: two are adjacent when they share at least *s* nodes. Betweenness and closeness are then ordinary shortest-path vertex centralities on that graph. At *s* = 1 these are the paper's hyperedge 1-betweenness and 1-closeness.

Usage

```
hg_edge_centrality(
  hg,
  s = 1L,
  measure = c("betweenness", "closeness"),
  normalized = TRUE,
  top = NULL,
  start = NULL,
  end = NULL,
  step = NULL,
  window = NULL,
  at = NULL,
  snapshot_mode = c("active", "cumulative"),
  multiedges = TRUE
)
```

Arguments

hg	A static <code>net_hg</code> or a <code>temporal_hypergraph()</code> .
s	One or more positive integer intersection thresholds.
measure	Any of "betweenness" and "closeness".
normalized	Use NetworkX/HypergraphX normalization: undirected betweenness is scaled by $2 / ((n - 1) * (n - 2))$, and closeness uses the Wasserman–Faust correction for disconnected line graphs.

top Optional number of highest-scoring hyperedges to retain per (time, s, measure) group.

start, end, step, window, at Measurement grid passed to `hg_snapshots()` when hg is temporal.

snapshot_mode, multiedges Snapshot mode ("active" or "cumulative") and multi-edge handling passed to `hg_snapshots()` when hg is temporal.

Value

A tidy data frame with edge, s, measure, and value; temporal input adds a leading time column, the snapshot time: a date for a calendar hypergraph, the number on its clock otherwise.

References

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. doi:10.1098/rsta.2023.0141

Examples

```
h <- group_hypergraph(
  data.frame(member = c("a", "b", "c", "b", "c", "d"),
              event = rep(c("e1", "e2"), each = 3)),
  "member", "event"
)
hg_edge Centrality(h, s = 1)
```

hg_edges	<i>Hyperedge-level measures, as a tidy table</i>
----------	--

Description

One row per hyperedge. `hg_measures(what = "edges")` reports cardinality alone; this adds the incident weight, how many other hyperedges a hyperedge meets, and how large its vertex neighbourhood is — the hyperedge-side counterparts of degree and neighbourhood size, and the descriptive layer used to characterise higher-order structure (Coupette et al. 2024). A temporal hypergraph is evaluated snapshot by snapshot, with a leading time column, so the distribution of hyperedge sizes at several dates (the paper’s Figure 4b) or the neighbourhood size of a tribunal over time (Figure 5c) come from one call.

Usage

```
hg_edges(
  hg,
  what = c("edges", "distribution", "summary"),
  measure = c("size", "weight", "n_incident_edges", "n_neighbors"),
  s = 1L,
  start = NULL,
```

```

    end = NULL,
    step = NULL,
    window = NULL,
    at = NULL,
    snapshot_mode = c("active", "cumulative"),
    multiedges = TRUE
  )

```

Arguments

hg	A text_hypergraph() , knn_hypergraph() , any hypergraphs <code>net_hg</code> , or a temporal_hypergraph() .
what	"edges" (default) for one row per hyperedge, "distribution" for the empirical distribution of measure across hyperedges, or "summary" for its mean, standard deviation and quartiles.
measure	Which column <code>what = "distribution"</code> and <code>what = "summary"</code> describe: "size" (default), "weight", "n_incident_edges" or "n_neighbors".
s	Minimum number of shared vertices for another hyperedge to count as incident. The default 1 is ordinary incidence; larger values match the thresholds used by hg_edge_centrality() . Several values give one block of rows each, with an <code>s</code> column.
start, end, step, window, at	Measurement grid passed to hg_snapshots() when <code>hg</code> is temporal: the bounds of the period, how often to look, how much time each look covers, or the instants themselves.
snapshot_mode, multiedges	Snapshot mode ("active" or "cumulative") and multi-edge handling passed to hg_snapshots() when <code>hg</code> is temporal.

Value

With `what = "edges"`, a base `data.frame` with one row per hyperedge and columns `edge` (name), `size` (integer, vertices it contains), `weight` (numeric, its incidence weights summed), `n_incident_edges` (integer, other hyperedges sharing at least `s` vertices) and `n_neighbors` (integer, vertices adjacent to a member without being one). With `what = "distribution"`, a `hypergraphs_distribution` table with one row per distinct observed value of measure, ascending, with columns `value`, `n`, `proportion` and `ccdf` — the complementary cumulative distribution $P(X \geq \text{value})$, so the first row's `ccdf` is always 1; its `plot()` draws the CCDF. With `what = "summary"`, one row with `n_edges`, `mean`, `sd`, `min`, `q25`, `median`, `q75` and `max`. Several `s` values add an `s` column; temporal input adds a leading time column, and the summary is then a `hypergraphs_series` whose `plot()` draws each statistic against time.

References

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. doi:10.1098/rsta.2023.0141

See Also

[hg_measures\(\)](#) for vertex-level measures, [hg_line_graph\(\)](#) for the graph whose degrees `n_incident_edges` reports.

Examples

```
hg <- text_hypergraph(c(a = "salt and soup", b = "soup and stars",
                        c = "stars and salt"))
hg_edges(hg)
hg_edges(hg, what = "distribution")
hg_edges(hg, what = "summary", measure = "n_neighbors")
```

hg_embed	<i>Low-Dimensional Hypergraph Embedding</i>
----------	---

Description

Returns the node coordinates computed by hypergraphs’ existing hypergraph spectral or symmetric-NMF engine without exposing the incidental k-means assignments produced by [hg_cluster\(\)](#).

Usage

```
hg_embed(
  hg,
  dimensions = 2L,
  type = c("zhou", "random_walk"),
  method = c("spectral", "symnmf"),
  seed = NULL,
  nstart = 25L,
  max_iter = 500L,
  tol = 1e-06
)
```

Arguments

hg	Any hypergraphs <code>net_hg</code> .
dimensions	Number of embedding coordinates, one whole number between 2 and <code>n_nodes - 1</code> ; anything else raises <code>hypergraphs_bad_input</code> .
type	Laplacian type: "zhou" or "random_walk".
method	"spectral" (default) or Hayashi et al.’s "symnmf". SymNMF requires a dense incidence matrix.
seed	Optional initialization seed.
nstart	Number of k-means starts for the shared engine; for spectral embeddings this affects only the discarded cluster assignment, not the returned coordinates. For SymNMF it is the number of factor restarts.
max_iter, tol	SymNMF iteration controls.

Value

A data frame with node, stationary mass pi, and dim1 through dim<dimensions>.

References

Zhou, D., Huang, J., & Schölkopf, B. (2006). Learning with hypergraphs: clustering, classification, and embedding. *Advances in Neural Information Processing Systems 19*. doi:10.7551/mitpress/7503.003.0205

Hayashi, K., Aksoy, S. G., Park, C. H., & Park, H. (2020). Hypergraph random walks, Laplacians, and clustering. *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, 495-504. doi:10.1145/3340531.3412034

Examples

```
h <- group_hypergraph(
  data.frame(node = c("a", "b", "c", "b", "c", "d"),
             edge = rep(c("e1", "e2"), each = 3)),
  "node", "edge"
)
hg_embed(h, dimensions = 2)
```

hg_euler

*Euler characteristic of a simplicial complex***Description**

$\chi = \sum_k (-1)^k f_k$, the alternating sum of the number of k-simplices; by the Euler-Poincare theorem it equals the alternating sum of the Betti numbers.

Usage

```
hg_euler(sc)
```

Arguments

sc A simplicial_complex from `simplicial()`.

Value

A single integer.

References

Hatcher, A. (2002). *Algebraic Topology*. Cambridge University Press.

See Also

[hg_betti\(\)](#)

Examples

```
adj <- matrix(c(0, 1, 1, 0,
               1, 0, 1, 1,
               1, 1, 0, 1,
               0, 1, 1, 0), 4, 4, dimnames = list(letters[1:4], letters[1:4]))
hg_euler(simplicial(adj))
```

hg_get

Read a table from any hypergraphs object

Description

hg_get() is the single reader of the package. Every object a constructor or a verb returns – a hypergraph, a memory network, a simplicial complex, an inference or a community result – hands over its tables through hg_get(), never through \$. what = selects the table; what = NULL (the default) returns the object's primary table. Filters, orderings and top = are named arguments of the method for that class.

Usage

```
hg_get(x, what = NULL, ...)

## Default S3 method:
hg_get(x, what = NULL, ...)

## S3 method for class 'net_hg_pairwise'
hg_get(x, what = c("edges", "nodes"), ..., sort_by = NULL, top = NULL)
```

Arguments

x	A hypergraphs object.
what	NULL (default) for the object's primary table, or the name of one of its tables (see <i>Tables by class</i>).
...	Arguments of the method for class(x): filters (order_min, min_count, dim, k, dimension, significant, ...), sort_by, and top (the first top rows, applied last).
sort_by	Ordering supported by the selected object's method.
top	Number of rows to return, applied after filters and ordering as described by the selected object's method.

Value

A base data.frame, one row per observation of the selected table. An object of a class without a method raises hypergraphs_bad_input, naming the class.

Tables by class

net_hg (**hypergraphs**) "edges" (default), "nodes", "memberships", "sets", "state_counts", "edge_data"; see [hg_get.net_hg\(\)](#).

text_hypergraph "weights" (default), "documents", "vocabulary", "sentences"; see [hg_get.text_hypergraph\(\)](#).

hg_hypergat Predictions (default), evaluation, document text, history and optional attention; see [hg_get.hg_hypergat\(\)](#).

net_temporal_hypergraph "memberships" (default), "edges", "nodes".

net_hon ([hon\(\)](#)) "rules" (default), "nodes", "pathways".

net_honem ([honem\(\)](#)) "embeddings" (default), "variance".

net_hypa ([hypa\(\)](#) **on sequences**) "anomalies" (default), "scores", "over", "under", "pathways".

net_mogen ([mogen\(\)](#)) "orders" (default), "transitions", "paths", "pathways".

net_markov_order ([markov_order\(\)](#)) "orders" (default), "null".

net_path_dependence ([memory\(\)](#)) "contexts".

net_markov_stability ([hg_markov_stability\(\)](#)) "states" (default), "passage_time", "stationary".

hypergraphs_bootstrap ([hg_bootstrap\(\)](#)) "edges".

hypergraphs_comparison ([hg_compare\(\)](#)) "edges".

hypergraphs_memory_communities ([hg_communities\(\)](#) **on a memory network**) "states" (default), "physical", "modules", "codelength", "trials".

simplicial_complex ([simplicial\(\)](#)) "simplices" (default), "f_vector", "degree".

q_analysis ([hg_qanalysis\(\)](#)) "q_levels" (default), "nodes".

persistent_homology ([hg_homology\(\)](#)) "persistence" (default), "beti".

persistence_landscape ([hg_landscape\(\)](#)) "landscape".

hg_communities ([hg_communities\(\)](#) **on a hypergraph**) see [hg_get.hg_communities\(\)](#).

hypergraphs_community_comparison, hypergraphs_motifs, net_hg_mmsbm see their methods.

Examples

```
hg <- window_hypergraph(list(s1 = c("a", "b", "a", "c")), window = 2L)
hg_get(hg)
hg_get(hg, what = "nodes")

seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
hg_get(hon(seqs, max_order = 2), what = "nodes")
```

hg_get.hg_classification

Read a classification result

Description

hg_classify() and hg_hypergat() with holdout return an hg_classification: one row per document with its true label, its predicted label, its split and whether a held-out prediction is correct. hg_get() reads the evaluation tables, and for hg_hypergat() the diagnostic attention of the trained network, computed on request.

Usage

```
## S3 method for class 'hg_classification'
hg_get(
  x,
  what = c("predictions", "accuracy", "classes", "confusion", "hyperedges",
    "hyperedge_words", "documents"),
  split = NULL,
  correct = NULL,
  node = NULL,
  sort_by = NULL,
  top = NULL,
  ...
)

## S3 method for class 'hg_classification'
print(x, ...)

## S3 method for class 'hg_classification'
summary(object, ...)

## S3 method for class 'hg_classification'
plot(x, y, type = c("confusion", "hyperedges"), node = NULL, top = 8L, ...)
```

Arguments

x	An hg_classification.
what	"predictions" (default), the per-document table: node, label (the true label, NA for an unlabelled document), predicted, score, margin, split ("train", "test" or "unlabelled") and correct (for the test documents); "accuracy", one row: n_test, accuracy, balanced_accuracy (the mean recall over the classes) and chance (one over the number of classes, the balanced accuracy of a random guess); "classes", one row per class: class, n_test, correct, recall (the share of the class's test documents predicted as the class) and precision (the share of the test documents predicted as the class that belong to it); "confusion", one row per true and predicted class: label, predicted,

	n. For a result of <code>hg_hypergat()</code> only: "hyperedges", one row per document and hyperedge (sentence or topic): node, label, predicted, hyperedge, kind, text, weight (the mean edge-level attention of the hyperedge's words), top_word and n_words, the hyperedges of each document from the heaviest; or "hyperedge_words", one row per document, hyperedge and word: node, label, predicted, hyperedge, kind, text, word, word_weight (the node-level attention of the word within the hyperedge) and edge_weight (the edge-level attention of the hyperedge for the word), the quantities of Figure 5 of Ding et al. (2020). These internal weights are not prediction explanations. Fitted HyperGAT models also support "documents", the prediction and text of each document.
split	For "predictions" or "documents": keep only the "train", the "test" or the "unlabelled" documents, the last being those without a label, whose predictions are the classifier's output.
correct	For "predictions" or "documents": TRUE keeps the test documents predicted correctly, FALSE those predicted wrongly.
node	For "predictions", "hyperedges" and "hyperedge_words": keep only these documents, given as ids or as a table with a node column, such as a table <code>hg_get()</code> returned. For "hyperedge_words", a table that also has a hyperedge column keeps only those hyperedges.
sort_by	For "predictions": "margin" or "score", from the largest.
top	For "predictions": the first top rows of each true label, so <code>split = "test"</code> , <code>correct = TRUE</code> , <code>sort_by</code> gives the most confident correct prediction of each class. For "hyperedges": the top heaviest hyperedges of each document. For "hyperedge_words": the top words of largest word_weight in each hyperedge.
...	Unused.
object	An <code>hg_classification</code> .
y	Unused.
type	For <code>plot()</code> : "confusion" (default), the counts of true against predicted labels on the test documents, or "hyperedges" (a result of <code>hg_hypergat()</code> only), the top heaviest hyperedges of each document in node, with its uniform normalization baseline. Without node, the first held-out document of each class is selected (the first training document if there is no holdout).

Value

A base `data.frame` as described under `what`. `print()` shows the held-out accuracy and the per-class table and returns `x` invisibly; `summary()` returns the per-class table; `plot()` returns a `ggplot` of the confusion table (`type = "confusion"`) or of the heaviest hyperedges of some documents (`type = "hyperedges"`).

References

Brodersen, K. H., Ong, C. S., Stephan, K. E., & Buhmann, J. M. (2010). The balanced accuracy and its posterior distribution. *Proceedings of the 20th International Conference on Pattern Recognition*, 3121-3124. doi:10.1109/ICPR.2010.764

Ding, K., Wang, J., Li, J., Li, D., & Liu, H. (2020). Be more with less: Hypergraph attention networks for inductive text classification. *EMNLP 2020*, 4927-4936.

Examples

```
hg <- text_hypergraph(head(forum_posts, 60), column = "text")
fit <- hg_classify(hg, labels = "topic", holdout = 0.25)
fit
hg_get(fit, what = "classes")
hg_get(fit, what = "confusion")
hg_get(fit, split = "test", correct = FALSE)
```

hg_get.hg_hypergat	<i>Read a fitted HyperGAT classifier</i>
--------------------	--

Description

The classifier retains its trained network, vocabulary and topic keywords. Reading a table or predicting new documents never trains the network. Attention tables describe internal aggregation weights. They are not class-specific contributions or explanations of a prediction.

Usage

```
## S3 method for class 'hg_hypergat'
hg_get(
  x,
  what = NULL,
  split = NULL,
  correct = NULL,
  node = NULL,
  sort_by = NULL,
  top = NULL,
  ...
)

## S3 method for class 'hg_hypergat'
print(x, ...)

## S3 method for class 'hg_hypergat'
summary(object, ...)

## S3 method for class 'hg_hypergat'
plot(
  x,
  y,
  type = c("confusion", "history", "hyperedges", "attention"),
  node = NULL,
  top = 8L,
```

```
    ...
)
```

Arguments

x	A fitted hg_hypergat.
what	NULL or "predictions", "accuracy", "classes", "confusion", "history", "documents", "attention", "hyperedges" or "hyperedge_words". Evaluation tables use held-out documents only. "documents" includes the text with its prediction.
split, correct, sort_by	Filters and ordering for predictions and documents, as in hg_get.hg_classification() .
node	Document ids or a table with a node column.
top	Number of rows per class (predictions and documents), per document (hyperedges or attention), or per hyperedge (hyperedge words).
...	Unused.
object	A fitted hg_hypergat.
y	Unused.
type	"confusion" (default), "history", "hyperedges" (the word hypergraph of one document), or "attention". Attention plots show weights alongside their uniform normalization baseline. These weights do not measure a sentence's contribution to the predicted class.

Value

A plain data frame. The attention tables include uniform normalization baselines: `word_uniform = 1 / n_words` within an edge, and `edge_uniform = 1 / n_edges` for a word. Hyperedge summaries include `uniform_weight` (mean `edge_uniform`) and `exclusive_fraction` (the fraction of words belonging only to that hyperedge). Such words have `edge_weight = 1` regardless of the learned parameters.

References

Ding, K., Wang, J., Li, J., Li, D., & Liu, H. (2020). Be more with less: Hypergraph attention networks for inductive text classification. *EMNLP 2020*, 4927-4936.

hg_get.hypergraphs_bootstrap

Inference table of a memory-network bootstrap

Description

Inference table of a memory-network bootstrap

Usage

```
## S3 method for class 'hypergraphs_bootstrap'
hg_get(
  x,
  what = "edges",
  ...,
  min_support = NULL,
  order_min = NULL,
  sort_by = NULL,
  top = NULL
)
```

Arguments

x	A hypergraphs_bootstrap object from hg_bootstrap() .
what	"edges", the only table: one row per rule edge.
...	Additional arguments (ignored).
min_support	Numeric in $[\emptyset, 1]$ or NULL. Keep only rule edges with at least this bootstrap support.
order_min	Integer or NULL. Keep only rule edges of at least this order (e.g. 2 for the genuinely higher-order rules).
sort_by	NULL (order/from/to, default), "count", "probability", or "support" - sort the table by that column, largest first (ties broken by from/to so the order is deterministic).
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame, one row per rule edge: from, to, order, count, probability, ci_lower, ci_upper, support, n_boot_used.

hg_get.hypergraphs_comparison

Edge table of a memory-network comparison

Description

Edge table of a memory-network comparison

Usage

```
## S3 method for class 'hypergraphs_comparison'
hg_get(x, what = "edges", ..., significant = FALSE, sort_by = NULL, top = NULL)
```

Arguments

x	A hypergraphs_comparison object from hg_compare() .
what	"edges", the only table: one row per pooled rule edge.
...	Additional arguments (ignored).
significant	Logical. TRUE restricts to edges whose adjusted p-value falls below the object's alpha. Default FALSE (all edges).
sort_by	NULL (order/from/to, default), "abs_diff", "count", or "p_adj" - sort by absolute difference or count (largest first) or adjusted p-value (smallest first), ties broken by from/to.
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame, one row per pooled rule edge (see [hg_compare\(\)](#) for the columns).

hg_get.hypergraphs_hypa

Tables of a path-anomaly (HYPA) fit

Description

Tables of a path-anomaly (HYPA) fit

Usage

```
## S3 method for class 'hypergraphs_hypa'
hg_get(x, ...)

## S3 method for class 'net_hypa'
hg_get(
  x,
  what = c("anomalies", "scores", "over", "under", "pathways"),
  ...,
  type = NULL,
  order_by = NULL,
  top = NULL
)
```

Arguments

x	A net_hypa object from hypa() on sequences.
...	Unused.

what	"anomalies" (default) for the anomalous paths selected by type, "scores" for every scored path, "over" or "under" for the significantly over- or under-represented paths with all their columns, or "pathways" for the anomalous paths written as pathways ("a b -> c": the context states, then the next state), the form <code>simplicial()</code> turns into simplices.
type	For "anomalies" and "pathways": "all", "over" or "under". Default: the type given to <code>hupa()</code> , else "all".
order_by	"sig" (smallest p-value of the path's own direction first), "ratio" (largest deviation first: highest ratio for over-represented paths, lowest for under-represented ones, largest absolute log ratio when both directions are listed), "freq" (most observed first) or "path" (alphabetical). Default: the order_by given to <code>hupa()</code> , else "sig". For "scores", "sig" sorts by the smaller of the two raw tails, <code>p_under</code> and <code>p_over</code> , so the paths that depart most from the null in either direction come first.
top	Integer or NULL. Return only the first top rows, applied after any filter and after order_by. Default NULL returns every row.

Value

A data.frame. For "anomalies", one row per anomalous path: path, count, expected, ratio, `p_adj` (the adjusted p-value of the path's own direction), direction, and order when several orders were scored. For "scores", "over" and "under", one row per path with every column of the fit: path, from, to, count, expected, ratio, the p-values (`p_value`, an alias of `p_under`, the lower tail $P(X \leq f)$; `p_over`, the strict upper tail $P(X > f)$; `p_adj_under`, `p_adj_over`), the direction ("over", "under" or "normal") and the path order. "scores" holds every possible path: the observed ones and the unobserved ones the observed transitions permit (count 0). For "pathways", one row per anomalous path with the column pathway.

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"),
            c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
fit <- hupa(seqs, order = 2, min_count = 1)
hg_get(fit, type = "over", order_by = "ratio", top = 5)
```

hg_get.hypergraphs_memory_communities

Tidy tables of a memory-network community result

Description

Tidy tables of a memory-network community result

Usage

```
## S3 method for class 'hypergraphs_memory_communities'
hg_get(
  x,
  what = c("states", "physical", "modules", "trials", "first_order", "codelength"),
  ...,
  community = NULL,
  overlapping = FALSE
)
```

Arguments

x	A hypergraphs_memory_communities object.
what	Which table: "states" (default) one row per memory node: node (the memory node, "a -> b"), state (the physical state it belongs to, b), community, flow. "physical" one row per state x community it appears in: state, community, flow ($\pi_{i \cap m}$), share (of that state's flow), n_communities (> 1 marks an overlapping state). "modules" one row per community: community, n_nodes (memory nodes), n_states, flow, exit_flow, enter_flow. "trials" one row per search run: run, codelength, n_communities, ari_to_best (adjusted Rand index with the kept partition), best. "first_order" one row per state: state, its community in the best first-order partition, and its first-order flow. "codelength" one row per model (memory, first_order): codelength, index_codelength, module_codelength, one_level_codelength, savings_bits, savings_pct (0 when the one-module codelength is 0, as for a single state: there is nothing to save), n_communities.
...	Ignored.
community	Integer vector or NULL: keep only these communities (tables "states", "physical", "modules").
overlapping	Logical: for what = "physical", keep only states that appear in more than one community. Default FALSE.

Value

A base data.frame as described under what.

Examples

```
seqs <- list(c("a", "h", "b", "a", "h", "b", "a"),
             c("c", "h", "d", "c", "h", "d", "c"))
comm <- hg_communities(hon(seqs, max_order = 2L), trials = 2L)
hg_get(comm, what = "modules")
hg_get(comm, what = "physical", overlapping = TRUE)
```

```
hg_get.hypergraphs_result
```

Print a result of an imported estimator

Description

Prints one line naming the result and its main settings, then the first rows of its default table, the table `hg_get()` returns.

Usage

```
## S3 method for class 'hypergraphs_result'
hg_get(x, ...)

## S3 method for class 'hypergraphs_result'
print(x, n = 10L, ...)

## S3 method for class 'hypergraphs_result'
plot(x, y, what = c("states", "passage_time"), ...)
```

Arguments

<code>x</code>	A result of <code>hon()</code> , <code>honem()</code> , <code>mogen()</code> , <code>markov_order()</code> , <code>memory()</code> , <code>hg_markov_stability()</code> , <code>hg_homology()</code> , <code>hg_landscape()</code> or <code>hg_qanalysis()</code> .
<code>...</code>	Unused.
<code>n</code>	Number of rows of the table to show. Default 10.
<code>y</code>	Unused.
<code>what</code>	For a Markov stability result: "states" (default), one panel of bars per measure, or "passage_time", the mean first-passage times as a heatmap (rows are the starting state).

Value

`x`, invisibly.

`plot()` returns a ggplot for a persistence landscape (the landscapes that are not zero everywhere, one colour and line type each) and for a Markov stability result; every other result is plotted by its estimator's method.

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
print(hon(seqs, max_order = 2), n = 3)
```

hg_get.hypergraphs_summary

Summaries of results

Description

summary() on a result returns every table of the result as a list of data frames, named after the what values of `hg_get()`, so `summary(x)$validation` is the same data frame as `hg_get(x, what = "validation")`. Results with overall figures add them as further one-row or per-group tables. Tables that a particular result does not hold are left out. The result itself is not changed.

Usage

```
## S3 method for class 'hypergraphs_summary'
hg_get(x, what = names(x)[1L], ...)

## S3 method for class 'hypergraphs_summary'
print(x, n = 5L, ...)

## S3 method for class 'hypergraphs_result'
summary(object, ...)

## S3 method for class 'hypergraphs_hypa'
summary(object, ...)

## S3 method for class 'hypergraphs_community_comparison'
summary(object, ...)

## S3 method for class 'hg_communities'
summary(object, ...)

## S3 method for class 'net_hg_mmsbm'
summary(object, ...)

## S3 method for class 'hypergraphs_motifs'
summary(object, ...)

## S3 method for class 'hypergraphs_memory_communities'
summary(object, ...)

## S3 method for class 'hypergraphs_bootstrap_group'
summary(object, ...)

## S3 method for class 'hypergraphs_memory_group'
summary(object, ...)

## S3 method for class 'hypergraphs_bootstrap'
```

```
summary(object, ...)

## S3 method for class 'hypergraphs_comparison'
summary(object, ...)

## S3 method for class 'hypergraphs_simplicial'
summary(object, ...)

## S3 method for class 'net_hg_topics'
summary(object, ...)
```

Arguments

x	A hypergraphs_summary.
what	For <code>hg_get()</code> on a summary: the name of one of its tables. Default: the first.
...	Unused.
n	For <code>print()</code> : number of rows of each table to show. Default 5.
object	A result: from <code>hon()</code> , <code>honem()</code> , <code>mogen()</code> , <code>markov_order()</code> , <code>memory()</code> , <code>hg_markov_stability()</code> , <code>hupa()</code> on sequences, <code>hg_bootstrap()</code> , <code>hg_compare()</code> , <code>hg_communities()</code> , <code>hg_compare_communities()</code> , <code>hg_mmsbm()</code> , <code>hg_motifs()</code> , <code>simplicial()</code> , <code>hg_homology()</code> , <code>hg_landscape()</code> or <code>hg_qanalysis()</code> .

Value

A hypergraphs_summary: a named list of data frames. Its print method shows the first rows of each.

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"),
            c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
fit_summary <- summary(memory(seqs, order = 2, min_count = 1))
fit_summary
fit_summary$contexts
```

hg_get.net_hg

Tables of a hypergraph

Description

One row per hyperedge: its identifier, size (number of distinct member states), the member states, and its weight (for hypergraphs built by `window_hypergraph()`, the number of windows collapsed into the hyperedge; NA for the other constructors, whose hyperedges are unweighted).

Usage

```
## S3 method for class 'net_hg'
hg_get(
  x,
  what = c("edges", "nodes", "memberships", "sets", "state_counts", "edge_data"),
  ...,
  sort_by = NULL,
  top = NULL
)
```

Arguments

x	A net_hg object.
what	"edges" (default) for one row per hyperedge, "nodes" for one row per node, or "memberships" for one row per node-in- hyperedge cell of the incidence matrix. A hypergraph of clustered sequences (group_hypergraph() on a clustering of sequences) also has "sets" and "state_counts". "edge_data" returns the per-hyperedge attribute table a constructor attached.
...	Additional arguments (ignored).
sort_by	NULL (construction order, default), "weight", or "size" - sort the table by that column, largest first (ties broken by hyperedge id so the order is deterministic).
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame. For what = "edges", one row per hyperedge with columns hyperedge (character id), size (integer), members (the member nodes, comma separated), and, for a hypergraph of windows ([window_hypergraph\(\)](#)), weight (the number of windows with that member set). sort_by is "size", or "weight" where it exists. For what = "nodes", one row per node with columns node and degree (the number of hyperedges it belongs to), plus block for a hypergraph with planted blocks ([random_hypergraph\(\)](#) with type = "sbm"); sort_by = "degree" orders it. For what = "memberships", one row per non-zero incidence cell with columns node, hyperedge and weight (the incidence value: 1 for a binary hypergraph, the summed weight for a weighted one), in hyperedge order; sort_by = "weight" orders it. For what = "sets", one row per hyperedge with group, hyperedge, set (the states joined by " + "), size, count (sequences of the group with exactly that set) and share (count over the group's sequences), in group order and decreasing count. For what = "state_counts", one row per group and state with group, node, count (sequences of the group containing the state) and share. For what = "edge_data", one row per hyperedge with edge and its attributes; asking for it of a hypergraph without an attribute table raises `hypergraphs_bad_input`.

Examples

```
hg <- window_hypergraph(list(s1 = c("a", "b", "a", "c")), window = 2L)
hg_get(hg)
hg_get(hg, sort_by = "weight")
hg_get(hg, sort_by = "weight", top = 3)
hg_get(hg, what = "nodes", sort_by = "degree")
```

```
hg_get(hg, what = "memberships")
```

hg_get.net_hg_cluster *Assignment table of a hypergraph clustering fit*

Description

Assignment table of a hypergraph clustering fit

Usage

```
## S3 method for class 'net_hg_cluster'
hg_get(x, what = "assignments", ..., top = NULL)
```

Arguments

x	A net_hg_cluster object.
what	"assignments", the only table.
...	Additional arguments (ignored).
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

The tidy assignment table: one row per node, columns node, cluster, pi (stationary probability of the node under the Laplacian's random walk) and the spectral-embedding or NMF-factor coordinates dim1..dimk.

hg_get.net_hg_measures

Tables of a hypergraph measures fit

Description

Tables of a hypergraph measures fit

Usage

```
## S3 method for class 'net_hg_measures'
hg_get(
  x,
  what = c("nodes", "edges", "global"),
  ...,
  sort_by = NULL,
  top = NULL
)
```

Arguments

x	A net_hg_measures object.
what	"nodes" (default) for the per-node measures, "edges" for the per-hyperedge measures, or "global" for the whole-hypergraph summary.
...	Additional arguments (ignored).
sort_by	NULL (node/edge order, default), or a column name to sort by, largest first. Ignored when what = "global".
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame. For what = "nodes", one row per node: node, hyperdegree, node_strength, max_edge_size. For what = "edges", one row per hyperedge: hyperedge, size. For what = "global", one row per statistic: measure, value.

hg_get.net_hg_transduction

Tables of a hypergraph transduction fit

Description

Tables of a hypergraph transduction fit

Usage

```
## S3 method for class 'net_hg_transduction'
hg_get(x, what = c("predictions", "scores"), ..., top = NULL)
```

Arguments

x	A net_hg_transduction object.
what	Character. "predictions" (default) for the one-row-per-node table, "scores" for the tidy long score table (one row per node x class: node, class, score).
...	Additional arguments (ignored).
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame as selected by what.

hg_get.net_hon	<i>Tables of a higher-order network</i>
----------------	---

Description

Tables of a higher-order network

Usage

```
## S3 method for class 'net_hon'
hg_get(
  x,
  what = c("rules", "nodes", "pathways"),
  ...,
  order_min = NULL,
  sort_by = NULL,
  top = NULL
)
```

Arguments

x	A net_hon object from hon() .
what	"rules" (default) for the extracted higher-order rules, "nodes" for the node table of the higher-order topology, or "pathways" for the genuinely higher-order rules written as pathways ("a b -> c": the context states, then the next state), most frequent first – the form simplicial() turns into simplices.
...	For what = "pathways": min_count, min_prob (drop rules below this count or probability) and order (keep one context order). Otherwise unused.
order_min	Integer or NULL. Keep only rules whose source context is at least this order (e.g. 2 for the genuinely higher-order rules). Only for what = "rules".
sort_by	NULL (construction order, default), "count" or "probability" - sort largest first, ties broken by from/to so the order is deterministic. Only for what = "rules".
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame. For what = "rules", one row per rule edge with columns path, from, to, count, probability, from_order, to_order. For what = "nodes", one row per higher-order node with columns id and node. For what = "pathways", one row per higher-order rule with the column pathway.

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"),
            c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
net <- hon(seqs, max_order = 2)
hg_get(net, order_min = 2)
hg_get(net, what = "nodes")
hg_get(net, what = "pathways")
```

hg_get.net_honem	<i>Tables of a higher-order network embedding</i>
------------------	---

Description

Tables of a higher-order network embedding

Usage

```
## S3 method for class 'net_honem'
hg_get(x, what = c("embeddings", "variance"), ..., top = NULL)
```

Arguments

x	A net_honem object from honem() .
what	"embeddings" (default) for the node coordinates, or "variance" for the per-dimension singular values.
...	Unused.
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame. For what = "embeddings", one row per higher-order node: node followed by one column per embedding dimension (dim1, dim2, ...). For what = "variance", one row per dimension: dimension, singular_value, proportion (that dimension's share of the total squared singular value; 0 for every dimension when all singular values are zero, as for a network without edges, whose embedding carries no variance).

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"),
            c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
emb <- honem(hon(seqs, max_order = 2), dim = 2)
hg_get(emb)
hg_get(emb, what = "variance")
```

hg_get.net_markov_order

Tables of a Markov order test

Description

Tables of a Markov order test

Usage

```
## S3 method for class 'net_markov_order'
hg_get(x, what = c("orders", "null"), ..., top = NULL)
```

Arguments

x	A net_markov_order object from markov_order() .
what	"orders" (default) for the per-order test table, or "null" for the permutation null distribution behind it.
...	Additional arguments (ignored).
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame. For what = "orders", one row per candidate order: order, log_likelihood, aic, bic, df, g2 (the likelihood-ratio statistic against the order below), p_value (its permutation p-value), p_asymptotic (its chi-squared p-value) and significant. For what = "null", one row per permutation replicate: order, replicate, g2 (the replicate's likelihood-ratio statistic under the null).

Examples

```
seqs <- list(c("a", "b", "a", "c", "a", "b"), c("b", "a", "c", "a", "b", "a"))
fit <- markov_order(seqs, max_order = 2L, n_perm = 20L, seed = 1L)
hg_get(fit)
hg_get(fit, what = "null", top = 5)
```

hg_get.net_markov_order_group

Tables of a grouped Markov order test

Description

The per-group test tables stacked into one data.frame, with a group column naming the group each row came from. This is the supported way to read a net_markov_order_group; the print method shows each group in turn but a comparison across groups needs one table.

Usage

```
## S3 method for class 'net_markov_order_group'
hg_get(x, what = c("orders", "null"), ..., top = NULL)
```

Arguments

x	A net_markov_order_group, as returned by markov_order() on a netobject_group.
what	"orders" (default) for one row per group and order, or "null" for the permutation null draws, one row per group, order and replicate.
...	Unused.
top	Keep only the first top rows, applied last.

Value

A base data.frame. For what = "orders", one row per group and order, with group first and then the columns of [hg_get.net_markov_order\(\)](#). For what = "null", one row per group, order and replicate with columns group, order, replicate, g2. Returns a zero-row frame with those columns when the group is empty.

Examples

```
seqs <- list(c("a", "b", "a", "c"), c("b", "a", "c", "a"))
fit <- markov_order(seqs, max_order = 2L, n_perm = 20L, seed = 1L)
hg_get(fit)
```

hg_get.net_markov_stability

Tables of a Markov stability analysis

Description

Tables of a Markov stability analysis

Usage

```
## S3 method for class 'net_markov_stability'
hg_get(
  x,
  what = c("states", "passage_time", "stationary"),
  ...,
  sort_by = NULL,
  decreasing = TRUE,
  from = NULL,
  to = NULL,
  top = NULL
)
```

Arguments

<code>x</code>	A <code>net_markov_stability</code> object from <code>hg_markov_stability()</code> .
<code>what</code>	Which table: "states" (default, one row per state), "passage_time" (one row per ordered pair of states) or "stationary" (one row per state, distribution only).
<code>...</code>	Ignored.
<code>sort_by</code>	NULL (construction order, the default) or a column name of the selected table.
<code>decreasing</code>	Logical. Sort <code>sort_by</code> from largest to smallest (TRUE, the default) or smallest to largest (FALSE). Ties are broken by the row key, ascending in both directions: state, or from then to for the passage table.
<code>from, to</code>	NULL (default, every state) or a character vector of states. Restrict the <code>what = "passage_time"</code> table to passages leaving the <code>from</code> states and arriving at the <code>to</code> states. Only that table has <code>from/to</code> columns; using either with another <code>what</code> , or naming a state the chain does not have, raises <code>hypergraphs_bad_input</code> .
<code>top</code>	Integer or NULL. Keep only the first <code>top</code> rows, applied last: after <code>what</code> , <code>from/to</code> and <code>sort_by</code> .

Value

A base data.frame. For `what = "states"`, one row per state with `state`, `persistence`, `stationary`, `return_time`, `sojourn_time`, `avg_time_to_others` and `avg_time_from_others`, as the estimator stores them (probabilities rounded to 4 decimals, times to 2). For `what = "passage_time"`, one row per ordered pair with `from`, `to` and `steps`. For `what = "stationary"`, one row per state with `state`, `stationary` and `return_time`; the passage and stationary tables are unrounded.

Examples

```
P <- matrix(c(0.5, 0.3, 0.2,
             0.2, 0.5, 0.3,
             0.1, 0.4, 0.5), 3, 3, byrow = TRUE,
           dimnames = list(c("A", "B", "C"), c("A", "B", "C")))
ms <- hg_markov_stability(P)
hg_get(ms, sort_by = "stationary")
```

```
hg_get(ms, what = "passage_time", from = "A")
hg_get(ms, what = "stationary")
```

hg_get.net_mogen

Tables of a multi-order generative model

Description

Tables of a multi-order generative model

Usage

```
## S3 method for class 'net_mogen'
hg_get(
  x,
  what = c("orders", "transitions", "paths", "pathways"),
  ...,
  order = NULL,
  k = 2L,
  min_count = 1L,
  top = NULL
)
```

Arguments

x	A net_mogen object from mogen() .
what	"orders" (default) for the per-order model-selection table, "transitions" for the fitted transition probabilities of one layer, "paths" for the frequency of every observed path of k states, or "pathways" for the layer's transitions written as pathways ("a b -> c"), the form simplicial() turns into simplices.
...	For what = "pathways", min_prob (drop transitions below this probability). Otherwise unused.
order	Integer or NULL. For what = "transitions" and "pathways": which layer to read. Default NULL uses the selected order.
k	Integer vector, for what = "paths": the path length or lengths in states, each between 2 and the model's highest order plus one, e.g. k = 3 or k = 2:4. Default 2 (single transitions).
min_count	Integer, for what = "transitions" and "pathways": drop transitions observed fewer times. Default 1.
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame. For what = "orders", one row per candidate order: order, log_likelihood, aic, bic, df, layer_df, and optimal (logical, TRUE for the selected order). For what = "transitions", one row per transition: path, count, probability, from, to. For what = "paths", one row per distinct path of each length in k (top applies within each length), grouped by length and most frequent first (ties alphabetical): k, path, count, proportion (share of all k-state paths, rounded to 4 decimals). For what = "pathways", one row per transition with the column pathway.

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"),
             c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
mg <- mogen(seqs, max_order = 2)
hg_get(mg)
hg_get(mg, what = "transitions", top = 5)
hg_get(mg, what = "paths", k = 3)
```

hg_get.net_path_dependence

Per-context table of a path-dependence analysis

Description

Per-context table of a path-dependence analysis

Usage

```
## S3 method for class 'net_path_dependence'
hg_get(
  x,
  what = "contexts",
  ...,
  min_count = NULL,
  flips = NULL,
  sort_by = "k1",
  top = NULL
)
```

Arguments

x	A net_path_dependence object from <code>memory()</code> .
what	"contexts", the only table.
...	Additional arguments (ignored).
min_count	Integer or NULL. Keep only contexts observed at least this many times.
flips	NULL (default, every context), TRUE for only the contexts whose most probable next state differs from the first-order prediction (flips is TRUE), or FALSE for only those where it agrees.

sort_by	"kl" (default) sorts by the order-k vs order-1 divergence, largest first; NULL keeps the construction order.
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame, one row per context: context, count (times observed), entropy_first_order and entropy_order_k (entropy of the next state given the last state and given the whole context), entropy_drop (their difference), kl (Kullback-Leibler divergence of the order-k from the first-order distribution), top_first_order and top_order_k (the most probable next state under each), and flips (TRUE when the two differ).

Examples

```
# one sequence per row
wide <- data.frame(t1 = c("a", "x", "a", "x"), t2 = c("b", "b", "b", "b"),
                  t3 = c("c", "d", "c", "d"), t4 = c("a", "x", "a", "x"),
                  t5 = c("b", "b", "b", "b"), t6 = c("c", "d", "c", "d"))
pd <- memory(wide, order = 2, min_count = 1)
hg_get(pd)
hg_get(pd, flips = TRUE)
```

hg_get.net_temporal_hypergraph

Tidy tables of a temporal hypergraph

Description

Tidy tables of a temporal hypergraph

Usage

```
## S3 method for class 'net_temporal_hypergraph'
hg_get(x, what = c("memberships", "edges", "nodes"), ...)
```

Arguments

x	A temporal_hypergraph() .
what	"memberships" (default) for one row per node-in-hyperedge spell (member, edge, start, end, weight), "edges" for one row per hyperedge with its clock and attributes, "nodes" for the node universe with entry times. Times are on the hypergraph's clock.
...	Ignored.

Value

A base data.frame.

hg_get.persistence_landscape

Table of a persistence landscape

Description

Table of a persistence landscape

Usage

```
## S3 method for class 'persistence_landscape'
hg_get(x, what = "landscape", ..., k = NULL, top = NULL)
```

Arguments

x	A persistence_landscape object from hg_landscape() .
what	"landscape", the only table.
...	Additional arguments (ignored).
k	Integer or NULL. Keep only this landscape level.
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame, one row per (level, grid point): k (landscape level), t (filtration value), value (landscape height).

Examples

```
set.seed(1)
w <- matrix(runif(36), 6, 6, dimnames = list(letters[1:6], letters[1:6]))
w <- (w + t(w)) / 2
diag(w) <- 0
pl <- hg_landscape(hg_homology(w, n_steps = 10),
                  dimension = 0)
hg_get(pl, k = 1, top = 5)
```

hg_get.persistent_homology

Tables of a persistent homology result

Description

Tables of a persistent homology result

Usage

```
## S3 method for class 'persistent_homology'
hg_get(
  x,
  what = c("persistence", "beti"),
  ...,
  dimension = NULL,
  sort_by = NULL,
  top = NULL
)
```

Arguments

x	A persistent_homology object from hg_homology() .
what	"persistence" (default) for the persistence diagram, or "beti" for the Betti curves across the filtration.
...	Additional arguments (ignored).
dimension	Integer or NULL. Keep only this homological dimension.
sort_by	NULL (construction order, default) or "persistence" - longest-lived features first. Only for what = "persistence".
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame. For what = "persistence", one row per feature: dimension, birth, death, persistence. For what = "beti", one row per (threshold, dimension): threshold, dimension, beti.

Examples

```
set.seed(1)
w <- matrix(runif(36), 6, 6, dimnames = list(letters[1:6], letters[1:6]))
w <- (w + t(w)) / 2
diag(w) <- 0
ph <- hg_homology(w, n_steps = 10)
hg_get(ph, sort_by = "persistence")
hg_get(ph, what = "beti", dimension = 0)
```

hg_get.q_analysis	<i>Tables of a q-analysis</i>
-------------------	-------------------------------

Description

Tables of a q-analysis

Usage

```
## S3 method for class 'q_analysis'
hg_get(x, what = c("q_levels", "nodes"), ..., top = NULL)
```

Arguments

x	A q_analysis object from hg_qanalysis() .
what	"q_levels" (default) for the number of q-connected components at each level, or "nodes" for each node's highest q-level.
...	Additional arguments (ignored).
top	Integer or NULL. Return only the first top rows, applied after any filter and after sort_by, so sort_by and top compose. Default NULL returns every row.

Value

A data.frame. For what = "q_levels", one row per level: q, components. For what = "nodes", one row per node: node, max_q.

Examples

```
adj <- matrix(c(0, 1, 1, 0,
                1, 0, 1, 1,
                1, 1, 0, 1,
                0, 1, 1, 0), 4, 4, dimnames = list(letters[1:4], letters[1:4]))
qa <- hg_qanalysis(simplicial(adj, type = "clique"))
hg_get(qa)
hg_get(qa, what = "nodes")
```

hg_get.simplicial_complex	<i>Tables of a simplicial complex</i>
---------------------------	---------------------------------------

Description

Tables of a simplicial complex

Usage

```
## S3 method for class 'simplicial_complex'
hg_get(
  x,
  what = c("simplices", "f_vector", "betti", "degree", "validation"),
  ...,
  dimension = NULL,
  normalized = FALSE,
  top = NULL
)
```

Arguments

<code>x</code>	A <code>simplicial_complex</code> object from <code>simplicial()</code> .
<code>what</code>	"simplices" (default) for one row per simplex, "f_vector" for the face counts by dimension, "betti" for the Betti numbers by dimension, "degree" for the simplicial degree of every node (the table of <code>hg_degree()</code>), or "validation" for the tests of a window complex built with <code>validate = TRUE</code> .
<code>...</code>	Additional arguments (ignored).
<code>dimension</code>	Integer or <code>NULL</code> . For <code>what = "simplices"</code> : keep only simplices of this dimension (0 vertices, 1 edges, 2 triangles). For <code>what = "validation"</code> : keep only the sets of <code>dimension + 1</code> actions.
<code>normalized</code>	Logical. Only for <code>what = "degree"</code> : divide each count by its largest possible value. Default <code>FALSE</code> .
<code>top</code>	Integer or <code>NULL</code> . Return only the first <code>top</code> rows, applied after any filter and after <code>sort_by</code> , so <code>sort_by</code> and <code>top</code> compose. Default <code>NULL</code> returns every row.

Value

A data.frame. For `what = "simplices"`, one row per simplex: simplex (its number), dimension, size, members (the node names, comma separated). For `what = "f_vector"`, one row per dimension: dimension, count. For `what = "betti"`, one row per dimension: dimension, betti (the number of connected parts, loops, voids, ...). For `what = "degree"`, one row per node, most connected first: node, d0, d1, ... and total. For `what = "validation"`, one row per tested set of actions, by size, then p-value, then z: members (the actions, comma separated), size, count (windows with exactly these actions), n_windows (windows with this many distinct actions), expected (the mean count under the null model), z (the count minus expected, in null standard deviations), p_value, p_adj (Benjamini-Hochberg over n_tests possible sets), n_tests, significant.

Examples

```
adj <- matrix(c(0, 1, 1, 0,
               1, 0, 1, 1,
               1, 1, 0, 1,
               0, 1, 1, 0), 4, 4, dimnames = list(letters[1:4], letters[1:4]))
sc <- simplicial(adj, type = "clique")
hg_get(sc)
hg_get(sc, what = "f_vector")
```

hg_get.text_hypergraph

Tidy tables of a text hypergraph

Description

Tidy tables of a text hypergraph

Usage

```
## S3 method for class 'text_hypergraph'
hg_get(
  x,
  what = c("weights", "documents", "vocabulary", "sentences"),
  node = NULL,
  sort_by = NULL,
  top = NULL,
  ...
)
```

Arguments

x	A text_hypergraph() object.
what	Which table: "weights" (default) – for the bag construction one row per document-word pair (doc, word, count, weight); for the window construction one row per window-content/word membership (edge, word, weight = window count); for the sentence construction one row per sentence/word membership (edge, word, weight = in-sentence count); for the knn construction one row per hyperedge membership (doc, edge, weight = cosine similarity). "sentences" (sentence construction only) gives one row per sentence hyperedge (edge, doc, sentence, n_tokens). "documents" gives one row per document (with n_tokens/n_types for token-based constructions, plus any metadata columns carried from the input). "vocabulary" gives one row per word (word, count, doc_freq, and idf under tf-idf weighting; empty for the knn construction, which has no token layer).
node	For "documents": keep only these documents, given as ids or as a table with a node column, such as the predictions of hg_classify() , in the order given.
sort_by	For "vocabulary": "count" (total occurrences) or "doc_freq" (documents that contain the word), from the largest, ties by word.
top	Keep only the first top rows of the returned table.
...	Unused.

Value

A base data.frame as described under what.

Examples

```
hg <- text_hypergraph(c(a = "salt and soup", b = "soup and stars"))
hg_get(hg)
hg_get(hg, what = "documents")
hg_get(hg, what = "documents", node = "b")
```

hg_growth

*Growth of a temporal hypergraph over time***Description**

Counts, on a measurement grid, the nodes and hyperedges present in the temporal hypergraph: the descriptive time series of Coupette et al. (2024, Figures 4a and 5b). Hyperedges are counted twice, as they are (`n_edges`, the multi-hypergraph) and as distinct member sets (`n_edges_distinct`, the binary hypergraph), so the two representations of the paper's Table 2 can be read against each other over time.

Usage

```
hg_growth(
  x,
  start = NULL,
  end = NULL,
  step = NULL,
  window = NULL,
  at = NULL,
  mode = c("active", "cumulative"),
  components = FALSE
)

## S3 method for class 'hypergraphs_series'
plot(x, columns = NULL, facets = TRUE, ...)
```

Arguments

<code>x</code>	A <code>hypergraphs_series</code> table (for plot).
<code>start, end</code>	Bounds of the measured period, as numbers on the hypergraph's clock or dates for a calendar hypergraph. Default: the observation window.
<code>step</code>	How often to measure, in the hypergraph's time unit. <code>NULL</code> (default) measures at every event time.
<code>window</code>	How much time each measurement covers. Defaults to <code>step</code> when a step is given (a partition of the period) and to <code>0</code> otherwise (a point sample); larger than <code>step</code> gives a rolling window; "all" measures the whole period as one window.
<code>at</code>	Instants to measure instead of a grid; may be a vector.
<code>mode</code>	"active" (default) or "cumulative", see Description. The former "all" is deprecated: it is "cumulative" with <code>at</code> unset.

components	Also report the connectivity of the measured hypergraph at each time: the number of connected components, the share of its nodes in the largest, and its diameter (longest shortest path between two nodes sharing a chain of hyperedges)? Default FALSE; the computation is a breadth-first search per node and time.
columns	Which numeric columns to draw; default all but time.
facets	Draw one panel per column (default TRUE, because node and hyperedge counts live on different scales) or all columns in one panel with a legend.
...	Unused; for S3 consistency.

Details

mode = "active" counts what each window measures – an interval hyperedge active in it, a contact hyperedge occurring in it – and adds the _cumulative columns, the hyperedges begun by the end of the window, the static aggregate that a point-aggregation model would report. mode = "cumulative" reports only that growing view, and then counts a node from its own entry time when the constructor received a node table with a start column, otherwise from the first hyperedge that contains it.

Value

A base data.frame of class hypergraphs_series, one row per window, with time (the window start on the hypergraph's clock), n_nodes, n_edges, n_edges_distinct, n_memberships, and for mode = "active" also n_nodes_cumulative, n_edges_cumulative and n_edges_distinct_cumulative. With components = TRUE, further n_components, largest_component (share of active nodes) and diameter. plot() draws each count against time, on the calendar when the hypergraph has one.

For plot, a ggplot object.

References

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. doi:10.1098/rsta.2023.0141

Examples

```
seats <- data.frame(
  case = rep(c("A", "B", "C"), each = 3),
  arbitrator = c("p1", "a1", "a2", "p2", "a1", "a3", "p1", "a4", "a5"),
  constituted = rep(c(1, 2, 4), each = 3), concluded = rep(c(4, 3, 6), each = 3)
)
thg <- temporal_hypergraph(seats, node = "arbitrator", hyperedge = "case",
  start = "constituted", end = "concluded")
growth <- hg_growth(thg)
growth
plot(growth)
# a regular grid: every two steps, each window two steps wide
hg_growth(thg, step = 2)
```

hg_hnhn

*HNHN node classifier***Description**

Trains Hypergraph Networks with Hyperedge Neurons (Dong et al. 2020). A full layer sends node features to learned hyperedge representations and then sends those representations back to nodes. The paper's normalization is implemented directly: alpha changes the relative contribution of large hyperedges in edge-to-node messages, while beta changes the contribution of high-degree nodes in node-to-edge messages. Zero recovers ordinary neighborhood means in each direction.

Usage

```
hg_hnhn(
  hg,
  labels,
  features = "incidence",
  hidden = 128L,
  alpha = 0,
  beta = 0,
  epochs = 200L,
  lr = 0.01,
  weight_decay = 5e-04,
  dropout = 0.5,
  validation = 0.1,
  seed = 1L,
  verbose = FALSE
)
```

Arguments

hg	A text_hypergraph() (or any hypergraphs net_hg), dense or sparse.
labels	The known labels: a named character vector (names are node identifiers, values class labels) or a tidy data.frame with a node column and a label, cluster or predicted column. At least two classes.
features	Vertex feature matrix, one row per node in hg node order (rownames must match the node names), e.g. sbert embeddings. The default "incidence" uses the hypergraph's own weighted incidence rows (tf-idf bag-of-words features for a nodes = "doc" text hypergraph).
hidden	Hidden width (default 128).
alpha	Hyperedge-cardinality normalization exponent.
beta	Vertex-degree normalization exponent.
epochs	Training epochs (default 200).
lr, weight_decay	Adam learning rate and L2 penalty.

dropout	Hidden-layer dropout.
validation	Fraction of labeled nodes held out to select an epoch.
seed	Reproducible R and torch seed.
verbose	Message loss every 20 epochs.

Value

A prediction data.frame in the same format as `hg_neural()`, with training history and the normalization exponents attached.

References

Dong, Y., Sawin, W., & Bengio, Y. (2020). HNHN: Hypergraph networks with hyperedge neurons. *ICML Graph Representation Learning and Beyond Workshop*.

hg_homology	<i>Persistent homology of a weighted network or a distance matrix</i>
-------------	---

Description

Tracks the homology classes (components, loops, voids) of a filtration: a weighted clique filtration of a similarity matrix (`type = "clique"`, strongest edges first) or a Vietoris-Rips filtration of a distance matrix (`type = "vr"`). Every class is paired with the simplex that creates it (birth) and the one that destroys it (death), by exact boundary-matrix reduction over $\mathbb{Z}/2$.

Usage

```
hg_homology(x, n_steps = 20L, max_dim = 3L, type = "clique", max_scale = NULL)
```

Arguments

x	A square matrix, a network object carrying one (netobject, tna), points for <code>type = "vr"</code> (see <code>simplicial()</code>), a simplicial_complex built with <code>type = "vr"</code> (its stored filtration is used as is), or a window complex from <code>simplicial(type = "window")</code> . For a window complex the filtration is the count filtration (Petri et al. 2013): the complex at count t holds the sets of actions observed in at least t windows and their faces, so a class is born at the count where it appears and dies at a lower count; birth, death and the Betti-curve threshold are counts, and <code>n_steps</code> is not used (every observed count is a threshold). A class that never dies (essential) has <code>death = 0</code> and <code>persistence = birth</code> , as in clique mode, and every row of the Betti curve, the last included, counts the $\mathbb{Z}/2$ classes alive at that count in the complex truncated at <code>max_dim</code> .
n_steps	Grid points of the reported Betti curve. Default 20; the persistence diagram itself is exact.
max_dim	Highest simplex dimension tracked. Default 3.
type	"clique" (default) or "vr".
max_scale	For <code>type = "vr"</code> : the longest edge admitted. NULL uses the largest distance.

Value

A persistent_homology object. Read it with `hg_get()`: the persistence diagram (what = "persistence", the default) or the Betti curves ("betti").

References

- Edelsbrunner, H., Letscher, D., & Zomorodian, A. (2002). Topological persistence and simplification. *Discrete & Computational Geometry*, 28, 511–533. doi:10.1007/s0045400228852
- Petri, G., Scolamiero, M., Donato, I., & Vaccarino, F. (2013). Topological strata of weighted complex networks. *PLoS ONE*, 8(6), e66506. doi:10.1371/journal.pone.0066506

See Also

`hg_landscape()`, `hg_bottleneck()`, `hg_wasserstein()`

Examples

```
set.seed(1)
w <- matrix(runif(36), 6, 6, dimnames = list(letters[1:6], letters[1:6]))
w <- (w + t(w)) / 2
diag(w) <- 0
ph <- hg_homology(w, n_steps = 10)
hg_get(ph, sort_by = "persistence")
```

hg_hypergat

HyperGAT document classifier

Description

Trains the dual-attention hypergraph network of Ding et al. (2020) – every document becomes its own hypergraph (its unique words as vertices, its sentences as hyperedges), two attention layers aggregate words into sentence and optional LDA-topic hyperedges and those hyperedges back into words, and a masked mean pool feeds a linear classifier. Inductive: only labeled documents are trained on, and every document (labeled or not) is scored. Needs the suggested **torch** package. Semantics follow the official implementation. `semantic = "none"` reproduces its sentence-only ("w/o semantic") ablation. `semantic = "lda"` adds the full paper path: online variational-Bayes LDA is fitted to labeled documents only, with the topic count defaulting to the number of classes, and each document receives one edge per topic containing the topic's top words present in that document. The official repository's `generate_lda.py` fits its topics on every document, test documents included; passing its topic words as `lda_keywords` reproduces that choice.

Usage

```
hg_hypergat(
  x,
  labels,
  column = NULL,
```

```

    id = NULL,
    stop_words = stop_words_en(),
    min_count = 1L,
    lowercase = TRUE,
    semantic = c("none", "lda"),
    lda_keywords = NULL,
    lda_topics = NULL,
    lda_top_n = 10L,
    lda_max_iter = 10L,
    lda_batch_size = 128L,
    lda_offset = 50,
    lda_decay = 0.7,
    lda_seed = 0L,
    embed_dim = 300L,
    hidden = 100L,
    epochs = 10L,
    lr = 0.001,
    dropout = 0.3,
    batch_size = 8L,
    weight_decay = 1e-06,
    lr_decay = 0.1,
    lr_step = 3L,
    validation = 0.1,
    class_weights = c("balanced", "none"),
    embeddings = NULL,
    seed = 1L,
    verbose = FALSE,
    what = c("predictions", "attention", "hyperedges", "hyperedge_words"),
    holdout = NULL
)

```

```

text_hypergat(
  x,
  labels,
  column = NULL,
  id = NULL,
  stop_words = stop_words_en(),
  min_count = 1L,
  lowercase = TRUE,
  semantic = c("none", "lda"),
  lda_keywords = NULL,
  lda_topics = NULL,
  lda_top_n = 10L,
  lda_max_iter = 10L,
  lda_batch_size = 128L,
  lda_offset = 50,
  lda_decay = 0.7,
  lda_seed = 0L,

```

```

    embed_dim = 300L,
    hidden = 100L,
    epochs = 10L,
    lr = 0.001,
    dropout = 0.3,
    batch_size = 8L,
    weight_decay = 1e-06,
    lr_decay = 0.1,
    lr_step = 3L,
    validation = 0.1,
    class_weights = c("balanced", "none"),
    embeddings = NULL,
    seed = 1L,
    verbose = FALSE,
    what = c("predictions", "attention", "hyperedges", "hyperedge_words"),
    holdout = NULL
)

```

Arguments

x	A character vector of documents (names become ids) or a data.frame with a text column.
labels	The known labels: the name of a column of x (a data.frame) holding each document's label, a named character vector (names are document ids, values class labels) or a tidy data.frame with a node column and a label, cluster or predicted column. At least two classes.
column, id	When x is a data.frame: the text column and the optional id column, as in text_hypergraph() .
stop_words	Words removed before building sentences (default stop_words_en()).
min_count	Minimum corpus frequency for a word to become a vertex: its number of occurrences (after stop-word removal) in the documents the vocabulary is built from, every repeat counted.
lowercase	Lowercase the text first.
semantic	"none" (default) for sentence hyperedges only, or "lda" to append the paper's semantic topic hyperedges.
lda_keywords	Optional precomputed topic keywords: a list of character vectors, one per topic. With semantic = "lda", NULL fits LDA natively; supplying the list bypasses fitting and reproduces a saved official preprocessing run.
lda_topics	Number of LDA topics. NULL (default) uses the number of classes, as in the paper. Ignored when lda_keywords is supplied.
lda_top_n	Number of highest-probability words per fitted topic (paper default 10).
lda_max_iter, lda_batch_size	Online variational-Bayes passes and minibatch size (scikit-learn defaults used by the official code: 10 and 128).
lda_offset, lda_decay	Online learning schedule (official values 50 and 0.7).

lda_seed	LDA initialization seed (official value 0), separate from the neural training seed.
embed_dim, hidden	Embedding and hidden width (official defaults 300 and 100).
epochs, lr, dropout, batch_size, weight_decay	Training hyperparameters (official defaults: 10, 0.001, 0.3, 8, 1e-6).
lr_decay, lr_step	StepLR schedule: multiply the learning rate by lr_decay every lr_step epochs (official 0.1 every 3).
validation	Fraction of the labeled documents held out (stratified) to pick the best epoch; 0 keeps the final weights.
class_weights	"balanced" (default, inverse-frequency loss weights as in the official run script) or "none".
embeddings	Optional pretrained word-vector matrix (rownames are words, embed_dim columns); words not covered keep their random initialization.
seed	Integer seed (R and torch); results are deterministic given a seed.
verbose	Message the loss each epoch.
what	Legacy extraction option. Fit once and use <code>hg_get()</code> with what = "attention", "hyperedges" or "hyperedge_words" instead.
holdout	NULL (default) trains on every given label. A share in (0, 1) hides that share within each class with seed. The fitted object prints its held-out accuracy and balanced accuracy.

Value

An hg_hypergat fitted classifier, also an [hg_classification](#) and a data frame. `print()` reports held-out evaluation when available. `hg_get()` reads predictions, per-class results, confusion, document text and training history. `plot()` shows confusion or training loss. `predict.hg_hypergat()` classifies new documents with the same network and frozen vocabulary.

Attention is computed on request, without training again. Word weights sum to one within each hyperedge; edge weights sum to one over each word's hyperedges. A word in just one edge gives it weight one automatically. These are internal aggregation weights, not class-specific contributions or explanations. The diagnostic tables and their normalization baselines are described in [hg_get.hg_hypergat\(\)](#). Softmax scores are not calibrated probabilities of correctness. The vocabulary is estimated from known labels only; held-out and unlabelled documents cannot change it.

References

Ding, K., Wang, J., Li, J., Li, D., & Liu, H. (2020). Be more with less: Hypergraph attention networks for inductive text classification. *EMNLP 2020*.

Examples

```
# a small, fast fit on 60 bundled forum posts; a real analysis trains
# for more epochs on the whole corpus
fit <- hg_hypergat(head(forum_posts, 60), column = "text",
```

```

                                labels = "topic", holdout = 0.25, epochs = 5)
fit
plot(fit)
hg_get(fit, what = "documents", split = "test", top = 3)
predict(fit, newdata = tail(forum_posts, 3))

```

hg_hypergcn

HyperGCN node classifier

Description

Implements Yadati et al.'s (2019) mediator expansion and two-layer GCN. Every hyperedge selects the pair of vertices farthest apart in the current transformed representation. With `method = "hypergcn"`, the expansion is recomputed at both layers and every epoch (Algorithm 1). `"fast"` fixes it once from the input features (Algorithm 2), and `"one"` uses only the farthest pair without mediators (Algorithm 3). Exact distance ties are resolved lexicographically for reproducibility.

Usage

```

hg_hypergcn(
  hg,
  labels,
  features = "incidence",
  method = c("hypergcn", "fast", "one"),
  hidden = 128L,
  epochs = 200L,
  lr = 0.01,
  weight_decay = 5e-04,
  dropout = 0.5,
  validation = 0.1,
  edge_weights = NULL,
  seed = 1L,
  verbose = FALSE
)

```

Arguments

hg	A text_hypergraph() (or any hypergraphs <code>net_hg</code>), dense or sparse.
labels	The known labels: a named character vector (names are node identifiers, values class labels) or a tidy data.frame with a node column and a label, cluster or predicted column. At least two classes.
features	Vertex feature matrix, one row per node in hg node order (rownames must match the node names), e.g. sbert embeddings. The default <code>"incidence"</code> uses the hypergraph's own weighted incidence rows (tf-idf bag-of-words features for a nodes = <code>"doc"</code> text hypergraph).
method	<code>"hypergcn"</code> , <code>"fast"</code> , or <code>"one"</code> for Algorithms 1–3.

hidden	Hidden width (default 128).
epochs	Training epochs (default 200).
lr, weight_decay	Adam learning rate and L2 penalty.
dropout	Hidden-layer dropout.
validation	Fraction of labeled nodes held out to select an epoch.
edge_weights	Optional positive hyperedge weights.
seed	Reproducible R and torch seed.
verbose	Message loss every 20 epochs.

Value

A prediction data.frame in the same format as [hg_neural\(\)](#), with training history attached.

References

Yadati, N., Nimishakavi, M., Yadav, P., Nitin, V., Louis, A., & Talukdar, P. (2019). HyperGCN: A new method of training graph convolutional networks on hypergraphs. *NeurIPS* 32.

hg_joint_cluster

Joint NMF clustering with an auxiliary vertex relation

Description

Implements the two multi-view objectives in Section 6.3.2 of Hayashi et al. (2020). method = "joint" is Eq. 18 (J-NMF), jointly factorizing the edge-dependent incidence X and a vertex relation S. method = "joint_symmetric" is Eq. 19 (JS-NMF), jointly factorizing the representative-digraph similarity $T = I - L$ and S. The paper uses a symmetric patent-citation indicator for S; any finite non-negative node-by-node relation matrix can be supplied here.

Usage

```
hg_joint_cluster(
  hg,
  relations,
  k,
  method = c("joint", "joint_symmetric"),
  alpha = 1,
  beta = 1,
  gamma = 1,
  type = c("random_walk", "zhou"),
  edge_weights = NULL,
  nstart = 10L,
  seed = NULL,
  max_iter = 500L,
  tol = 1e-06
)
```

Arguments

hg	A connected, dense net_hg (a sparse one raises hypergraphs_sparse_unsupported: the factorization is dense).
relations	A non-negative n_nodes by n_nodes numeric matrix. If it has dimnames, rows and columns are reordered to hg\$nodes.
k	Number of clusters, between 2 and n_nodes - 1.
method	"joint" for J-NMF (Eq. 18) or "joint_symmetric" for JS-NMF (Eq. 19).
alpha	Non-negative symmetry penalty in JS-NMF.
beta	Non-negative agreement penalty between the content and relation vertex factors.
gamma	Non-negative weight of the auxiliary-relation loss.
type, edge_weights	Laplacian inputs for JS-NMF.
nstart	Number of random initializations, a whole number ≥ 1 .
seed	Optional integer initialization seed.
max_iter	Maximum multiplicative-update iterations, a whole number ≥ 1 .
tol	Relative objective tolerance, a positive number. A count or tolerance outside these ranges (including Inf) raises hypergraphs_bad_input.

Value

A net_hg_cluster object. Its \$embedding is the fitted vertex factor M; \$params contains all fitted factors, the objective trace and convergence diagnostics.

References

Hayashi, K., Aksoy, S. G., Park, C. H., & Park, H. (2020). Hypergraph random walks, Laplacians, and clustering. *CIKM 2020*, 495-504. doi:10.1145/3340531.3412034

hg_keywords

Characteristic words (keywords) per cluster

Description

For a clustered hypergraph, ranks each cluster's hyperedges – on a nodes = "doc" `text_hypergraph()`, its words – by one of four scores computed from the hypergraph, or by a score table supplied from another fit. This is the topic-description step that turns a partition from `hg_cluster()` into named topics.

Usage

```

hg_keywords(
  hg,
  clusters,
  n = 10L,
  type = NULL,
  sort_by = c("score", "share"),
  min_docs = 1L,
  centrality = c("pagerank", "clique", "Z", "H"),
  scores = NULL,
  collapse = FALSE
)

## S3 method for class 'hypergraphs_keywords'
print(x, ...)

## S3 method for class 'hypergraphs_keywords'
plot(x, value = c("score", "share"), label = TRUE, ncol = NULL, ...)

```

Arguments

hg	The hypergraph the clustering was computed on, or the sentence hypergraph of the same documents (see <i>Sentence scope</i>).
clusters	The tidy table returned by hg_cluster() (columns node, cluster), a named vector of cluster labels, or the name of a column of the hypergraph's document table, so a group variable of the corpus such as "period" gives its words directly.
n	Keywords per cluster (default 10); Inf returns all.
type	Which score ranks the words: any of "mass", "frequency", "ctfidf", "centrality". Several at once give one block of rows per score. Default "mass", or none when only scores is wanted.
sort_by	"score" (default) ranks a cluster's words by the selected score, the heavy vocabulary; "share" ranks them by the fraction of the word's total score that falls in the cluster, the distinctive vocabulary. Ties are broken by the other column, then alphabetically.
min_docs	Keep only words that occur in at least this many of the cluster's documents (default 1). The support floor that stops sort_by = "share" from surfacing words a cluster owns because they occur once.
centrality	For type = "centrality", the hg_centrality() measure: "pagerank" (default; it reads the incidence weights and does not tie on words present in every document, as the clique measure does), "clique", "Z" or "H".
scores	An external score table (columns node, word and one numeric column of finite values), summed per cluster and added as its own block. Several rows for one document and word add their scores but count as one document in n_docs.
collapse	If TRUE, return one row per type and cluster with the words joined into a single comma-separated string – a display table, not tidy data. Default FALSE.

x	For plot() and print(): a table returned by hg_keywords().
...	Unused; for S3 consistency.
value	For plot(): which column the bars show, "score" (default) or "share"; match it to sort_by.
label	For plot(): print each bar's value at its end (default TRUE).
ncol	For plot(): panels per row; default one column per score type, so clusters run down and types across. With a single type and many clusters, pass e.g. ncol = 4.

Details

- "mass" (default): the summed incidence weight a cluster's documents place on the word (tf-idf mass under weight = "tfidf", counts under weight = "n"). Works on any hypergraph.
- "frequency": the raw token count of the word in the cluster's documents, whatever the incidence weighting.
- "ctfidf": class-based tf-idf (Grootendorst 2022), the topic representation BERTopic uses: the cluster's L1-normalised term frequency times $\log(A / f_w + 1)$, where f_w is the word's corpus count and A the average token count per cluster (truncated to an integer, as in the reference implementation).
- "centrality": the word's centrality within the cluster's own word hypergraph (words as nodes, the cluster's documents as hyperedges, incidence = the stored weights), from [hg_centrality\(\)](#) with the measure named in centrality.

"frequency", "ctfidf" and "centrality" need the token-level layer of a bag-of-words document hypergraph (construction = "bag", nodes = "doc"); other hypergraphs raise hypergraphs_bad_input.

Sentence scope. When hg is a text_hypergraph(construction = "sentence") of the documents, clusters still names documents, and the scores are computed over the cluster's *sentences*: two words are then related by sharing a sentence rather than a document. "mass" and "frequency" sum the in-sentence counts; "centrality" is the word's centrality among the cluster's sentence hyperedges.

External scores. scores takes a data.frame with columns node, word and one numeric column (for instance the attention table from hg_hypermot(what = "attention")). Its values are summed per cluster and reported as their own block, labelled by the numeric column's name.

Value

A base data.frame of class hypergraphs_keywords, one row per type-cluster-keyword triple, columns type, cluster, size (the cluster's documents), rank, word, score (the selected score), share (score divided by the word's summed score over all clusters) and n_docs (the cluster's documents containing the word), ranked by sort_by within type and cluster; only words with a positive score and at least min_docs documents appear, so a cluster with no such word has no rows (and no row in the collapsed view), and a table in which no cluster has one has zero rows and the same columns. With collapse = TRUE: one row per type and cluster, columns type, cluster, size and words. The print method shows the collapsed view, truncated to the console width; the returned table itself is the long form. Raises hypergraphs_bad_input for unknown node names, a type that needs the token layer on a hypergraph without one, a malformed scores table, or a bag-of-words text_hypergraph(nodes = "word"), whose hyperedges are documents rather than words.

`plot()` returns a ggplot: one panel per cluster (rows) and score type (columns), each with its own word axis, horizontal bars of value per word, Okabe-Ito fill by type. It needs the long form and raises `hypergraphs_bad_input` on a collapsed table.

References

Grootendorst, M. (2022). BERTopic: Neural topic modeling with a class-based TF-IDF procedure. arXiv:2203.05794.

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots",
  cooking_2 = "this soup recipe needs salt on a cold night",
  space_1 = "the telescope revealed a distant galaxy and stars",
  space_2 = "astronomers aimed the telescope at the stars all night"
), stop_words = c("the", "with", "and", "a", "this", "at", "on", "all"))
topics <- hg_cluster(hg, k = 2, seed = 1)
hg_keywords(hg, topics, n = 3)
hg_keywords(hg, topics, n = 3, type = "ctfidf")
words <- hg_keywords(hg, topics, n = 4,
  type = c("frequency", "ctfidf", "centrality"))
plot(words)
# distinctive rather than heavy vocabulary: rank by share, with support
hg_keywords(hg, topics, n = 3, sort_by = "share", min_docs = 1)
```

hg_landscape

Persistence landscape of a persistence diagram

Description

The persistence landscape: at every filtration value t , the k -th largest of the tent functions $\max(0, \min(t - b, d - t))$ of the diagram's (birth, death) pairs in one homology dimension. Essential classes, which never die, are left out.

Usage

```
hg_landscape(ph, k_max = 5L, dimension = 1L, t_grid = NULL)
```

Arguments

<code>ph</code>	A <code>hg_homology()</code> result, or a data.frame with columns <code>dimension</code> , <code>birth</code> , <code>death</code> .
<code>k_max</code>	Highest landscape level. Default 5.
<code>dimension</code>	Homology dimension. Default 1.
<code>t_grid</code>	Evaluation points. NULL uses 200 points spanning the pairs.

Value

A persistence_landscape object. Read it with [hg_get\(\)](#): one row per level and grid point (what = "landscape").

References

Bubenik, P. (2015). Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16, 77–102.

See Also

[hg_homology\(\)](#)

Examples

```
set.seed(1)
w <- matrix(runif(36), 6, 6, dimnames = list(letters[1:6], letters[1:6]))
w <- (w + t(w)) / 2
diag(w) <- 0
pl <- hg_landscape(hg_homology(w, n_steps = 10), dimension = 0)
hg_get(pl, k = 1, top = 5)
```

hg_laplacian

Normalized hypergraph Laplacian

Description

Computes the normalized Laplacian of a hypergraph, either the classic Zhou-Huang-Scholkopf form on the binary incidence pattern (type = "zhou") or the random-walk form with edge-dependent vertex weights (type = "random_walk"), in which the weighted incidence cells (e.g. the summed weights produced by [group_hypergraph\(\)](#)) determine where a random walker lands inside a hyperedge, and the resulting non-reversible walk is symmetrized through its stationary distribution (Chung 2005). Both Laplacians are symmetric positive semi-definite with eigenvalues in $[0, 2]$; for a binary incidence with unit hyperedge weights they coincide.

Usage

```
hg_laplacian(hg, type = c("zhou", "random_walk"), edge_weights = NULL)
```

Arguments

hg	A net_hg from network_hypergraph() or group_hypergraph() , dense or sparse (sparse = TRUE). Must be connected and have at least one hyperedge. Empty hyperedges (which random_hypergraph() can produce) contribute nothing.
type	Character. "zhou" (default) for the Zhou et al. (2006) normalized Laplacian on the binary incidence pattern, or "random_walk" for the Hayashi et al. (2020) EDVW random-walk Laplacian on the weighted incidence.

edge_weights A single positive number (recycled) or a numeric vector of positive hyperedge weights (length `hg$n_hyperedges`), or NULL for the default. Hypergraphs built by `window_hypergraph()` default to their window counts (for both types). Otherwise the default is type-specific: unit weights for "zhou"; for "random_walk" the Hayashi et al. heuristic - the population standard deviation of each hyperedge's (non-zero) vertex weights plus one - which reduces to unit weights on a binary incidence.

Value

A symmetric `n_nodes` x `n_nodes` numeric matrix (node names as `dimnames`) with attributes `type` (the Laplacian type), `pi` (named stationary distribution of the underlying random walk) and `edge_weights` (the hyperedge weights, one per hyperedge; an empty hyperedge's default weight is 1 and unused). For a sparse hypergraph the matrix is a sparse symmetric `Matrix` (class `dsCMatrix`) with the same `dimnames` and attributes, and the random-walk stationary distribution is found by power iteration rather than a dense eigendecomposition. A disconnected hypergraph raises `hypergraphs_hypergraph_disconnected`.

References

Zhou, D., Huang, J., & Scholkopf, B. (2006). Learning with hypergraphs: Clustering, classification, and embedding. *NeurIPS 19*. doi:10.7551/mitpress/7503.003.0205

Hayashi, K., Aksoy, S. G., Park, C. H., & Park, H. (2020). Hypergraph random walks, Laplacians, and clustering. *CIKM 2020*, 495-504. doi:10.1145/3340531.3412034

Chung, F. (2005). Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1), 1-19.

Examples

```
events <- data.frame(
  person = c("a", "b", "c", "a", "b", "d", "c", "d", "e", "e", "a"),
  meeting = c("m1", "m1", "m1", "m2", "m2", "m2", "m3", "m3", "m3",
              "m4", "m4"),
  hours = c(2, 1, 1, 3, 2, 1, 2, 2, 4, 1, 1)
)
hg <- group_hypergraph(events, node = "person", hyperedge = "meeting",
                       weight = "hours")
L <- hg_laplacian(hg, type = "random_walk")
range(eigen(L, symmetric = TRUE, only.values = TRUE)$values)
```

hg_line_graph

The s-line graph of a hypergraph

Description

Turns the hyperedges into vertices: two hyperedges are adjacent when they share at least `s` vertices, and the edge weight is how many they share. `s = 1` is the ordinary line graph (any overlap connects); raising `s` keeps only substantial overlaps, which is how walk-based measures on hyperedges are parametrised (Aksoy et al. 2020).

Usage

```
hg_line_graph(hg, s = 1, what = c("edges", "matrix"))
```

Arguments

hg	A text_hypergraph() , knn_hypergraph() , or any hypergraphs <code>net_hg</code> .
s	Minimum number of shared vertices for two hyperedges to be adjacent. A single integer of at least 1; 1 (default) is the ordinary line graph.
what	"edges" (default) for the tidy edge list, or "matrix" for the symmetric overlap matrix to hand to a graph engine.

Details

This is the projection of the dual: `hg_line_graph(hg, s = 1)` returns the same pairs as `pairwise_network(dual_hypergraph(hg), weighted = FALSE)`, which the tests assert.

Value

With `what = "edges"`, a base `data.frame` with one row per unordered hyperedge pair sharing at least `s` vertices, sorted by `from` then `to`, with columns `from`, `to` (hyperedge names) and `weight` (the number of shared vertices). A zero-row `data.frame` with those columns when no pair reaches `s`. With `what = "matrix"`, the symmetric `n_hyperedges` x `n_hyperedges` overlap matrix, zero on the diagonal and wherever the overlap is below `s`, sparse if `hg`'s incidence is sparse.

References

Aksoy, S. G., Joslyn, C., Ortiz Marrero, C., Praggastis, B., & Purvine, E. (2020). Hypernetwork science via high-order hypergraph walks. *EPJ Data Science*, 9(1), 16. doi:10.1140/epjds/s13688-020002310

See Also

[pairwise_network\(\)](#) for the projection onto vertices, [dual_hypergraph\(\)](#) for the role swap itself.

Examples

```
hg <- text_hypergraph(c(a = "salt and soup", b = "soup and stars",
                        c = "stars and salt"))
hg_line_graph(hg)
hg_line_graph(hg, s = 2)
```

hg_markov_stability *Markov stability of the states of a transition network*

Description

Per-state persistence (self-transition probability), stationary probability, return time, sojourn time and mean first-passage times to and from the other states of a first-order Markov chain.

Usage

```
hg_markov_stability(
  data,
  normalize = TRUE,
  action = NULL,
  actor = NULL,
  time = NULL,
  session = NULL,
  time_threshold = 900,
  timezone = "UTC"
)
```

Arguments

data	A row-stochastic transition matrix, a network object (netobject, tna, cograph_network, netobject_group) whose weights are the transition matrix, or sequences in any form described in sequence-input , from which the first-order transition probabilities are estimated.
normalize	Logical. Normalise the rows to sum to one. Default TRUE.
action, actor, time, session, time_threshold, timezone	Long-format arguments (see sequence-input); leave the column names NULL for wide, list or model input.

Value

A net_markov_stability object. Read it with [hg_get\(\)](#): one row per state (what = "states", the default), the mean first-passage times ("passage_time") or the stationary distribution ("stationary").

Conditions

The quantities are defined only for an irreducible chain, one in which every state can reach every other. A chain that is not irreducible (a transient or absorbing state, or several closed classes, as a pruned higher-order network can have) raises hypergraphs_not_ergodic, a hypergraphs_bad_input, naming those states.

References

Kemeny, J. G., & Snell, J. L. (1976). *Finite Markov Chains*. Springer.

See Also[markov_order\(\)](#)**Examples**

```
P <- matrix(c(0.5, 0.3, 0.2,
             0.2, 0.5, 0.3,
             0.1, 0.4, 0.5), 3, 3, byrow = TRUE,
           dimnames = list(c("A", "B", "C"), c("A", "B", "C")))
ms <- hg_markov_stability(P)
hg_get(ms, sort_by = "stationary")
hg_get(ms, what = "passage_time", from = "A")
```

hg_membership

*Soft topic membership from the spectral embedding***Description**

Turns a hard partition into a membership of every document in every topic. The documents are placed in the spectral embedding of the hypergraph Laplacian that [hg_cluster\(\)](#) cuts (the same type and edge_weights), each topic's centre is the mean position of its documents, and the membership of a document in a topic is the fuzzy c-means weight with fuzziness 2: the inverse squared distance to that centre, normalised over the topics. A document at a centre has membership 1 there (shared equally when several centres coincide); a document halfway between two centres has 0.5 in each. The values sum to one over the topics, so their level depends on the number of topics k: the uniform value is 1 / k, and with many topics even a clearly assigned document has a modest membership. Read them as ratios between topics (a document with 0.15 and 0.11 in two topics is 1.4 times closer to the first), not as probabilities of belonging.

Usage

```
hg_membership(
  hg,
  clusters,
  type = c("zhou", "random_walk"),
  edge_weights = NULL
)

## S3 method for class 'hypergraphs_membership'
plot(x, ...)
```

Arguments

hg The document hypergraph the clustering was computed on.

clusters The tidy table returned by [hg_cluster\(\)](#) (columns node, cluster), or a named vector of cluster labels.

type, edge_weights	Passed to <code>hg_cluster()</code> to reproduce the embedding the partition was cut in (defaults as there).
x	A table returned by the verb.
...	Unused; for S3 consistency.

Value

A base `data.frame` of class `hypergraphs_membership`, one row per document and topic: `node`, `cluster` (the hard label), `topic`, `membership` (rows of one document sum to one). `plot()` draws, per topic, the distribution of its documents' membership in it: a topic whose documents sit near 1 is compact, one whose documents spread towards 0.5 overlaps its neighbours. Raises `hypergraphs_bad_input` for unknown node names.

References

Bezdek, J. C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum.

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots",
  cooking_2 = "this soup recipe needs salt on a cold night",
  space_1 = "the telescope revealed a distant galaxy and stars",
  space_2 = "astronomers aimed the telescope at the stars all night"
), stop_words = c("the", "with", "and", "a", "this", "at", "on", "all"))
topics <- hg_cluster(hg, k = 2, seed = 1)
hg_membership(hg, topics)
```

hg_mmsbm

Mixed-membership communities of a hypergraph (Hy-MMSBM)

Description

Fits the hypergraph mixed-membership stochastic block model of Ruggeri et al. (2023), which gives every node a membership vector over k communities from a likelihood model of the whole hypergraph. The weight A_e of a hyperedge e is Poisson with mean $\lambda_e/\kappa_{|e|}$, where

$$\lambda_e = \sum_{i < j \in e} u_i^\top w u_j,$$

u is the non-negative $N \times k$ membership matrix, w the symmetric non-negative $k \times k$ affinity between communities and $\kappa_n = \frac{n(n-1)}{2} \binom{N-2}{n-2}$ normalises for the size n of the hyperedge. Summed over every possible hyperedge up to the largest size D , the normalisers give the constant $C = \sum_{n=2}^D 2/(n(n-1)) = 2(1 - 1/D)$ and the log-likelihood (Eq. 5, up to additive constants that do not depend on u, w)

$$\mathcal{L} = -C \sum_{i < j} u_i^\top w u_j + \sum_e A_e \log \lambda_e.$$

Usage

```

hg_mmsbm(
  hg,
  k,
  assortative = FALSE,
  nstart = 10L,
  max_iter = 5000L,
  tol = 1e-05,
  check_every = 10L,
  criterion = c("membership", "parameters"),
  w_prior = 1,
  u_prior = 0,
  max_size = NULL,
  edge_weights = NULL,
  seed = NULL,
  parallel = FALSE,
  n_cores = 2L
)

## S3 method for class 'net_hg_mmsbm'
hg_get(
  x,
  what = c("membership", "nodes", "affinity", "restarts"),
  ...,
  top = NULL
)

## S3 method for class 'net_hg_mmsbm'
print(x, n = 10L, ...)

## S3 method for class 'net_hg_mmsbm'
plot(x, type = c("membership", "affinity", "restarts"), labels = NULL, ...)

```

Arguments

hg	A hypergraphs net_hg (text_hypergraph() , group_hypergraph() , window_hypergraph() , ...).
k	Number of communities, a whole number between 1 and the number of nodes.
assortative	If TRUE, w is diagonal (communities interact only within themselves); default FALSE, the full affinity.
nstart	Number of random starts (default 10L, the authors' --training_rounds).
max_iter	Maximum EM steps per start (default 5000L).
tol	Convergence tolerance (default 1e-5), compared with the change between two consecutive steps every check_every steps. A start that hits max_iter first is reported as not converged; if the kept start did not converge, a hypergraphs_no_converge warning is raised.
check_every	Steps between convergence checks (default 10L, as the authors').

criterion	What must stop changing: "membership" (default), the largest absolute change of any membership; or "parameters", the authors' rule, the Frobenius change of w divided by k and of u divided by the number of nodes. The likelihood is unchanged when u is multiplied by c and w divided by c^2 ; with the default w_prior and no u_prior the fit keeps drifting along that direction (w shrinks, u grows) while the memberships and the log-likelihood have settled, so "parameters" rarely fires there.
w_prior, u_prior	Rates of the exponential priors on w and u (MAP estimation); 0 means none (maximum likelihood). The defaults $w_prior = 1$, $u_prior = 0$ are the authors' "half-Bayesian" setting that fixes the scale of the parameters. As in the authors' code, the w prior acts on Cw (the parametrisation used during the iterations).
max_size	NULL (default: every hyperedge) or the largest hyperedge size kept; it is also the D in C . With NULL, D is the largest size in the data.
edge_weights	NULL or positive hyperedge weights A_e , one per hyperedge or one recycled. NULL uses the window counts of a <code>window_hypergraph()</code> and 1 otherwise. The incidence is read as binary: the stored cell weights (word counts, tf-idf) do not enter the model.
seed	NULL (default: the current random stream) or a whole number for the random starts; the caller's stream is restored on exit.
parallel	Logical. Fit the <code>nstart</code> starts with <code>parallel::mclapply()</code> (not on Windows, where they run serially). Default FALSE. All starting values are drawn first, in order, from one stream, so the result equals the serial one.
n_cores	Integer. Cores when <code>parallel = TRUE</code> (default 2).
x	A <code>net_hg_mmsbm</code> object.
what	Which table (see Value).
...	Unused; for S3 consistency.
top	NULL (default, every row) or the number of first rows to return.
n	Number of rows of the default table to print. Default 10.
type	For <code>plot()</code> : "membership" (default) stacks each node's memberships in a bar, nodes grouped in panels by hard community; "affinity" draws w as a labelled tile map; "restarts" the log-likelihood of every start, shaped by convergence.
labels	For <code>plot(type = "membership")</code> : show node names on the axis (default: when there are at most 40 nodes).

Details

The fit is the authors' expectation-maximisation (Eqs. 8-9, with the exponential priors of their Appendix A), ported from their reference code: in each step w is updated, then u ; C is absorbed into w during the iterations and divided out at the end. Every quantity goes through $\lambda_e = (s_e^\top w s_e - \sum_{i \in e} u_i^\top w u_i)/2$ with $s_e = \sum_{i \in e} u_i$ and sparse products with the incidence matrix, so one step costs $O(\text{nnz } k + Ek^2)$ and hyperedges of any size (a word used in every document) are handled exactly. EM finds a local optimum only: the fit is run from `nstart` random starts and the one with the highest $\mathcal{L}$ is kept (Algorithm 1). The restarts table reports the log-likelihood, the iterations, the

convergence flag of every start, and the adjusted Rand index of its hard partition with the kept one, so the stability of the solution is visible.

Hyperedges of size one carry no pair and are outside the model (their λ_e is zero); they are left out and counted. `max_size` optionally leaves out larger hyperedges too, as the authors' `--max_hye_size`; the default NULL keeps them all. A node left with no hyperedge has $u_i = 0$ after the first step and no membership (NA), with a `hypergraphs_isolated_nodes` warning. A node that does lie in fitted hyperedges can still have its memberships driven to zero (or to subnormal remnants) by the multiplicative EM updates; any row whose total is below the precision of the largest row is treated as having no membership (NA) and reported with a `hypergraphs_collapsed_membership` warning, rather than normalising remnants into spurious exact mixtures. This is a property of the model, not of the fit: λ_e sums over the pairs of a hyperedge, so in a hyperedge of three or more nodes the pairs that avoid node i can explain it on their own, and $u_i = 0$ then costs the likelihood nothing while removing its share of the $-C \sum_{i < j}$ term. Documents bound by word hyperedges, nearly all of size three or more, are the common case: on a 6,630-document legal corpus about a third of the documents collapse for $k = 3$, with or without stop words and with `max_size = 25`. For the topic mixture of every document of a corpus use `hg_topics()`.

The membership of node i in community k is $u_{ik} / \sum_q u_{iq}$; the hard community is its largest entry. The size of u_i reflects how active the node is, its direction the communities it mixes; at the maximum of the likelihood many memberships sit at or near 0 and 1, and mixing is read from the nodes that do not. Communities are numbered by decreasing total membership. On a `text_hypergraph()` with documents as nodes, the communities are topics and the memberships the documents' topic shares.

Value

An object of class `net_hg_mmsbm`, a list read through `hg_get(x, what = )`:

- "membership" (default): one row per node and community, columns node, community, membership (sums to one over a node's rows) and membership_weight (the fitted, unnormalised membership parameter u);
- "nodes": one row per node, node, community (the largest membership) and membership (its value);
- "affinity": one row per community pair with from $\leq$ to, columns from, to, affinity (the fitted w);
- "restarts": one row per start, run, log_likelihood, iterations, converged, best, ari_to_best (adjusted Rand index of the start's hard partition with the kept one).

`print()` gives the fit, `summary()` one row per community (community, size hard-assigned nodes, mass total membership, mean_membership of its assigned nodes, `w_within`), and `plot()` the memberships, the affinity or the restarts. Raises `hypergraphs_bad_input` for a non-hypergraph, an invalid argument or no hyperedge of size two or more.

References

- Ruggeri, N., Contisciani, M., Battiston, F., & De Bacco, C. (2023). Community detection in large hypergraphs. *Science Advances*, 9(28), eadg9159. doi:10.1126/sciadv.adg9159
- Hubert, L., & Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2, 193–218. doi:10.1007/BF01908075

See Also

[hg_membership\(\)](#) for soft membership read from a hard spectral partition.

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots and salt",
  cooking_2 = "this soup recipe needs onions, carrots and salt",
  cooking_3 = "carrots and onions make a sweet soup",
  space_1 = "the telescope revealed a distant galaxy and stars",
  space_2 = "astronomers aimed the telescope at the stars",
  space_3 = "a distant galaxy full of stars",
  both = "astronomers eat soup with carrots under the stars"
), stop_words = c("the", "with", "and", "a", "this", "at", "of", "under"))
fit <- hg_mmsbm(hg, k = 2, seed = 1)
fit
hg_get(fit, what = "nodes")
hg_get(fit, what = "restarts")
```

hg_modularity

Hypergraph modularity of a partition

Description

Scores a partition of the nodes of a hypergraph with the hypergraph modularity of Kaminski et al. (2019): the weighted share of hyperedges that a community "owns" (the edge contribution) minus the share expected under the generalised Chung-Lu null model that preserves weighted degrees and hyperedge sizes (the degree tax). The hypergraph is read as a family of node sets: cell values of the incidence are ignored, only membership and one weight per hyperedge count.

Usage

```
hg_modularity(
  hg,
  partition,
  type = c("linear", "majority", "strict"),
  edge_weights = NULL,
  what = c("score", "communities")
)
```

Arguments

hg	A net_hg.
partition	The partition to score: a data.frame with node and a label column (community, cluster, label or predicted), a named label vector, an unnamed vector in node order, or an hg_communities() fit (its AMI medoid is scored). Every node must be labelled, once: a node listed twice with the same label counts once, with different labels it raises hypergraphs_bad_input.

type	Hyperedge weighting $\omega(d, c)$: "linear" (default), "majority" or "strict".
edge_weights	Positive hyperedge weights (one per hyperedge, or one value recycled). NULL uses the window counts of a window_hypergraph() , else unit weights.
what	"score" (default; one row) or "communities" (one row per community; its modularity column sums to the score).

Details

A hyperedge of size d whose largest part holds c of its nodes contributes $w_e \omega(d, c)$ to the community holding that part when $c > d/2$. The three types are HyperNetX's wdc functions: "strict" ($\omega = 1$ only when the whole edge lies in one community, the strict modularity of Kaminski et al. 2019), "majority" ($\omega = 1$ when a strict majority does), and "linear" ($\omega = c/d$ for a strict majority, the HyperNetX default; Kaminski, Pralat & Theberge 2020). The degree tax of community A is $\sum_d W_d \sum_{c > d/2} \omega(d, c) \text{Binom}(c; d, \text{vol}(A)/\text{vol}(V))$, with W_d the total weight of d -edges and vol the weighted degree. Both terms are divided by the total hyperedge weight.

Value

A base data.frame. For what = "score": one row with type, modularity, edge_contribution, degree_tax (both already divided by total_weight, so modularity = edge_contribution - degree_tax), n_communities, n_edges (non-empty hyperedges) and total_weight. For what = "communities": one row per community with community, n_nodes, volume (summed weighted degree), edge_contribution, degree_tax and modularity, sorted by decreasing modularity. Raises hypergraphs_bad_input for a non-net_hg input, a partition that misses a node, has NA labels or conflicting repeated nodes, or a hypergraph with no non-empty hyperedge; invalid edge_weights fail the shared weight check.

References

- Kaminski, B., Poulin, V., Pralat, P., Szufel, P., & Theberge, F. (2019). Clustering via hypergraph modularity. *PLoS ONE*, 14(11), e0224307. doi:10.1371/journal.pone.0224307
- Kaminski, B., Pralat, P., & Theberge, F. (2020). Community detection algorithm using hypergraph modularity. In *Complex Networks & Their Applications IX*, Studies in Computational Intelligence 943, 152-163. doi:10.1007/9783030653477_13

See Also

[hg_communities\(\)](#) with type = "irmm" to find a partition.

Examples

```
dat <- data.frame(
  member = c("a", "b", "c", "a", "b", "c", "x", "y", "z", "x", "y", "z",
            "c", "x"),
  edge = c(rep(paste0("e", 1:4), each = 3), "e5", "e5")
)
h <- group_hypergraph(dat, "member", "edge")
split <- data.frame(node = c("a", "b", "c", "x", "y", "z"),
  community = c(1, 1, 1, 2, 2, 2))
```

```

hg_modularity(h, split)
hg_modularity(h, split, type = "strict")
hg_modularity(h, split, what = "communities")

```

hg_motifs

Y, T and O motif census with a hypergraph configuration null

Description

Counts the three possible induced four-node motifs in a simple 3-uniform hypergraph: Y contains two of the four possible triples, T contains three, and O contains all four. Repeated identical hyperedges are collapsed for the motif census, matching HypergraphX 1.5. The null repeatedly reshuffles pairs of equal-size hyperedges while preserving every node degree and edge cardinality before duplicate-edge collapse.

Usage

```

hg_motifs(
  hg,
  n = 1000L,
  seed = NULL,
  what = c("test", "counts", "draws"),
  alternative = c("two_sided", "greater", "less"),
  start = NULL,
  end = NULL,
  step = NULL,
  window = NULL,
  at = NULL,
  snapshot_mode = c("active", "cumulative"),
  multiedges = TRUE
)

## S3 method for class 'hypergraphs_motifs'
hg_get(x, what = c("test", "draws"), ...)

## S3 method for class 'hypergraphs_motifs'
plot(x, motif = c("Y", "T", "O"), ...)

```

Arguments

hg	A 3-uniform net_hg or a temporal_hypergraph() .
n	Number of configuration-model draws. The paper uses 1000.
seed	Optional reproducibility seed; the caller's RNG state is restored on exit.
what	"test" for observed/null summaries, "counts" for observed counts only, or "draws" for every null count.
alternative	Empirical permutation-test direction.

start, end, step, window, at
 Measurement grid passed to `hg_snapshots()` when hg is temporal.

snapshot_mode, multiedges
 Snapshot mode ("active" or "cumulative") and multi-edge handling passed to `hg_snapshots()` when hg is temporal.

x
 A hypergraphs_motifs test table.

...
 Unused; for S3 consistency.

motif
 Which motif's null distribution to draw: "Y" (default), "T" or "O".

Value

A tidy data frame. The test table has one row per motif: motif, count (observed), expected and null_sd (mean and standard deviation of the null counts), z, p_value (empirical), delta (relative abundance), normalized_delta (the motif profile used by HypergraphX), n_null (null draws) and method. Temporal input adds a leading time column, the snapshot time: a date for a calendar hypergraph, the number on its clock otherwise.

For `hg_get()`, the test table (what = "test") or every null count (what = "draws", columns run, motif, count) as a plain data.frame.

For plot, a ggplot object: the null distribution of the motif count as a histogram with the observed count as a vertical line and the z-score annotated (the paper's Figure 7).

References

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. doi:10.1098/rsta.2023.0141

Examples

```
dat <- data.frame(
  member = c(1, 2, 3, 1, 2, 4), edge = rep(c("e1", "e2"), each = 3)
)
h <- group_hypergraph(dat, "member", "edge")
hg_motifs(h, what = "counts")
```

hg_neural

Hypergraph neural network classifier (HGNN)

Description

Trains the two-layer hypergraph convolutional network of Feng et al. (2019) on a hypergraph: each layer propagates linearly transformed vertex features through $G = D_v^{-1/2} H W D_e^{-1} H^T D_v^{-1/2}$ (the Zhou similarity operator), with ReLU and dropout between layers and a cross-entropy loss on the labeled vertices. Needs the suggested **torch** package.

Usage

```
hg_neural(
  hg,
  labels,
  features = "incidence",
  hidden = 128L,
  epochs = 600L,
  lr = 0.01,
  weight_decay = 5e-04,
  dropout = 0.5,
  validation = 0.1,
  edge_weights = NULL,
  seed = 1L,
  verbose = FALSE
)
```

Arguments

hg	A text_hypergraph() (or any hypergraphs <code>net_hg</code>), dense or sparse.
labels	The known labels: a named character vector (names are node identifiers, values class labels) or a tidy data.frame with a node column and a label, cluster or predicted column. At least two classes.
features	Vertex feature matrix, one row per node in hg node order (rownames must match the node names), e.g. <code>sbert</code> embeddings. The default "incidence" uses the hypergraph's own weighted incidence rows (tf-idf bag-of-words features for a nodes = "doc" text hypergraph).
hidden	Width of the hidden layer.
epochs	Training epochs (Adam).
lr, weight_decay	Adam learning rate and L2 penalty. The defaults (<code>lr = 0.01</code> , <code>epochs = 600</code>) were selected by validation accuracy on held-out seeds for high-dimensional sparse text features; the original paper's <code>lr = 0.001</code> , 200 epochs underfits that regime.
dropout	Dropout rate after the hidden layer.
validation	Fraction of the labeled seeds held out (stratified) to pick the best epoch; 0 trains on all seeds for epochs and keeps the final weights.
edge_weights	Optional positive hyperedge weights <code>W</code> (one per hyperedge); default all 1, the paper's initialization.
seed	Integer seed for weight initialization, dropout and the validation split (torch and R RNGs); results are deterministic given a seed.
verbose	If TRUE, message the loss every 20 epochs.

Value

A base data.frame, one row per node, with columns `node`, `label` (the given label or NA), `predicted`, `score` (softmax probability of the winning class) and `margin` (winner minus runner-up probability). The training history is attached as attribute "history" (data.frame: epoch, loss, val_accuracy).

References

Feng, Y., You, H., Zhang, Z., Ji, R., & Gao, Y. (2019). Hypergraph neural networks. *AAAI* 33.

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots",
  cooking_2 = "this soup recipe needs salt on a cold night",
  space_1 = "the telescope revealed a distant galaxy and stars",
  space_2 = "astronomers aimed the telescope at the stars all night"
), stop_words = c("the", "with", "and", "a", "this", "at", "on", "all"))
hg_neural(hg, labels = c(cooking_1 = "cooking", space_1 = "space"),
  hidden = 8, epochs = 50, validation = 0)
```

hg_null_test

Degree-preserving null-model test of hypergraph structure

Description

Tests whether an observed structural statistic exceeds what the degree sequences alone imply. The null model fixes both margins of the binary membership (every vertex keeps its hyperdegree, every hyperedge its size) and randomizes the memberships by checkerboard swaps (Gotelli 2000) – a sequential MCMC with burn-in $10 * \text{nnz}$ swap attempts and thinning nnz between samples, nnz being the number of memberships. Statistics are evaluated on the binarized hypergraph (weights carry no meaning under this null), with the definitions `hg_measures()` uses: "density", "avg_edge_size" and "pairwise_participation" equal the values of `hg_measures(hg, what = "summary")` (density is $m / \text{choose}(n, k)$ for a k -uniform hypergraph, $\text{sum}(|e|) / (n * m)$ otherwise).

Usage

```
hg_null_test(
  hg,
  statistic = c("pairwise_participation", "density", "avg_edge_size", "avg_jaccard",
    "repeated_edges", "repeated_pairs"),
  method = c("swap", "configuration", "assignment"),
  n = 199L,
  seed = NULL,
  alternative = c("two_sided", "greater", "less")
)
```

Arguments

hg	A <code>text_hypergraph()</code> , <code>knn_hypergraph()</code> , or any hypergraphs <code>net_hg</code> .
statistic	Statistics to test; any of "pairwise_participation", "density", "avg_edge_size", "avg_jaccard" (mean pairwise edge Jaccard), "repeated_edges" (hyperedges whose member set already occurred: the paper's repeated collaboration trios)

	and "repeated_pairs" (member pairs counted with multiplicity minus the distinct co-occurring pairs: its repeated collaboration pairs). Several allowed.
method	Which null. "swap" (default) is the checkerboard swap chain above: both margins are preserved <i>exactly</i> , and no vertex may repeat within a hyperedge. "configuration" draws independent stub-matchings (Chodrow 2020), the standard higher-order configuration model: vertex stubs are matched uniformly at random to hyperedge slots, so margins hold only up to collapse (a vertex whose stubs land twice in one hyperedge loses a degree). "assignment" is the degree-ordered assignment Coupette et al. (2024) use for their repeated-collaboration test: vertices in decreasing degree each choose their hyperedges without replacement among those with a free slot, so both margins hold exactly and no vertex repeats within a hyperedge, at the price of a draw that is not uniform over configurations. "swap" is the stricter null; use "configuration" to compare against the higher-order network literature, which reports it.
n	Number of null samples: one whole number ≥ 19 (default 199L).
seed	Seed for the null draws; set it for a reproducible test. The global RNG state is restored on exit.
alternative	"two_sided" (default), "greater", or "less".

Details

The permutation p-value is $(1 + \text{extreme}) / (n + 1)$ (Phipson & Smyth 2010), two-sided by default around the null mean; null_lo/null_hi are the 2.5% and 97.5% null quantiles – report them with the observed value, not the p-value alone.

Value

A base data.frame, one row per statistic: statistic, observed, null_mean, null_lo, null_hi, z, p_value, n, method.

Conditions

Raises hypergraphs_bad_input for broken contracts, including an n that is not one whole number ≥ 19 . A statistic the hypergraph cannot define – "avg_edge_size" without hyperedges, "avg_jaccard" with fewer than two hyperedges – is undefined in every null draw as well (the nulls keep the vertex and hyperedge counts), so its row is NA in every column but statistic, n and method, with a hypergraphs_undefined_statistic warning; the other requested statistics are tested as usual. method = "configuration" signals a hypergraphs_configuration_collapse warning when stub matching collapses more than 1% of memberships on average, which happens whenever hyperedges are large relative to the vertex set – the usual case in the document orientation.

References

- Gotelli, N. J. (2000). Null model analysis of species co-occurrence patterns. *Ecology*, 81(9).
- Phipson, B., & Smyth, G. K. (2010). Permutation p-values should never be zero. *Statistical Applications in Genetics and Molecular Biology*, 9(1).
- Chodrow, P. S. (2020). Configuration models of random hypergraphs. *Journal of Complex Networks*, 8(3), cnaa018. doi:10.1093/comnet/cnaa018

Examples

```
hg <- text_hypergraph(c(
  a = "salt and soup and onions",
  b = "soup and salt",
  c = "stars and sky and salt",
  d = "stars and sky"
))
hg_null_test(hg, statistic = "pairwise_participation", n = 49, seed = 1)
```

hg_pagerank

Hypergraph PageRank with edge-dependent vertex weights

Description

Ranks the vertices of a weighted hypergraph by the stationary importance of the Chitra & Raphael (2019) random walk: from a vertex, pick an incident hyperedge with probability proportional to its edge weight, then land on a member vertex with probability proportional to that vertex's *edge-dependent* incidence weight (for a text hypergraph, e.g., the tf-idf of the word in the document). Damping and an optional personalization vector turn the stationary distribution into PageRank (Page et al. 1999): with probability damping the walker follows the hypergraph walk, otherwise it teleports.

Usage

```
hg_pagerank(
  hg,
  damping = 0.85,
  personalized = NULL,
  edge_weights = NULL,
  sort_by = NULL,
  n = Inf,
  max_iter = 1000L,
  tol = 1e-12
)
```

Arguments

hg	A <code>text_hypergraph()</code> , <code>knn_hypergraph()</code> , or any hypergraphs <code>net_hg</code> (connected when <code>damping = 1</code>).
damping	Probability of following the hypergraph walk (default 0.85); 1 - damping is the teleport probability. Must be in (0, 1]; damping = 1 gives the pure stationary distribution, which is unique only on a connected hypergraph: a disconnected one raises <code>hypergraphs_hypergraph_disconnected</code> .
personalized	Optional restart preference: a character vector of vertex names (uniform teleport over exactly those vertices; a name given twice counts once), or a named vector of finite non-negative teleport weights over (a subset of) the vertex names, each name at most once; unnamed vertices get teleport probability 0. NULL (default) teleports uniformly.

edge_weights	Positive hyperedge weights (one per hyperedge), or NULL (default): a window hypergraph's window counts when present, otherwise the Hayashi et al. heuristic used by the Laplacian engines (the population standard deviation of each edge's non-zero vertex weights plus one, which reduces to unit weights on a binary incidence).
sort_by	NULL (default, vertex order) or "pagerank" to sort descending (ties broken by vertex name).
n	Return only the first n rows after sorting: a whole number of at least 1, or Inf (default) for all.
max_iter, tol	Power-iteration cap (a whole number of at least 1) and L1 convergence tolerance (a positive number).

Details

Chitra & Raphael prove that with edge-*independent* vertex weights the walk collapses to a random walk on a weighted clique-expansion graph — so the hypergraph brings genuinely new information exactly when the incidence weights differ across the hyperedges a vertex belongs to (tested against the closed-form graph stationary distribution). With `damping = 1` and default `edge_weights`, the result equals the stationary distribution of the Hayashi et al. (2020) EDVW walk used by `hg_cluster()` (tested at $1e-12$ against `hg_laplacian()`), and with uniform teleportation it equals `hg centrality(type = "pagerank")` (tested).

Value

A base `data.frame`, one row per vertex, with columns `node` and `pagerank` (non-negative, summing to 1).

Conditions

Raises `hypergraphs_bad_input` for broken contracts (including a non-finite, duplicated or unknown personalized entry), `hypergraphs_hypergraph_disconnected` for `damping = 1` on a disconnected hypergraph, and warns with `hypergraphs_no_converge` (returning the last iterate) when `max_iter` is reached before `tol`.

References

- Chitra, U., & Raphael, B. J. (2019). Random walks on hypergraphs with edge-dependent vertex weights. *ICML 2019*.
- Hayashi, K., Aksoy, S. G., Park, C. H., & Park, H. (2020). Hypergraph random walks, Laplacians, and clustering. *CIKM 2020*. doi:10.1145/3340531.3412034
- Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank citation ranking: Bringing order to the web. Stanford InfoLab.

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots",
  cooking_2 = "this soup recipe needs salt on a cold night",
```

```

space_1 = "the telescope revealed a distant galaxy and stars",
space_2 = "astronomers aimed the telescope at the stars all night"
), stop_words = c("the", "with", "and", "a", "this", "at", "on", "all"),
  weight = "tfidf")
hg_pagerank(hg, sort_by = "pagerank")
hg_pagerank(hg, personalized = c(cooking_1 = 1), sort_by = "pagerank")

```

hg_qanalysis

Q-analysis of a simplicial complex

Description

Q-connectivity (Atkin 1974): two maximal simplices are q-connected when they share a face of dimension at least q. Reports the number of q-connected components at every level and each node's highest level.

Usage

```
hg_qanalysis(sc)
```

Arguments

sc A simplicial_complex from `simplicial()`.

Value

A q_analysis object. Read it with `hg_get()`: the components per level (what = "q_levels", the default) or each node's highest level ("nodes").

References

Atkin, R. H. (1974). *Mathematical Structure in Human Affairs*. Heinemann.

See Also

`simplicial()`

Examples

```

adj <- matrix(c(0, 1, 1, 0,
               1, 0, 1, 1,
               1, 1, 0, 1,
               0, 1, 1, 0), 4, 4, dimnames = list(letters[1:4], letters[1:4]))
hg_get(hg_qanalysis(simplicial(adj)))

```

hg_representations	<i>Graph and hypergraph representations of the same data</i>
--------------------	--

Description

Reports, side by side, the four ways Coupette et al. (2024, Table 2) model one dataset: the hypergraph as it is (mh, repeated member sets counted separately), the binary hypergraph of distinct member sets (bh), and the graph the hypergraph reduces to, with (mg) or without (bg) edge multiplicities. The graph is the clique expansion for collaboration data (every pair of members of a tribunal is an edge) or the citation graph for data with hyperedge sources (the citing decision is joined to every decision in the block). The degree statistics show how much the choice of representation changes what a node looks like.

Usage

```
hg_representations(hg, graph = c("clique", "citation"), edge_source = NULL)
```

Arguments

hg	A static net_hg, typically a snapshot of a temporal_hypergraph() .
graph	"clique" (default) or "citation", the graph projection compared; see pairwise_network() . "citation" needs hyperedge sources.
edge_source	Hyperedge sources for graph = "citation", as in pairwise_network() ; omitted when the hypergraph carries them.

Value

A base data.frame with one row per representation, in the order bg, mg, bh, mh, and columns representation, type ("graph" or "hypergraph"), n_nodes, n_edges, mean_degree, median_degree. A directed citation graph adds median_out_degree and median_in_degree, and its mean_degree is the mean out-degree, which equals the mean in-degree. Hypergraph degree is the number of hyperedges containing a node; every node of the hypergraph counts, including nodes in no hyperedge.

References

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. doi:10.1098/rsta.2023.0141

Examples

```
tribunals <- data.frame(
  member = c("p1", "a1", "a2", "p2", "a1", "a3", "p1", "a1", "a2"),
  case = rep(c("A", "B", "C"), each = 3)
)
hg <- group_hypergraph(tribunals, "member", "case")
hg_representations(hg)
```

hg_seeds	<i>Seed labels from a clustering, for spreading or training</i>
----------	---

Description

Turns an unsupervised partition into the labeled examples that `hg_classify()`, `hg_neural()` and `hg_hypergat()` take: from each cluster, the `n` nodes with the highest stationary probability `pi` (the cluster's most representative members under the random walk). Ties are broken alphabetically by node name so the selection is deterministic.

Usage

```
hg_seeds(embedding, n = 5L)
```

Arguments

<code>embedding</code>	The data.frame from <code>hg_cluster(what = "embedding")</code> – columns <code>node</code> , <code>cluster</code> , <code>pi</code> .
<code>n</code>	Seeds per cluster (default 5L); clusters smaller than <code>n</code> contribute all their nodes.

Value

A named character vector – names are node identifiers, values their cluster labels – ready to pass as the `labels` argument of `hg_classify()`, `hg_neural()` or `hg_hypergat()`.

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots",
  cooking_2 = "this soup recipe needs salt on a cold night",
  space_1 = "the telescope revealed a distant galaxy and stars",
  space_2 = "astronomers aimed the telescope at the stars all night"
), stop_words = c("the", "with", "and", "a", "this", "at", "on", "all"))
pool <- hg_cluster(hg, k = 2, seed = 1, what = "embedding")
hg_seeds(pool, n = 1)
```

hg_sequences	<i>Per-actor state sequences from a clustered corpus</i>
--------------	--

Description

Turns document-level topic assignments into the canonical long sequence table the memory family consumes. Each document of a `text_hypergraph()` carries whatever metadata came with the corpus (an author, a turn, a timestamp); `hg_cluster()` gives each document a topic. `hg_sequences()` joins the two, orders the documents within each actor and returns one row per document as actor / time / action – the long event table every memory-network verb reads through its action / actor / time arguments (`hon()`, `mogen()`, `hupa()`, `markov_order()`, `hg_bootstrap()`, `hg_compare()`, `window_hypergraph()`):

Usage

```
hg_sequences(hg, clusters = NULL, actor, order_by, state = NULL, topics = NULL)
```

Arguments

hg	A text_hypergraph() – more generally, any <code>net_hg</code> carrying a documents table, the one reached with <code>hg_get(hg, what = "documents")</code> .
clusters	A partition of the documents, as returned by hg_cluster() : a <code>data.frame</code> with one row per node, a node column holding document identifiers and a state column (<code>cluster</code> by default, or the column named by <code>state</code>). <code>NULL</code> (the default) takes the state from the documents table instead, in which case <code>state</code> is required.
actor	Name of the documents-table column that groups the documents into sequences, such as an author, a participant, a course or a year. One sequence is returned per distinct value.
order_by	Name of the documents-table column ordering the documents within an actor – a turn number, an index, a date or a timestamp. Anything <code>order()</code> can sort is accepted, and the values are carried through to the time column unchanged.
state	Name of the column holding each document's state. With <code>clusters</code> supplied it names a column of <code>clusters</code> and defaults to <code>"cluster"</code> ; with <code>clusters = NULL</code> it names a column of the documents table and has no default.
topics	A mixed-membership topic model of <code>hg</code> fitted by hg_topics() . Each document's state is then its main topic, the topic with its largest share (<code>hg_get(topics, what = "documents")</code>). Give topics instead of <code>clusters</code> and <code>state</code> .

Details

```
seqs <- hg_sequences(hg, topics, actor = "student", order_by = "turn")
hon(seqs, action = "action", actor = "actor", time = "time")
```

Those three column names are fixed, which is what makes the second call the same every time. Passing the long table without `action =` raises `hypergraphs_long_format`: read as wide (one trajectory per row) it would give the wrong model.

This closes the one missing edge in the cross-family design: `hg_get(x, what = "pathways")` bridges the memory family to the others and [window_hypergraph\(\)](#) bridges sequences to hypergraphs, but until now the text family dead-ended at clustering and the caller had to assemble the sequence table themselves.

Value

A base `data.frame` with one row per document of `hg`, ordered by `actor` and then `time`, with columns

actor character. The value of the actor column; one sequence per distinct value.

time The value of the `order_by` column, type unchanged, so a numeric turn stays numeric and a Date stays a Date.

action character. The document's state – its cluster label under `clusters`, or the state column of the documents table.

Conditions

A `hypergraphs_bad_input` error is raised when `hg` is not a hypergraph, when it carries no documents table, when `actor`, `order_by` or `state` does not name a column of the table it is looked up in, when `clusters` lacks a node column or assigns a document more than one state, when some document of `hg` has no row in `clusters`, or when the `actor`, `time` or `state` values contain NA (which would silently shorten a sequence).

References

Xu, J., Wickramaratne, T. L., & Chawla, N. V. (2016). Representing higher-order dependencies in networks. *Science Advances*, 2(5), e1600028.

Zhou, D., Huang, J., & Scholkopf, B. (2006). Learning with hypergraphs: clustering, classification, and embedding. *Advances in Neural Information Processing Systems*, 19, 1601-1608. doi:10.7551/mitpress/7503.003.0205

See Also

`hg_cluster()` for the partition, `hon()` and `hg_bootstrap()` for what to do with the sequences, `window_hypergraph()` to read them back into a hypergraph.

Examples

```
posts <- data.frame(
  student = rep(c("ana", "bo"), each = 4),
  turn = rep(1:4, times = 2),
  phase = rep(c("early", "early", "late", "late"), times = 2),
  text = c(
    "simmer the soup with onions and carrots",
    "this soup recipe needs salt on a cold night",
    "the telescope revealed a distant galaxy and stars",
    "astronomers aimed the telescope at the stars all night",
    "a galaxy of stars seen through the telescope",
    "salt and onions make the soup",
    "the soup simmered all night with carrots",
    "astronomers watched the distant galaxy"
  ),
  stringsAsFactors = FALSE
)
hg <- text_hypergraph(posts, column = "text",
  stop_words = c("the", "with", "and", "a", "this",
    "at", "on", "all", "of", "through"))
topics <- hg_cluster(hg, k = 2, seed = 1)
hg_sequences(hg, topics, actor = "student", order_by = "turn")

# A state that is already a column of the corpus needs no clustering.
hg_sequences(hg, actor = "student", order_by = "turn", state = "phase")

# Into the memory family: every memory verb reads the long table.
seqs <- hg_sequences(hg, topics, actor = "student", order_by = "turn")
hon(seqs, action = "action", actor = "actor", time = "time", max_order = 2L)
```

hg_snapshot

*Extract one snapshot from a temporal hypergraph***Description**

A snapshot is a static `net_hg` holding the hyperedges a window measures. `mode = "active"` follows the format: an interval hyperedge is active on its closed interval, a contact hyperedge at its instant. `mode = "cumulative"` keeps every hyperedge begun by the end of the window, the growing view of a contact log; with `at` unset it is the static aggregate of everything observed.

Usage

```
hg_snapshot(
  x,
  at = NULL,
  window = 0,
  mode = c("active", "cumulative"),
  multiedges = TRUE
)
```

Arguments

<code>x</code>	A temporal_hypergraph() .
<code>at</code>	One time value, as a number on the hypergraph's clock or a date for a calendar hypergraph. <code>NULL</code> (default) is the end of observation.
<code>window</code>	How much time the snapshot covers, in the hypergraph's time unit: <code>0</code> (default) evaluates the instant <code>at</code> ; a positive width covers <code>[at, at + window)</code> ; <code>"all"</code> covers the whole observation period.
<code>mode</code>	<code>"active"</code> (default) or <code>"cumulative"</code> , see Description. The former <code>"all"</code> is deprecated: it is <code>"cumulative"</code> with <code>at</code> unset.
<code>multiedges</code>	Keep distinct edge identities with identical member sets? <code>TRUE</code> matches a multi-hypergraph; <code>FALSE</code> collapses them to one edge and records their counts in <code>edge_multiplicity</code> .

Value

A static `net_hg` usable by every hypergraphs `hypergraph` verb; its params record `at`, `window`, `temporal_mode` and the clock.

hg_snapshots

*Extract a sequence of temporal-hypergraph snapshots***Description**

Snapshots on a measurement grid with Dynet's meaning: `start` and `end` bound the measured period (default: the observation window), `step` is how often to look, `window` how much time each look covers, and `at` names instants directly. Without `step` and `at`, the grid is the event times of the hypergraph, each looked at as a point.

Usage

```
hg_snapshots(
  x,
  start = NULL,
  end = NULL,
  step = NULL,
  window = NULL,
  mode = c("active", "cumulative"),
  at = NULL,
  multiedges = TRUE
)
```

Arguments

<code>x</code>	A temporal_hypergraph() .
<code>start, end</code>	Bounds of the measured period, as numbers on the hypergraph's clock or dates for a calendar hypergraph. Default: the observation window.
<code>step</code>	How often to measure, in the hypergraph's time unit. <code>NULL</code> (default) measures at every event time.
<code>window</code>	How much time each measurement covers. Defaults to <code>step</code> when a step is given (a partition of the period) and to <code>0</code> otherwise (a point sample); larger than <code>step</code> gives a rolling window; "all" measures the whole period as one window.
<code>mode</code>	"active" (default) or "cumulative", see Description. The former "all" is deprecated: it is "cumulative" with <code>at</code> unset.
<code>at</code>	Instants to measure instead of a grid; may be a vector.
<code>multiedges</code>	Keep distinct edge identities with identical member sets? <code>TRUE</code> matches a multi-hypergraph; <code>FALSE</code> collapses them to one edge and records their counts in <code>edge_multiplicity</code> .

Value

A named list of `net_hg` objects with class `net_hg_snapshots`, named by the window starts on the hypergraph's clock.

Examples

```
seats <- data.frame(
  case = rep(c("A", "B", "C"), each = 3),
  arbitrator = c("p1", "a1", "a2", "p2", "a1", "a3", "p1", "a4", "a5"),
  constituted = rep(c(1, 2, 4), each = 3), concluded = rep(c(4, 3, 6), each = 3)
)
thg <- temporal_hypergraph(seats, node = "arbitrator", hyperedge = "case",
  start = "constituted", end = "concluded")
yearly <- hg_snapshots(thg, step = 2)
names(yearly)
```

hg_stability

Stability of a hypergraph clustering across resolutions

Description

Asks whether the partition `hg_cluster()` finds is a property of the data or of the particular sample, one resolution k at a time.

Usage

```
hg_stability(
  hg,
  k,
  type = c("zhou", "random_walk"),
  resample = c("subset", "seeds"),
  n_boot = 100L,
  fraction = 0.5,
  seed = 1L,
  what = c("k", "clusters"),
  seeds = c(1L, 99L),
  nstart = 25L
)
```

Arguments

hg	A <code>text_hypergraph()</code> (or any hypergraphs <code>net_hg</code>).
k	Vector of cluster counts to test, each at least 2.
type	"zhou" or "random_walk", the Laplacian of <code>hg_cluster()</code> .
resample	"subset" (default) for subsampling stability, or "seeds" for the two-seed solver check.
n_boot	Number of subsamples (default 100L).
fraction	Share of the nodes in each subsample, strictly between 0 and 1 (default 0.5, as in <code>fpc::clusterboot()</code>).
seed	Seed for the subsample draws and the k-means starts of every fit (default 1L).

what	For <code>resample = "subset"</code> : "k" (default) for one row per resolution, or "clusters" for one row per resolution and cluster.
seeds	For <code>resample = "seeds"</code> : two distinct k-means seeds (default <code>c(1L, 99L)</code>).
nstart	Number of k-means starts per fit (default 25L).

Details

`resample = "subset"` (**default**) is cluster-wise subsampling stability (Hennig 2007; the subsampling scheme of Ben-Hur et al. 2002). The nodes (on a document hypergraph, the documents) are clustered once in full. Then, `n_boot` times, a subsample of `floor(fraction * n)` nodes is drawn without replacement, the sub-hypergraph is formed by restricting every hyperedge to the drawn nodes (hyperedges left empty are dropped; the incidence weights are kept, so a tf-idf corpus keeps the full corpus's idf), and it is clustered again with the same `k`, `type`, `seed` and `nstart`. Each original cluster, restricted to the drawn nodes, is matched to its most similar subsample cluster by the Jaccard coefficient $|A \cap B| / |A \cup B|$ (0 when the cluster has no drawn member). A cluster's stability is its mean Jaccard over the subsamples. Hennig (2007) reads a mean of 0.5 or below as a dissolved cluster and 0.75 or above as a stable one. The subsamples are drawn once, up front, after `set.seed(seed)`, so they are the ones `fpc::clusterboot(bootmethod = "subset", subtuning = floor(fraction * n), seed = seed)` draws, and the same subsamples serve every `k`. The caller's random number stream is restored on exit.

A subsample can disconnect the hypergraph (a word that tied two groups of documents may not be drawn); `hg_cluster()` cannot cut a disconnected hypergraph, so that run is not scored, is counted in `n_failed`, and a warning of class `hypergraphs_hypergraph_disconnected` reports the count.

The eigengap column is the gap $\lambda_{k+1} - \lambda_k$ between consecutive eigenvalues of the full hypergraph's Laplacian (the gap column of `hg_cluster(what = "eigenvalues")`); the eigengap heuristic (von Luxburg 2007, Sec. 8.1) prefers the `k` where it is large relative to the gaps before it. Read it beside the stability: a `k` with a large gap and stable clusters is supported twice.

`resample = "seeds"` keeps the check this verb made before 0.6.0: it fits the full hypergraph twice per `k` with the two k-means seeds and reports whether the partitions agree. It measures whether the solver is deterministic on this embedding, not whether the structure survives a change of sample, and it never resamples the data.

Value

A base data.frame.

`resample = "subset"`, `what = "k"`: one row per resolution with `k`, `n_runs` (subsamples scored), `n_failed` (subsamples that disconnected the hypergraph), `mean_jaccard` and `min_jaccard` (the mean and the smallest of the clusters' stabilities), `n_stable` (clusters with stability at least 0.75), `n_dissolved` (clusters with stability 0.5 or below) and `eigengap`.

`resample = "subset"`, `what = "clusters"`: one row per resolution and cluster of the full partition, with `k`, `cluster`, `size` (its nodes), `jaccard` (its stability, `fpc::clusterboot()`'s `subsetmean`), `n_dissolved` (subsamples with Jaccard 0.5 or below, `subsetbrd`), `n_recovered` (subsamples with Jaccard above 0.75, `subsetrecover`) and `n_runs`.

`resample = "seeds"`: one row per resolution with `k`, `identical_partition` (are the two partitions literally identical, labels included) and `ari` (their adjusted Rand index).

Raises `hypergraphs_bad_input` for a fraction outside (0, 1), a subsample too small to hold `k` clusters, a non-positive `n_boot`, or `what = "clusters"` with `resample = "seeds"`.

References

- Hennig, C. (2007). Cluster-wise assessment of cluster stability. *Computational Statistics & Data Analysis*, 52(1), 258–271. doi:10.1016/j.csda.2006.11.025
- Ben-Hur, A., Elisseeff, A., & Guyon, I. (2002). A stability based method for discovering structure in clustered data. *Pacific Symposium on Biocomputing*, 7, 6–17. doi:10.1142/9789812799623_0002
- von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, 17(4), 395–416. doi:10.1007/s112220079033z
- Hubert, L., & Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2, 193–218.

Examples

```
hg <- text_hypergraph(head(covid_abstracts, 40), column = "abstract",
                      id = "doc", stop_words = stop_words_en(),
                      min_count = 3)
hg_stability(hg, k = 2:4, n_boot = 20)
hg_stability(hg, k = 3, n_boot = 20, what = "clusters")
hg_stability(hg, k = 2, resample = "seeds")
```

hg_subset

Sub-hypergraph by hyperedges, nodes or hyperedge attributes

Description

Keeps part of a hypergraph and returns it as a `net_hg` with the same incidence weights, edge metadata and multiplicities. Three selectors combine by intersection. `edges` names the hyperedges to keep. `nodes` keeps the hyperedges whose members all lie in the set, the induced sub-hypergraph of `HypergraphX`. `where` keeps the hyperedges whose attributes take given values, so `where = c(citing = "153-001")` on a citation-block hypergraph is the hypergraph of one citing decision (Coupette et al. 2024, Figure 3).

Usage

```
hg_subset(
  hg,
  edges = NULL,
  nodes = NULL,
  where = NULL,
  size = NULL,
  component = c("all", "largest"),
  drop_isolated = TRUE
)
```

Arguments

hg	A net_hg (from <code>group_hypergraph()</code> , <code>hg_snapshot()</code> , <code>text_hypergraph()</code> , ...).
edges	Hyperedge names to keep: a character vector, or a data.frame with an edge column such as the table <code>hg_edge centrality()</code> or <code>hg_edges()</code> returns, so a ranking can be passed straight through.
nodes	Character vector of node names. Only hyperedges contained in this set are kept, and the node set of the result is exactly nodes.
where	Hyperedge attribute values to keep: a named vector or list, one element per attribute column of the edge metadata (the columns <code>temporal_hypergraph()</code> keeps because they are constant within a hyperedge), each holding the value or values to keep.
size	Hyperedge sizes to keep, as a vector of member counts (distinct members): <code>size = 3</code> keeps the hyperedges with exactly three members, the 3-uniform hypergraph that <code>hg_motifs()</code> needs.
component	"all" (default) keeps every connected component; "largest" keeps only the largest connected component, in which two nodes are connected when a chain of shared hyperedges joins them. Spectral clustering and label spreading need a connected hypergraph (Hayashi et al. 2020), and a corpus of short texts often holds a few texts that share no word with the rest. Applied after the other filters.
drop_isolated	Drop nodes that belong to no retained hyperedge? Default TRUE. Ignored when nodes is given.

Value

A net_hg whose incidence matrix is the selected sub-matrix of the input, sparse if the input is sparse. Edge metadata (`edge_data`), duplicate multiplicities (`edge_multiplicity`) and the window counts of a `window_hypergraph()` are subset alongside. The nodes the subset removes are recorded in `params$subset$removed`, and the labels of removed nodes are set aside with a `hypergraphs_dropped_documents` warning by `hg_classify()`, `hg_keywords()` and the other verbs that take labels, so the table that built the hypergraph can be passed back whole.

References

- Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. doi:10.1098/rsta.2023.0141
- Hayashi, K., Aksoy, S. G., Park, C. H., & Park, H. (2020). Hypergraph random walks, Laplacians, and clustering. *Proceedings of CIKM 2020*, 495-504. doi:10.1145/3340531.3412034

Examples

```
dat <- data.frame(
  member = c("a", "b", "c", "b", "c", "d", "d", "e"),
  event = c("e1", "e1", "e1", "e2", "e2", "e2", "e3", "e3"),
  kind = c("x", "x", "x", "x", "x", "x", "y", "y")
)
hg <- group_hypergraph(dat, node = "member", hyperedge = "event")
```

```

hg_subset(hg, edges = c("e1", "e2"))
hg_subset(hg, nodes = c("b", "c", "d", "e"))
hg_subset(hg, where = c(kind = "y"))
hg_subset(hg, edges = c("e1", "e3"), component = "largest")

```

hg_topic_quality	<i>Topic coherence and exclusivity</i>
------------------	--

Description

Scores the top words of each topic on two axes, with the published definitions that topic-model software reports, so hypergraph topics and topics from another model (LDA, STM) can be compared on the same corpus. The corpus is the document hypergraph: its documents are the reference collection and a word's documents are the documents it occurs in.

Usage

```

hg_topic_quality(
  hg,
  clusters = NULL,
  words = NULL,
  n = 10L,
  min_docs = 1L,
  coherence = c("umass", "npmi", "npmi_cluster"),
  exclusivity = c("frex", "share", "none"),
  sort_by = c("probability", "share"),
  frexw = 0.7,
  topics = NULL
)

## S3 method for class 'hypergraphs_topic_quality'
plot(x, ...)

```

Arguments

hg	The document hypergraph the topics describe: a bag-of-words <code>text_hypergraph()</code> with documents as nodes (the default construction).
clusters	The tidy table returned by <code>hg_cluster()</code> (columns node, cluster), or a named vector of cluster labels. NULL when only words are scored.
words	NULL (default); a data.frame of topic words with columns word and topic (or cluster, so <code>hg_keywords()</code> output is passed as it is) and optionally rank (the order within a topic; row order otherwise); or a character matrix with one column per topic and rows in rank order, as <code>topicmodels::terms()</code> returns it (column names are the topic labels). Every word must be in the vocabulary of hg. With clusters too, every topic must be a cluster label.
n	Top words per topic to score (default 10).

min_docs	Support floor: with clusters and no words, keep only words found in at least this many of the cluster's documents (default 1).
coherence	"umass" (default), "npmi" or "npmi_cluster"; see Details.
exclusivity	"frex" (default), "share" or "none"; see Details.
sort_by	How a cluster's top words are chosen when words is not given: "probability" (default) or "share"; see Details.
frexw	Weight on exclusivity in FREX, in (0, 1) (default 0.7, as in stm).
topics	A mixed-membership topic model of hg fitted by <code>hg_topics()</code> . Its topics are scored on their most probable words, and FREX is computed from its word distributions $P(w z)$, as stm computes it from an STM's. size is then the topic's expected number of documents (its summed shares). Give topics instead of clusters and words.
x	A table returned by the verb.
...	Unused; for S3 consistency.

Details

Topics. A topic is either a cluster of documents (clusters, from `hg_cluster()` or any labelling) or a list of words (words, e.g. the top terms of a fitted LDA or STM model), or both. For a cluster, the word distribution is the plug-in estimate $\beta_{kw} = c_{kw} / \sum_v c_{kv}$, where c_{kw} is the token count of word w in the cluster's documents, over the words that occur in some clustered document; its top words are the n most probable (sort_by = "probability", ties in vocabulary order, as stm ranks β) or the n with the largest share of their corpus count in the cluster (sort_by = "share", the ranking of `hg_keywords()` with sort_by = "share"), keeping words found in at least min_docs of the cluster's documents. Given words, those words are scored in their given order.

Coherence (coherence =), with $D(v)$ the number of documents containing v , $D(v, u)$ the number containing both, N the number of documents and $v_1, \dots, v_M$ the top words in rank order:

- "umass" (default): the UMass coherence of Mimno et al. (2011), $\sum_{m=2}^M \sum_{l=1}^{m-1} \log\{(D(v_m, v_l) + \epsilon) / (D(v_l) + \epsilon)\}$, a sum over the $M(M-1)/2$ pairs that conditions on the higher-ranked word, counted on the whole corpus, with $\epsilon = 0.01$ in numerator and denominator as in `stm::semanticCoherence()` (Roberts et al. 2019); Mimno et al. add 1 to the numerator only. At most 0; closer to 0 is more coherent.
- "npmi": the mean normalised pointwise mutual information over the word pairs (Bouma 2009; the topic-coherence measure of Lau et al. 2014), with document co-occurrence probabilities $p(v) = D(v)/N$ and $p(v, u) = D(v, u)/N$ on the whole corpus: $\log\{p(v, u) / (p(v)p(u))\} / -\log p(v, u)$; -1 for a pair that never co-occurs (the limit), 1 for a pair present in every document.
- "npmi_cluster": the same NPMI counted only on the cluster's own documents (the definition this verb used before 0.6.0; needs clusters). It rewards words that co-occur inside the topic whatever they do elsewhere, so it is not comparable with published values.

Exclusivity (exclusivity =), which needs the cluster word distributions and so clusters:

- "frex" (default): the sum over the top words of FREX (Bischof & Airola 2012), the weighted harmonic mean of a word's rank in the topic's distribution and its rank in exclusivity $\beta_{kw} / \sum_j \beta_{jw}$, both as empirical CDFs over the vocabulary: $1/(w/ex + (1-w)/fr)$ with weight frexw on exclusivity, exactly as `stm::exclusivity()` computes it. Between 0 and n ; larger is more exclusive.

- "share": the mean over the top words of the share of the word's corpus count that falls in the cluster, $c_{kw} / \sum_j c_{jw}$ (the definition used before 0.6.0); 1 when the cluster owns its words.
- "none": no exclusivity (NA); the choice when only words are given.

The measures before 0.6.0 are coherence = "npmi_cluster", exclusivity = "share", sort_by = "share".

Value

A base data.frame of class hypergraphs_topic_quality, one row per topic (cluster labels in natural order, or topics of words in natural order): topic, size (the cluster's documents; NA without clusters), n_words (top words scored), coherence, exclusivity (NA for fewer than two words, or exclusivity = "none"), coherence_type and exclusivity_type. plot() draws exclusivity against coherence with one labelled point per topic. Raises hypergraphs_bad_input for unknown node names, a malformed words table, words outside the vocabulary, topics of words that are not cluster labels, a measure that needs clusters without them, or a hypergraph whose nodes are not documents.

References

- Mimno, D., Wallach, H. M., Talley, E., Leenders, M., & McCallum, A. (2011). Optimizing semantic coherence in topic models. *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 262–272. <https://aclanthology.org/D11-1024/>
- Bouma, G. (2009). Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL*, 31–40.
- Lau, J. H., Newman, D., & Baldwin, T. (2014). Machine reading tea leaves: Automatically evaluating topic coherence and topic model quality. *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 530–539. doi:10.3115/v1/E14-1056
- Bischof, J. M., & Airolidi, E. M. (2012). Summarizing topical content with word frequency and exclusivity. *Proceedings of the 29th International Conference on Machine Learning (ICML)*. arXiv:1206.4631.
- Roberts, M. E., Stewart, B. M., & Tingley, D. (2019). stm: An R package for structural topic models. *Journal of Statistical Software*, 91(2), 1–40. doi:10.18637/jss.v091.i02

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots",
  cooking_2 = "this soup recipe needs salt on a cold night",
  space_1 = "the telescope revealed a distant galaxy and stars",
  space_2 = "astronomers aimed the telescope at the stars all night"
), stop_words = c("the", "with", "and", "a", "this", "at", "on", "all"))
topics <- hg_cluster(hg, k = 2, seed = 1)
hg_topic_quality(hg, topics, n = 3)
hg_topic_quality(hg, topics, n = 3, coherence = "npmi")
# topic words from another model, scored on the same corpus
lda_words <- data.frame(topic = c("t1", "t1", "t2", "t2"),
  word = c("soup", "salt", "telescope", "stars"))
hg_topic_quality(hg, words = lda_words, exclusivity = "none")
```

hg_topic_search	<i>Choose the number of topics by coherence and exclusivity</i>
-----------------	---

Description

Fits `hg_topics()` for every number of topics in `k` and scores each model by the mean semantic coherence (UMass; Mimno et al. 2011) and the mean exclusivity (FREX; Bischof & Airoldi 2012) of its topics, computed by `hg_topic_quality()` on the `n` most probable words. This is the diagnostic of `stm::searchK()` (Roberts, Stewart & Tingley 2019). Adding topics usually makes them more exclusive and less coherent, so no single number is best; Roberts et al. (2014) choose among the models on the frontier of the two, those that no other model beats on both. A model is on the frontier when no other number of topics has both a higher mean coherence and a higher mean exclusivity.

Usage

```
hg_topic_search(
  hg,
  k,
  nstart = 1L,
  n = 10L,
  frexw = 0.7,
  max_iter = 1000L,
  tol = 1e-05,
  seed = 1L,
  parallel = FALSE,
  n_cores = 2L
)

## S3 method for class 'hypergraphs_topic_search'
plot(x, ...)
```

Arguments

<code>hg</code>	A <code>net_hg</code> or <code>text_hypergraph</code> .
<code>k</code>	Numbers of topics to compare, at least two whole numbers.
<code>nstart</code>	Number of random starts (default 10).
<code>n</code>	Top words per topic scored (default 10).
<code>frexw</code>	Weight on exclusivity in FREX, in (0, 1) (default 0.7).
<code>max_iter</code>	Maximum multiplicative updates per start (default 1000).
<code>tol</code>	Relative change of the divergence between two checks, ten updates apart, below which a start has converged (default 1e-5); 0 runs every start for <code>max_iter</code> updates.
<code>seed</code>	Seed of the first start; start <code>r</code> uses <code>seed + r - 1</code> . The caller's random-number stream is restored on exit. Default 1.

parallel	Fit the numbers of topics with <code>parallel::mclapply()</code> (not on Windows); every fit seeds itself, so the result equals the serial one. Default FALSE.
n_cores	Cores when <code>parallel = TRUE</code> (default 2).
x	A table returned by the verb.
...	Unused.

Value

A base `data.frame` of class `hypergraphs_topic_search`, one row per number of topics: `k`, coherence and exclusivity (means over the topics), divergence (of the kept start), agreement (mean agreement of the topics across the starts; NA with `nstart = 1`), converged and frontier. `plot()` draws exclusivity against coherence, one labelled point per number of topics, the frontier joined by a line and drawn as filled points. Conditions are those of `hg_topics()`; a start that does not converge raises `hypergraphs_no_converge`.

References

- Roberts, M. E., Stewart, B. M., & Tingley, D. (2019). `stm`: An R package for structural topic models. *Journal of Statistical Software*, 91(2), 1-40. doi:[10.18637/jss.v091.i02](https://doi.org/10.18637/jss.v091.i02)
- Roberts, M. E., Stewart, B. M., Tingley, D., Lucas, C., Leder-Luis, J., Gadarian, S. K., Albertson, B., & Rand, D. G. (2014). Structural topic models for open-ended survey responses. *American Journal of Political Science*, 58(4), 1064-1082. doi:[10.1111/ajps.12103](https://doi.org/10.1111/ajps.12103)
- Mimno, D., Wallach, H. M., Talley, E., Leenders, M., & McCallum, A. (2011). Optimizing semantic coherence in topic models. *Proceedings of EMNLP 2011*, 262-272.
- Bischof, J. M., & Airoldi, E. M. (2012). Summarizing topical content with word frequency and exclusivity. *Proceedings of ICML 2012*, 201-208.

See Also

[hg_topics\(\)](#), [hg_topic_quality\(\)](#).

Examples

```
corpus <- c(
  a = "soup salt onion soup broth", b = "salt soup broth onion",
  c = "stars sky moon night", d = "sky stars night moon moon",
  e = "soup stars salt sky night broth", f = "onion salt stars broth")
search <- hg_topic_search(text_hypergraph(corpus), k = 2:3, nstart = 2,
                          n = 4)
search
```


hg_topics

*Mixed-membership topic model of a hypergraph***Description**

Factorizes the document-word counts of a hypergraph into topics, so that every document is a mixture of topics and every topic a distribution over words. The counts X are approximated by WH , with non-negative W (documents by topics) and H (topics by words), by the multiplicative updates that minimise the generalised Kullback-Leibler divergence (Lee & Seung 1999, 2001). With this divergence the fit is the maximum-likelihood solution of probabilistic latent semantic analysis (Hofmann 1999; Gaussier & Goutte 2005): the normalised rows of W are the documents' topic shares $P(z | d)$ and the normalised rows of H the topics' word probabilities $P(w | z)$. LDA is the same model with Dirichlet priors on these distributions.

Usage

```
hg_topics(
  hg,
  k,
  nstart = 10L,
  max_iter = 1000L,
  tol = 1e-05,
  depth = 10L,
  seed = 1L,
  parallel = FALSE,
  n_cores = 2L
)

## S3 method for class 'net_hg_topics'
hg_get(
  x,
  what = c("topics", "shares", "words", "documents", "restarts", "prevalence"),
  ...,
  n = 10L,
  topic = NULL,
  group = NULL,
  top = NULL
)

## S3 method for class 'net_hg_topics'
print(x, n = 10L, ...)

## S3 method for class 'net_hg_topics'
plot(x, y, n = 8L, type = c("words", "prevalence"), group = NULL, ...)
```

Arguments

hg A net_hg or text_hypergraph.

k	Number of topics, a whole number of at least 2 and smaller than the numbers of documents and words.
nstart	Number of random starts (default 10).
max_iter	Maximum multiplicative updates per start (default 1000).
tol	Relative change of the divergence between two checks, ten updates apart, below which a start has converged (default 1e-5); 0 runs every start for max_iter updates.
depth	Number of top words compared between starts (default 10).
seed	Seed of the first start; start r uses seed + r - 1. The caller's random-number stream is restored on exit. Default 1.
parallel	Run the starts with <code>parallel::mclapply()</code> (not on Windows). Every start seeds itself, so the result equals the serial one. Default FALSE.
n_cores	Cores when parallel = TRUE (default 2).
x	A <code>net_hg_topics</code> object.
what	Which table: "topics" (default), "shares", "words", "documents", "restarts" or "prevalence" (the topics' mean share within each group of documents given by group).
...	Unused.
n	For what = "words": the number of words per topic (default 10). For <code>print()</code> : rows of the topic table shown.
topic	For what = "shares" and "words": keep only these topics (labels such as "Topic 3").
group	For what = "prevalence": the group of each document, as the name of a column of the modelled text hypergraph's document table (group = "period"), a named character vector (names are document ids) or a data.frame with a node column and a label, cluster, community or predicted column. Documents without a group are left out.
top	Keep only the first top rows of the returned table.
y	Unused.
type	For <code>plot()</code> : "words" (default), the n most probable words of every topic, or "prevalence", the mean share of every topic in each group of documents given by group.

Details

The fit depends on its random start, so it is run from nstart starts and the start with the lowest divergence is kept. Every other start is compared with it by the agreement of the topics' top words (Greene, O'Callaghan & Cunningham 2014): the topics of the two fits are matched one to one by the Hungarian method (Kuhn 1955) on the average Jaccard coefficient of their ranked top depth words, and the matched coefficients are averaged. A topic that the other starts find again has a high agreement.

A text hypergraph from `text_hypergraph()` is factorized on its word counts, whatever its node orientation and incidence weighting; any other hypergraph on its incidence, with its nodes as documents and its hyperedges as words.

Value

A `net_hg_topics` object. Read it with `hg_get()`: one row per topic (what = "topics", the default: topic, prevalence (its mean share over the documents), documents (the summed shares, the expected number of documents), agreement (its mean matched average Jaccard over the other starts) and top_words); one row per document and topic ("shares": node, topic, share, summing to one over a document's topics); one row per topic and word ("words": topic, rank, word, probability, n words per topic); one row per document with its largest share ("documents": node, topic, share); or one row per start ("restarts": run, divergence, iterations, converged, best, agreement); or, with group, one row per topic and group ("prevalence": topic, group, prevalence (the mean share over the group's documents), documents (the summed shares), n (the group's documents) and top_words), the topic prevalence by covariate of Roberts et al. (2014). `print()` shows the topics, `summary()` every table, and `plot()` the top words of every topic. A start that reaches `max_iter` without converging raises the warning `hypergraphs_no_converge`.

`plot()` returns a ggplot of the n most probable words of every topic, one panel per topic, or with `type = "prevalence"` of the prevalence of every topic in each group, one bar per topic and group, the topics labelled by their top words.

Conditions

`hypergraphs_bad_input` for an input that is not a hypergraph, a k outside its range, or invalid control arguments.

References

- Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401, 788-791. doi:10.1038/44565
- Lee, D. D., & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. *Advances in Neural Information Processing Systems*, 13, 556-562.
- Hofmann, T. (1999). Probabilistic latent semantic indexing. *Proceedings of SIGIR 1999*, 50-57. doi:10.1145/312624.312649
- Gaussier, E., & Goutte, C. (2005). Relation between PLSA and NMF and implications. *Proceedings of SIGIR 2005*, 601-602. doi:10.1145/1076034.1076148
- Greene, D., O'Callaghan, D., & Cunningham, P. (2014). How many topics? Stability analysis for topic models. *Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2014)*, 498-513. doi:10.1007/9783662448489_32
- Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2, 83-97. doi:10.1002/nav.3800020109
- Roberts, M. E., Stewart, B. M., Tingley, D., Lucas, C., Leder-Luis, J., Gadarian, S. K., Albertson, B., & Rand, D. G. (2014). Structural topic models for open-ended survey responses. *American Journal of Political Science*, 58(4), 1064-1082. doi:10.1111/ajps.12103

See Also

`hg_topic_quality()` scores the topics' words, `hg_cluster()` for a partition of the documents.

Examples

```
corpus <- c(
  a = "soup salt onion soup broth", b = "salt soup broth onion",
  c = "stars sky moon night", d = "sky stars night moon moon",
  e = "soup stars salt sky night broth")
fit <- hg_topics(text_hypergraph(corpus), k = 2, nstart = 3)
fit
hg_get(fit, what = "shares")
hg_get(fit, what = "words", n = 4)
```

hg_transitivity	<i>Local clustering coefficients of a hypergraph</i>
-----------------	--

Description

Computes one or more published local clustering coefficients for every node of a [net_hg](#). Each variant is named after its definition:

Usage

```
hg_transitivity(
  hg,
  type = "projection",
  what = c("nodes", "summary"),
  sort_by = NULL,
  n = Inf
)
```

Arguments

hg	A hypergraphs <code>net_hg</code> (network_hypergraph() , group_hypergraph() , text_hypergraph() , ...), dense or sparse.
type	One or more of "projection" (default), "extra_overlap", "two_node_union", "two_node_min", "two_node_max".
what	"nodes" (default; one row per node) or "summary" (one row per type with the mean local coefficient over the nodes where it is defined, the average local clustering used by Chodrow 2020, and the number of nodes where it is undefined).
sort_by	For what = "nodes", optional type to sort by, descending (ties broken by node name, NA last).
n	Return only the first n node rows after sorting: a whole number ≥ 1 , or Inf (default) for all.

Details

- "projection" – the Watts & Strogatz (1998) local clustering of the unweighted clique expansion: the share of pairs of a node's neighbours that are themselves neighbours. A path through hyperedges is closed by *any* hyperedge, so a single hyperedge of size three already closes a triangle (the path-based generalization reviewed by Battiston et al. 2020, Sec. III.C).
- "extra_overlap" – Zhou & Nakhleh (2011), in the statement of Klimm, Deane & Reinert (2021): for every unordered pair of hyperedges e, f containing the node, the *extra overlap*

$$EO(e, f) = \frac{|N(D_{e,f}) \cap D_{f,e}| + |N(D_{f,e}) \cap D_{e,f}|}{|D_{e,f}| + |D_{f,e}|}, \quad D_{e,f} = e \setminus f,$$

is the share of the non-shared members that are linked to the other side by a *third* hyperedge ($N(S)$ is the union of the hypergraph neighbourhoods of S); $EO = 0$ when $e = f$. The node's coefficient is the mean EO over its $k(k-1)/2$ pairs of hyperedges. It reduces to the ordinary clustering coefficient on a graph and does not count the triangles *inside* a hyperedge.

- "two_node_union", "two_node_min", "two_node_max" – the two-mode clustering of Latapy, Magnien & Del Vecchio (2008), applied to hypernetworks by Gallagher & Goldberg (2013): for nodes u, v with hyperedge sets $M(u), M(v)$, $cc(u, v) = |M(u) \cap M(v)| / |M(u) \cup M(v)|$ (union), or with $\min / \max(|M(u)|, |M(v)|)$ in the denominator; the node's coefficient averages $cc(u, v)$ over its hypergraph neighbours v (Latapy et al.'s $N(N(u))$).

Undefined values are NA. A node with fewer than two projection neighbours has no "projection" coefficient, a node in no hyperedge has no "extra_overlap" coefficient (hyperdegree 1 gives 0, as Klimm et al. define it), and a node without neighbours has no "two_node_*" coefficient. XGI 0.10.2 reports 0 in all of these cases; the values agree everywhere else (the oracle test maps NA to 0 before comparing).

The global clustering coefficient $C_2(H)$ of Estrada & Rodriguez-Velazquez (2006) is not provided (no independent oracle).

Duplicate hyperedges are distinct hyperedges (as in XGI): they raise the hyperdegree and, for "extra_overlap", contribute pairs with $EO = 0$. "extra_overlap" forms a dense edge-by-edge matrix and visits every pair of intersecting hyperedges, so it is meant for hypergraphs with at most a few thousand hyperedges; the other variants use sparse products when the incidence is sparse.

Value

A base data.frame. For what = "nodes": one row per node with column node and one numeric column per requested type (in $[\emptyset, 1]$, NA where undefined). For what = "summary": one row per type with columns type, mean_local and n_undefined.

Conditions

Raises hypergraphs_bad_input when hg is not a net_hg or n is not a count.

References

- Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393(6684), 440-442. doi:10.1038/30918
- Zhou, W., & Nakhleh, L. (2011). Properties of metabolic graphs: biological organization or representation artifacts? *BMC Bioinformatics*, 12, 132. doi:10.1186/1471210512132

Klimm, F., Deane, C. M., & Reinert, G. (2021). Hypergraphs for predicting essential genes using multiprotein complex data. *Journal of Complex Networks*, 9(2), cnaa028. doi:10.1093/comnet/cnaa028

Latapy, M., Magnien, C., & Del Vecchio, N. (2008). Basic notions for the analysis of large two-mode networks. *Social Networks*, 30(1), 31-48. doi:10.1016/j.socnet.2007.04.006

Gallagher, S. R., & Goldberg, D. S. (2013). Clustering coefficients in protein interaction hypernetworks. *Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedical Informatics (BCB '13)*, 552-560. doi:10.1145/2506583.2506635

Battiston, F., Cencetti, G., Iacopini, I., Latora, V., Lucas, M., Patania, A., Young, J.-G., & Petri, G. (2020). Networks beyond pairwise interactions: Structure and dynamics. *Physics Reports*, 874, 1-92. doi:10.1016/j.physrep.2020.05.004

Chodrow, P. S. (2020). Configuration models of random hypergraphs. *Journal of Complex Networks*, 8(3), cnaa018. doi:10.1093/comnet/cnaa018

See Also

[hg_assortativity\(\)](#), [hg_measures\(\)](#), [pairwise_network\(\)](#).

Examples

```
groups <- data.frame(
  member = c("a", "b", "c", "a", "d", "b", "d", "c", "d", "e"),
  group = c("g1", "g1", "g1", "g2", "g2", "g3", "g3", "g4", "g4", "g4")
)
hg <- group_hypergraph(groups, node = "member", hyperedge = "group")
hg_transitivity(hg, type = c("projection", "extra_overlap"))
hg_transitivity(hg, type = "two_node_union", what = "summary")
```

hg_wasserstein

Wasserstein Distance Between Persistence Diagrams

Description

Computes the finite-order Wasserstein distance between persistence diagrams. For order q , it minimizes the sum of powered matching costs over bijections between diagram points and copies of the diagonal, then takes the q -th root. Point-to-point costs use an L_p ground metric; `internal_p = Inf` gives the usual L_∞ convention used by GUDHI.

Usage

```
hg_wasserstein(d1, d2, dimension = NULL, order = 1, internal_p = Inf)
```

Arguments

d1, d2	persistent_homology objects from <code>hg_homology()</code> , or data.frames with columns dimension, birth, death (finite births; deaths finite or Inf). An invalid diagram raises <code>hypergraphs_bad_input</code> .
dimension	Integer vector of dimensions to compare. NULL (default) compares all dimensions appearing in either diagram.
order	Finite Wasserstein order, a number greater than or equal to one. The common choices are 1 and 2.
internal_p	Ground-metric order, a number greater than or equal to one or Inf (the default).

Details

Finite points may match the diagonal. Essential classes are matched only to essential classes, and a dimension whose essential counts differ has distance Inf. An `hg_homology()` result marks its essential classes itself (death = Inf in Vietoris–Rips mode, death = 0 in clique mode and for a window complex); in a data.frame an essential class is written with death = Inf, and death = 0 is an ordinary finite death. The finite assignment is solved exactly with a native Hungarian algorithm, so no optional optimization package is required. Distances are computed on rescaled coordinates, so large orders and wide coordinate ranges do not overflow.

Value

Named numeric vector, one value per requested dimension. Names are "dim_<k>".

References

Kerber, M., Morozov, D., & Nigmetov, A. (2017). Geometry helps to compare persistence diagrams. *Journal of Experimental Algorithmics*, 22, 1–20. doi:10.1145/3064175

Examples

```
d1 <- data.frame(dimension = 0L, birth = 0, death = 2)
d2 <- data.frame(dimension = 0L, birth = 0, death = 3)
hg_wasserstein(d1, d2)
```

hon

Higher-order network (HON) from sequences

Description

Builds a higher-order network: a state is split into memory nodes ("a -> b": state b reached from a) wherever the next-state distribution depends on where the sequence came from, as decided by the BuildHON / BuildHON+ rule extraction.

Usage

```

hon(
  data,
  max_order = 5L,
  min_freq = 1L,
  collapse_repeats = FALSE,
  method = "hon+",
  action = NULL,
  actor = NULL,
  time = NULL,
  session = NULL,
  time_threshold = 900,
  timezone = "UTC",
  group = NULL
)

```

Arguments

data	Sequences in any form described in sequence-input : a long event table (with action), a wide data.frame, a list of vectors, or a model object carrying its sequences.
max_order	Integer. Highest order considered. Default 5.
min_freq	Integer. Transitions observed fewer times are treated as zero. Default 1.
collapse_repeats	Logical. Collapse adjacent repeated states before extraction. Default FALSE.
method	"hon+" (default, parameter-free BuildHON+) or "hon" (the original eager BuildHON).
action, actor, time, session, time_threshold, timezone	Long-format arguments (see sequence-input); leave the column names NULL for wide, list or model input.
group	NULL (default) for one network, or the grouping of the sequences for a group model: the name of a column of data (long or wide format), or a vector with one label per sequence (per row of a wide data frame, per element of a list). A group model holds one network per group and the data of each group; it is the input of hg_compare() and can be passed to hg_bootstrap() and hg_get() .

Value

A `net_hon` object (also a `cograph_network`, so it plots directly). Read its tables with [hg_get\(\)](#): the rules (what = "rules", the default), the memory nodes ("nodes") and the higher-order pathways ("pathways").

References

Xu, J., Wickramaratne, T. L., & Chawla, N. V. (2016). Representing higher-order dependencies in networks. *Science Advances*, 2(5), e1600028. doi:10.1126/sciadv.1600028

Saebi, M., Xu, J., Kaplan, L. M., Ribeiro, B., & Chawla, N. V. (2020). Efficient modeling of higher-order dependencies in networks: from algorithm to application for anomaly detection. *EPJ Data Science*, 9, 15. doi:[10.1140/epjds/s1368802000233y](https://doi.org/10.1140/epjds/s1368802000233y)

See Also

[honem\(\)](#), [hg_centrality\(\)](#), [hg_communities\(\)](#), [hg_bootstrap\(\)](#), [hg_compare\(\)](#)

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"),
            c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
net <- hon(seqs, max_order = 2)
hg_get(net)

# the same network from a long event table
events <- data.frame(
  who = rep(c("s1", "s2", "s3", "s4"), each = 6),
  step = rep(1:6, 4),
  what = unlist(seqs)
)
hg_get(hon(events, action = "what", actor = "who", time = "step",
            max_order = 2), what = "nodes")
```

honem

Embedding of a higher-order network (HONEM)

Description

Low-dimensional node embedding of a higher-order network that preserves its higher-order dependencies: exponentially decaying powers of the network's transition matrix, followed by a truncated singular value decomposition.

Usage

```
honem(hon, dim = 32L, max_power = 10L)
```

Arguments

hon	A net_hon from hon() , or a square weighted adjacency matrix.
dim	Integer. Embedding dimension, capped at the number of nodes minus one. Default 32.
max_power	Integer. Longest walk length in the neighbourhood matrix. Default 10.

Value

A net_honem object. Read it with [hg_get\(\)](#): the node coordinates (what = "embeddings", the default) or the variance per dimension ("variance").

References

Saebi, M., Ciampaglia, G. L., Kaplan, L. M., & Chawla, N. V. (2020). HONEM: Learning embedding for higher order networks. *Big Data*, 8(4), 255–269. doi:10.1089/big.2019.0169

See Also

[hon\(\)](#)

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"),
            c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
emb <- honem(hon(seqs, max_order = 2), dim = 2)
hg_get(emb)
```

hypo

Hypergeometric anomalies of paths or of co-occurring node pairs

Description

One verb, two estimators, chosen by the class of `x`:

a hypergraph (`net_hg`) node pairs scored against a hypergeometric null fitted to every node's co-occurrence strength; returns a table. See [hypo.net_hg\(\)](#).

sequences (a long, wide or list input, or a model object carrying sequences; see [sequence-input](#)) paths of a De Bruijn graph scored against the same null (HYPA); returns a `net_hypo` object. See [hypo.default\(\)](#).

Each method keeps its own arguments; passing an argument that only the other method takes raises `hypergraphs_bad_input`.

Usage

```
hypo(x, ...)
```

Arguments

`x` A `net_hg`, or sequences.
`...` Arguments of the method for `class(x)`.

Value

See the methods.

References

LaRock, T., Nanumyan, V., Scholtes, I., Casiraghi, G., Eliassi-Rad, T., and Schweitzer, F. (2020). HYPA: Efficient detection of path anomalies in time series data on networks. *Proceedings of the 2020 SIAM International Conference on Data Mining*, 460–468. doi:10.1137/1.9781611976236.52

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"),
             c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
hg_get(hypo(seqs, order = 2, min_count = 1))
```

hypo.default

*Path anomalies (HYPA) in sequences***Description**

The sequence method of `hypo()`. Builds the De Bruijn graph of order `order` from the sequences and scores every possible path of order + 1 states (every observed path, plus every unobserved path that the observed transitions permit) against a hypergeometric null whose propensities are fitted so that the expected counts preserve the in- and out-strength of every De Bruijn node (LaRock et al. 2020). Paths observed far more often than the null predicts are over-represented, far less often under-represented. A path is under-represented when the lower tail $P(X \leq f)$ of its observed count f , adjusted by `p_adjust`, is below `alpha`, and otherwise over-represented when the strict upper tail $P(X > f)$, adjusted the same way, is below `alpha`.

Usage

```
## Default S3 method:
hypo(
  x,
  order = 2L,
  alpha = 0.05,
  min_count = 5L,
  p_adjust = "none",
  type = c("all", "over", "under"),
  order_by = c("sig", "ratio", "freq", "path"),
  n = 10L,
  action = NULL,
  actor = NULL,
  time = NULL,
  session = NULL,
  time_threshold = 900,
  timezone = "UTC",
  ...
)

## S3 method for class 'hypergraphs_hypo'
print(x, n = NULL, ...)
```

Arguments

`x` Sequences in any form described in [sequence-input](#): a long event table (with action), a wide data.frame, a list of vectors, or a model object carrying its sequences.

order	Integer (or integer vector) of De Bruijn orders. Default 2.
alpha	Significance level of the anomaly labels, in (0, 0.5). Default 0.05.
min_count	Single whole number ≥ 1 . A path is over-represented only if it is observed at least min_count times and under-represented only if it is expected at least min_count times, so a path that never occurs can still be under-represented. Default 5.
p_adjust	Multiplicity correction: any <code>stats::p.adjust</code> method or "none". Default "none": LaRock et al. (2020) classify against alpha without correction. "BH" controls the false discovery rate; the two tails are adjusted separately, within each order, over every possible path.
type	Which anomalous paths the printed result lists: "all" (default), "over" (paths observed more often than expected) or "under" (less often).
order_by	How the printed paths are ordered: "sig" (default, smallest p-value of the path's own direction first), "ratio" (largest deviation first: highest observed/expected ratio for over-represented paths, lowest for under-represented ones), "freq" (most observed first) or "path" (alphabetical).
n	Number of paths printed per direction. Default 10.
action, actor, time, session, time_threshold, timezone	Long-format arguments (see sequence-input); leave the column names NULL for wide, list or model input.
...	Must be empty: an argument that only the hypergraph method takes (top) raises <code>hypergraphs_bad_input</code> .

Value

A `net_hypo` object (also a `cograph_network`) that prints the selected anomalous paths. `type`, `order_by` and `n` change only what is printed and the default of `hg_get()`; every path is scored. Read the tables with `hg_get()`: the anomalous paths (what = "anomalies", the default), every scored path ("scores"), the over- or under-represented ones ("over", "under"), or the anomalous paths as pathways ("pathways").

References

LaRock, T., Nanumyan, V., Scholtes, I., Casiraghi, G., Eliassi-Rad, T., and Schweitzer, F. (2020). HYPA: Efficient detection of path anomalies in time series data on networks. *Proceedings of the 2020 SIAM International Conference on Data Mining*, 460-468. doi:10.1137/1.9781611976236.52

See Also

[hypo.net_hg\(\)](#) for the same null on hypergraph co-occurrence.

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"),
            c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
fit <- hypo(seqs, order = 2, min_count = 1, type = "over")
fit
hg_get(fit)
```

hypa.net_hg

Hypergeometric anomaly detection for co-occurring node pairs

Description

The hypergraph method of `hypa()`. Scores pairs of nodes against a hypergeometric null that keeps every node's co-occurrence strength and otherwise pairs the nodes at random. A pair that shares far more hyperedges than the null predicts is over-represented; a pair that shares far fewer, possibly none, is under-represented. This is the undirected counterpart of `hypa.default()`, which applies the same null to the paths of a higher-order network, and it reports the same columns so the two read alike.

Usage

```
## S3 method for class 'net_hg'
hypa(x, min_count = 5L, alpha = 0.05, p_adjust = "none", top = NULL, ...)
```

Arguments

<code>x</code>	A <code>net_hg</code> , or a snapshot of a temporal one.
<code>min_count</code>	Single whole number ≥ 1 (default 5L). A pair is over-represented only if it co-occurs at least <code>min_count</code> times and under-represented only if it is expected to co-occur at least <code>min_count</code> times. Pairs eligible for neither are not scored.
<code>alpha</code>	Significance threshold for the anomaly column, a single number in $(0, 0.5)$ (default 0.05), applied to the adjusted p-values.
<code>p_adjust</code>	Multiplicity correction, any <code>stats::p.adjust</code> method or "none" (default "none", as in LaRock et al. 2020). "BH" controls the false discovery rate within each direction.
<code>top</code>	Optional number of rows to keep, the most significantly over-represented first. NULL (default) keeps every scored pair.
<code>...</code>	Must be empty: an argument that only the sequence method takes (order, action, ...) raises hypergraphs_bad_input.

Details

The test is analytic, so it needs no resampling. That is what makes it usable where `hg_null_test()` is not: `hg_null_test()` answers whether the hypergraph as a whole carries more repetition than chance, by Monte Carlo, and returns one row; this returns one row per pair.

Observed counts. For nodes $i < j$, c_{ij} is the number of hyperedges that contain both (hyperedge weights are ignored; a repeated hyperedge counts each time). The total is $m = \sum_{i < j} c_{ij}$ and the strength of node i is $s_i = \sum_{j \neq i} c_{ij}$, so $\sum_i s_i = 2m$.

Null model. Every unordered pair of distinct nodes with positive strength is a possible cell, whether it co-occurs or not. The expected count is $E_{ij} = x_i x_j$, the product form of Chung and Lu (2002), with the fitness x fitted so that $\sum_{j \neq i} E_{ij} = s_i$ for every node: the null preserves each node's observed strength, as the propensity fit of LaRock et al. (2020) preserves the in- and out-strengths

of a De Bruijn graph. The fit is the damped fixed point $x_i \leftarrow \sqrt{x_i s_i / (X - x_i)}$, $X = \sum_k x_k$, iterated until the largest strength error falls below $10^{-10}m$ or 10,000 sweeps have run. Each sweep costs $O(n)$; no $n \times n$ matrix is formed. With three or more nodes the fit has a solution with positive fitness exactly when no node takes part in every co-occurring pair. In a star-like hypergraph, where one node is in every co-occurring pair, the fit cannot converge, and the result is returned with a warning of class `hypergraphs_no_converge`.

Test. The propensity of a pair is $\Xi_{ij} = mE_{ij}$, which sums to m^2 . The observed count $f = c_{ij}$ is referred to a Hypergeometric(N, K, n) law with $N = m^2$, $K = \text{round}(\Xi_{ij})$ and $n = m$ draws, whose mean is E_{ij} . `p_under` is $P(X \leq f)$ and `p_over` is the strict upper tail $P(X > f) = 1 - \text{p_under}$, so a pair is over-represented when its HYPA score $P(X \leq f)$ exceeds $1 - \alpha$, the rule of LaRock et al. (2020). With small counts the strict upper tail rejects more often than α under the null; `min_count` limits the problem.

Scored pairs and classification. A pair is scored when it could be classified: when it co-occurs at least `min_count` times (eligible for over-representation) or is expected to co-occur at least `min_count` times (eligible for under-representation, including pairs that never co-occur). There are at most $m/\text{min_count}$ pairs of the second kind, because the expected counts sum to m . A scored pair is "under" when `expected >= min_count` and `p_adjusted_under < alpha`, otherwise "over" when `observed >= min_count` and `p_adjusted_over < alpha`, otherwise "normal". `min_count` selects the hypotheses by count, never by p-value, and the null is fitted on every pair whatever the floor, so raising the floor changes which pairs are tested but never the expected counts.

Multiple testing. LaRock et al. (2020) classify against α without correction, and so does the default `p_adjust = "none"`. Any `stats::p.adjust` method can be applied instead. The under- and over-representation p-values are then adjusted separately, and each family is every possible pair ($n(n-1)/2$ for the n nodes with positive strength), as in the sequence method, which adjusts over every possible path. Only the scored pairs are computed; the others enter `stats::p.adjust()` through its `n` argument, which treats them as larger p-values. Adjusting within the scored pairs alone would under-correct, because a pair enters the scored set partly through its observed count.

Time is handled by scoring a cumulative snapshot rather than by slicing into periods. Disjoint periods split a pair's evidence across cells while multiplying the number of tests, so both terms move the wrong way; a sequence of `hg_snapshot(hg, at = ..., mode = "cumulative")` accumulates evidence instead and shows when a pair becomes anomalous.

Value

A base data.frame, one row per scored pair, most significantly over-represented first (by `p_adjusted_over`, ties broken by `ratio`, then by `name`), with columns `from`, `to` (the pair, in the node order of the hypergraph), `observed` (co-occurrence count, 0 for a pair that never co-occurs), `expected` (the fitted E_{ij}), `ratio` (observed / expected), `p_under`, `p_over`, `p_adjusted_under`, `p_adjusted_over` and `anomaly` ("over", "under" or "normal"). The attribute "hypo_fit" records the fit of the null: `converged`, `iterations`, `max_error` (largest absolute strength error) and `n_cooccurrences` (m).

Errors of class `hypergraphs_bad_input` are raised for an invalid argument, and of class `hypergraphs_empty_result` when no pair co-occurs or no pair reaches `min_count` in either sense. A warning of class `hypergraphs_no_converge` is raised when the null cannot be fitted to tolerance.

References

LaRock, T., Nanumyan, V., Scholtes, I., Casiraghi, G., Eliassi-Rad, T., and Schweitzer, F. (2020). HYPA: Efficient detection of path anomalies in time series data on networks. *Proceedings of the*

2020 *SIAM International Conference on Data Mining*, 460-468. doi:10.1137/1.9781611976236.52

Chung, F., and Lu, L. (2002). Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, 6(2), 125-145. doi:10.1007/PL00012580

See Also

[hypo.default\(\)](#) for the same null on the paths of sequences, [hg_null_test\(\)](#) for the Monte Carlo whole-hypergraph tests.

Examples

```
memberships <- data.frame(
  actor = c("a","b","c", "a","b","d", "a","b","e", "c","d","e", "a","b","f"),
  group = rep(c("g1","g2","g3","g4","g5"), each = 3)
)
hg <- group_hypergraph(memberships, node = "actor", hyperedge = "group")
hypo(hg, min_count = 2L)
```

hypergraph

Build a hypergraph from any input

Description

`hypergraph()` is the main constructor of the package. It reads the input and builds the hypergraph the input describes, through the constructor of that kind of data. A data frame is read in one of three formats, each with its own argument names, and the format is recognised from the arguments given:

Usage

```
hypergraph(data, ...)
```

Arguments

<code>data</code>	The input: a data frame, a list of sequences, a network, a topic model or a clustering of sequences.
<code>...</code>	Arguments of the constructor that the input selects.

Details

- **Membership data** name a node and a hyperedge (or from and to for an edge list): every value of hyperedge becomes a hyperedge of the nodes it holds ([group_hypergraph\(\)](#)). With time, start or end the hyperedges carry a clock ([temporal_hypergraph\(\)](#)).
- **Event data** name an action, the column of what happened, with the session and actor it belongs to, in the vocabulary of the memory family. Without window, the actions of each session become one hyperedge; a session is read within its actor when both are given, and with actor alone each actor's actions become one hyperedge. With window, every window of consecutive actions within an actor becomes a hyperedge ([window_hypergraph\(\)](#)), ordered by time.

- With `group`, `top` or `min_share`, the sets of either format are counted and the most frequent become hyperedges; `group` names the comparison variable within whose values the sets are counted.

A list of sequences is read as event data with `window`. A network, as a weight matrix, a sparse matrix, a `netobject` or a `cograph_network`, has its cliques promoted to hyperedges (`network_hypergraph()`); with `window`, a model object built from sequences is read as sequences instead. A topic model fitted by `hg_topics()`, or a clustering of sequences, becomes the hypergraph of its frequent sets (`group_hypergraph()`).

Every argument in `...` is passed to that constructor, whose documentation describes it, and the result is the constructor's own.

Value

A `net_hg` object (a `net_temporal_hypergraph` for data with a clock), identical to the result of the constructor called directly.

Conditions

`hypergraphs_bad_input` for an input of a class no constructor reads, naming the class; for event data given a clock without `window`, since the set of a session has no order; and the conditions of the constructor called.

References

Battiston, F., Cencetti, G., Iacopini, I., Latora, V., Lucas, M., Patania, A., Young, J.-G., & Petri, G. (2020). Networks beyond pairwise interactions: Structure and dynamics. *Physics Reports*, 874, 1-92. doi:10.1016/j.physrep.2020.05.004

See Also

`group_hypergraph()`, `window_hypergraph()`, `temporal_hypergraph()`, `network_hypergraph()`, `text_hypergraph()` for a corpus.

Examples

```
meetings <- data.frame(
  person = c("Alice", "Bob", "Carol", "Alice", "Bob", "Dave", "Eve"),
  meeting = c("m1", "m1", "m1", "m2", "m2", "m3", "m3"))
hypergraph(meetings, node = "person", hyperedge = "meeting")

visits <- data.frame(
  user = c("u1", "u1", "u1", "u1", "u2", "u2"),
  visit = c(1, 1, 2, 2, 1, 1),
  page = c("home", "cart", "home", "help", "home", "cart"))
hypergraph(visits, action = "page", actor = "user", session = "visit")

sessions <- list(c("a", "b", "c", "a"), c("b", "c", "d"))
hypergraph(sessions, window = 2)

weights <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3,
```

```
dimnames = list(c("x", "y", "z"), c("x", "y", "z"))
hypergraph(weights)
```

icsid_tribunals	<i>ICSID arbitration tribunals, one row per seat</i>
-----------------	--

Description

The 742 cases registered at the International Centre for Settlement of Investment Disputes between 1974 and June 2023 for which the three tribunal members and the dates are known, as released by Coupette, Hartung and Katz (2024): one row per seat, so a case has three rows. Arbitrators sharing a case are the co-occurrence data of a temporal hypergraph whose hyperedges are tribunals active from constitution to conclusion; see [temporal_hypergraph\(\)](#).

Usage

```
icsid_tribunals
```

Format

- A data frame with 2,226 rows and 10 columns:
- case** ICSID case number, the hyperedge.
 - seat** "president", "arbitrator_1" or "arbitrator_2".
 - arbitrator** Name of the arbitrator holding the seat, the node.
 - registered** Date the case was registered.
 - constituted** Date the tribunal was constituted.
 - concluded** Date the case concluded; NA for a pending case, whose tribunal stays active through the end of observation (2023-06-15).
 - is_concluded** Whether the archive marks the case as concluded.
 - economic_sector** Economic sector of the dispute.
 - subject** Subject of the dispute.
 - respondent** Respondent state.

Source

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs (ICSID dataset). Zenodo. [doi:10.5281/zenodo.8081513](#), via the replication archive [doi:10.5281/zenodo.8081507](#). Licence CC BY-NC 4.0. Built by data-raw/legal_hypergraphs.R.

References

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. [doi:10.1098/rsta.2023.0141](#)

Examples

```
head(icsid_tribunals)
tribunals <- temporal_hypergraph(icsid_tribunals, node = "arbitrator",
                                hyperedge = "case", start = "constituted",
                                end = "concluded")
summary(tribunals)
```

knn_hypergraph	<i>Build a k-nearest-neighbor hypergraph from embeddings</i>
----------------	--

Description

Every row of embeddings becomes one hyperedge containing the row itself and its k nearest neighbors by cosine similarity (ties broken deterministically by row name). With `weight = "cosine"` the incidence entries are the similarities to the hyperedge's center (1 for the center itself); `weight = "binary"` gives the unweighted membership hypergraph (the `HyperG::knn_hypergraph()` construction, which this generalizes).

Usage

```
knn_hypergraph(embeddings, k, weight = c("cosine", "binary"))
```

Arguments

<code>embeddings</code>	Numeric matrix, one row per item, with unique non-empty rownames (the item IDs). Finite values only, none missing; no all-zero rows.
<code>k</code>	Number of neighbors per hyperedge, one whole number between 1 and <code>nrow(embeddings) - 1</code> ; a fraction is refused, not truncated.
<code>weight</code>	"cosine" (default) or "binary".

Details

Cosine similarity and Euclidean distance order neighbors identically on L2-normalized embeddings, so normalized encoder output (the default of `sbert::encode()`) gives the conventional kNN structure.

Value

A `net_hg` (from `group_hypergraph()`) with one hyperedge per item, each of size $k + 1$, plus a `knn` field recording k and the weighting. Accepted by `hg_measures()`, `hg_centrality()`, `hg_cluster()`, and `hg_classify()`.

Conditions

Raises `hypergraphs_bad_input` for broken contracts, and `hypergraphs_nonpositive_similarity` when `weight = "cosine"` selects a neighbor with similarity ≤ 0 – a non-positive incidence weight would invalidate the random-walk machinery downstream; use `weight = "binary"` or a smaller k instead.

References

Feng, Y., You, H., Zhang, Z., Ji, R., & Gao, Y. (2019). Hypergraph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01), 3558-3565. doi:10.1609/aaai.v33i01.33013558

Examples

```
set.seed(1)
emb <- matrix(rnorm(20, mean = 3), nrow = 5,
              dimnames = list(paste0("doc_", 1:5), NULL))
hg <- knn_hypergraph(emb, k = 2)
hg_measures(hg, what = "edges")
```

long-data

Human-AI Vibe Coding Interaction Data (Long Format)

Description

Coded turns from 429 human-AI pair programming sessions across 34 projects, in long format: `human_long` holds the human turns (10,796 rows), `ai_long` the AI turns (8,551 rows). Each session's ordered codes form one categorical sequence, which makes the pair a natural two-cohort input for the higher-order verbs – e.g. `hg_bootstrap(human_long, action = "code", actor = "session_id", time = "timestamp")` or `hg_compare(human_long, ai_long, ...)`.

Usage

`human_long`

`ai_long`

Format

Data frames in long format with 9 columns:

message_id Integer. Turn index.

project Character. Project identifier (Project_1 .. Project_34).

session_id Character. Unique session hash.

timestamp Integer. Unix timestamp for ordering.

session_date Character. Date of the session (YYYY-MM-DD).

code Character. Interaction code.

cluster Character. High-level cluster: Directive, Evaluative, or Metacognitive (human codes); AI turns carry their own scheme.

code_order Integer. Order of the code within the session.

order_in_session Integer. Absolute turn order within the session.

An object of class `data.frame` with 10796 rows and 9 columns.

An object of class `data.frame` with 8551 rows and 9 columns.

Details

The same data feed all three structure families: the ordered codes are sequences for the memory family, a session is a natural hyperedge over the codes that co-occur in it (`group_hypergraph()`), and a fitted memory network becomes a pathway complex for the simplicial family (`simplicial()` with `type = "pathway"`).

Source

Saqr, M. (2026). Human-AI vibe coding interaction study. <https://saqr.me/blog/2026/human-ai-interaction-cograp>

Examples

```
bs <- hg_bootstrap(human_long, action = "code", actor = "session_id",
                  time = "timestamp", n_boot = 20, max_order = 2,
                  seed = 1)
rules <- hg_get(bs, order_min = 2)
head(rules)
```

markov_order

Test the Markov order of sequences

Description

At each order k the likelihood-ratio statistic G^2 tests whether the k -th state back adds information about the next state given the $k - 1$ most recent ones; its null distribution comes from an exact within-context permutation. The order is raised while the test rejects.

Usage

```
markov_order(
  data,
  max_order = 3L,
  n_perm = 500L,
  alpha = 0.05,
  parallel = FALSE,
  n_cores = 2L,
  seed = NULL,
  action = NULL,
  actor = NULL,
  time = NULL,
  session = NULL,
  time_threshold = 900,
  timezone = "UTC"
)
```

Arguments

data	Sequences in any form described in sequence-input ; a netobject_group gives one test per group.
max_order	Integer. Highest order tested. Default 3.
n_perm	Integer. Permutations per order. Default 500.
alpha	Significance level of the sequential selection. Default 0.05.
parallel	Logical. Run the permutations with <code>parallel::mclapply()</code> (not on Windows). Default FALSE.
n_cores	Integer. Cores when <code>parallel = TRUE</code> . Default 2.
seed	Integer or NULL. Seed for the permutations.
action, actor, time, session, time_threshold, timezone	Long-format arguments (see sequence-input); leave the column names NULL for wide, list or model input.

Value

A `net_markov_order` object (a `net_markov_order_group` for a `netobject_group`). Read it with `hg_get()`: the per-order test table (what = "orders", the default) or the permutation null draws ("null").

References

Anderson, T. W., & Goodman, L. A. (1957). Statistical inference about Markov chains. *The Annals of Mathematical Statistics*, 28(1), 89–110. doi:10.1214/aoms/1177707039

Good, P. (2005). *Permutation, Parametric and Bootstrap Tests of Hypotheses* (3rd ed.). Springer.

See Also

[mogen\(\)](#), [memory\(\)](#)

Examples

```
seqs <- list(c("a", "b", "a", "c", "a", "b"), c("b", "a", "c", "a", "b", "a"))
fit <- markov_order(seqs, max_order = 2L, n_perm = 20L, seed = 1L)
hg_get(fit)
```

memory

Per-context path dependence

Description

For every context of order states (the order most recent states), compares the observed next-state distribution with the first-order prediction from the most recent state alone: the Kullback-Leibler divergence and entropy drop say how much the longer history adds, and `flips` marks the contexts where it changes the most likely next state.

Usage

```
memory(
  data,
  order = 2L,
  min_count = 5L,
  base = 2,
  action = NULL,
  actor = NULL,
  time = NULL,
  session = NULL,
  time_threshold = 900,
  timezone = "UTC"
)
```

Arguments

data	Sequences in any form described in sequence-input : a long event table (with action), a wide data.frame, a list of vectors, or a model object carrying its sequences.
order	Integer ≥ 2 . The number of states in the conditioning context: 2 conditions on the last two states (contexts such as "a -> b") and compares that two-step memory with one-step, 3 on the last three. Default 2.
min_count	Integer. Contexts observed fewer times are dropped. Default 5.
base	Logarithm base of the entropies and divergences. Default 2 (bits).
action, actor, time, session, time_threshold, timezone	Long-format arguments (see sequence-input); leave the column names NULL for wide, list or model input.

Value

A `net_path_dependence` object. Read it with `hg_get()`: one row per context (what = "contexts"), sorted by divergence, largest first.

References

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1), 79–86. doi:[10.1214/aoms/1177729694](https://doi.org/10.1214/aoms/1177729694)

Cover, T. M., & Thomas, J. A. (2006). *Elements of Information Theory* (2nd ed.). Wiley.

See Also

[markov_order\(\)](#), [mogen\(\)](#)

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"),
             c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
# order = 2: two-state contexts ("a -> b")
```

```
pd <- memory(seqs, order = 2, min_count = 1)
hg_get(pd)
# order = 3: three-state contexts ("a -> b -> c")
hg_get(memory(seqs, order = 3, min_count = 1))
```

mogen

Multi-order generative model (MOGen) of sequences

Description

Fits Markov chains of every order from 0 to `max_order` as layers of one multi-order model (higher-order De Bruijn graphs) and selects the order that best balances fit and parsimony.

Usage

```
mogen(
  data,
  max_order = 5L,
  criterion = c("aic", "bic", "lrt"),
  lrt_alpha = 0.01,
  action = NULL,
  actor = NULL,
  time = NULL,
  session = NULL,
  time_threshold = 900,
  timezone = "UTC"
)
```

Arguments

<code>data</code>	Sequences in any form described in sequence-input : a long event table (with action), a wide data.frame, a list of vectors, or a model object carrying its sequences.
<code>max_order</code>	Integer. Highest Markov order tested. Default 5; capped at the longest sequence minus one, with a message.
<code>criterion</code>	"aic" (default), "bic" or "lrt" (likelihood-ratio test).
<code>lrt_alpha</code>	Significance level of the likelihood-ratio test. Default 0.01.
<code>action, actor, time, session, time_threshold, timezone</code>	Long-format arguments (see sequence-input); leave the column names NULL for wide, list or model input.

Value

A `net_mogen` object (also a `cograph_network`). Read it with `hg_get()`: the order-selection table (what = "orders", the default), the fitted transitions of a layer ("transitions"), the path counts ("paths", with `k =`) and the pathways ("pathways").

References

- Scholtes, I. (2017). When is a network a network? Multi-order graphical model selection in pathways and temporal networks. *Proceedings of KDD 2017*, 1037–1046. doi:[10.1145/3097983.3098145](https://doi.org/10.1145/3097983.3098145)
- Gote, C., Casiraghi, G., Schweitzer, F., & Scholtes, I. (2023). Predicting variable-length paths in networked systems using multi-order generative models. *Applied Network Science*, 8, 68. doi:[10.1007/s41109023005960](https://doi.org/10.1007/s41109023005960)

See Also

[markov_order\(\)](#), [memory\(\)](#)

Examples

```
seqs <- list(c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"),
            c("a", "b", "c", "a", "b", "c"), c("x", "b", "d", "x", "b", "d"))
mg <- mogen(seqs, max_order = 2)
hg_get(mg)
hg_get(mg, what = "paths", k = 3)
```

network_hypergraph	<i>Higher-order hypergraph from a network's clique structure</i>
--------------------	--

Description

Takes a network and produces a hypergraph by promoting k -cliques ($k \geq 3$) to k -hyperedges. Each k -clique is independently included as a k -hyperedge with probability p . Optionally retains the underlying pairwise edges as 2-hyperedges. Foundation for higher-order analyses.

Usage

```
network_hypergraph(
  net,
  p = 1,
  type = c("clique", "vr", "rips"),
  include_pairwise = TRUE,
  max_size = 3L,
  threshold = 0,
  seed = NULL
)

## S3 method for class 'net_hg'
print(x, n = 10L, ...)

## S3 method for class 'net_hg'
summary(object, ...)
```

Arguments

net	A netobject, cograph_network, simplicial_complex, or numeric adjacency / weight matrix. Directed inputs are symmetrised by the underlying clique enumerator.
p	Probability in $[0, 1]$ that each k -clique with $k \geq 3$ becomes a k -hyperedge. Default 1 (deterministic - every found clique is included).
type	Hyperedge enumeration. "clique" (default) promotes k -cliques in the binarised adjacency to k -hyperedges. A metric Vietoris-Rips construction is not implemented ; "vr" / "rips" are accepted by <code>match.arg</code> but raise an error rather than silently aliasing "clique".
include_pairwise	Logical. Include 2-edges from the input network as 2-hyperedges. Default TRUE. Set FALSE for a "fully higher-order" hypergraph containing only k -hyperedges with $k \geq 3$.
max_size	Integer ≥ 2 . Maximum hyperedge size to extract. Default 3L (triangles only). 4L also includes 4-cliques as 4-hyperedges, etc.
threshold	Numeric. Edge weight cutoff used to binarise the adjacency for clique enumeration. Default 0 (any non-zero weight is an edge).
seed	Optional integer for reproducible Bernoulli sampling when $0 < p < 1$.
x	A net_hg object (for print).
n	Number of rows of the default table to print. Default 10.
...	Additional arguments (ignored).
object	A net_hg object (for summary).

Details

The construction follows Burgio, Matamalas, Gomez & Arenas (2020) on simplicial / hypergraph contagion. For each k -clique with $k \geq 3$ found in the underlying graph (via `simplicial()`), an independent Bernoulli(p) trial decides whether that clique becomes a k -hyperedge. Underlying pairwise edges are always retained when `include_pairwise = TRUE`, so the resulting hypergraph contains both the original 2-edge structure and the sampled higher-order interactions.

At the limits:

- $p = 0$ with `include_pairwise = TRUE` reproduces the input pairwise network as a hypergraph of size-2 edges.
- $p = 1$ with `include_pairwise = FALSE` returns a fully higher-order hypergraph containing only the k -hyperedges ($k \geq 3$) found in the network's clique complex.

Value

A net_hg object: a list with components

hyperedges List of integer vectors. Each entry is a hyperedge given as the sorted node indices it spans.

incidence Numeric matrix of size `n_nodes` x `n_hyperedges`. `incidence[i, j] = 1` iff node i belongs to hyperedge j . Row names are node names; column names are `h1, h2, ...`

nodes Character vector of node names.
 n_nodes, n_hyperedges Scalar counts.
 size_distribution Named integer vector: number of hyperedges of each size, named size_2, size_3, ...
 params Recorded call parameters: source, type, p, include_pairwise, max_size, threshold, seed.

The input `x` invisibly.

A `data.frame`, one row per node: node, degree (the number of hyperedges the node belongs to).
 Returned **invisibly**: `summary(x)` prints the summary and nothing else; assign the result to keep the table.

References

Burgio, G., Matamalas, J. T., Gomez, S., & Arenas, A. (2020). Evolution of cooperation in the presence of higher-order interactions: from networks to hypergraphs. *Entropy* 22(7), 744. doi:10.3390/e22070744

See Also

`simplicial()` (underlying clique enumeration).

Examples

```
set.seed(1)
n <- 8
adj <- matrix(stats::rbinom(n * n, 1, 0.5), n, n)
diag(adj) <- 0
adj <- (adj + t(adj)) > 0
rownames(adj) <- colnames(adj) <- LETTERS[seq_len(n)]
hg <- network_hypergraph(adj, p = 1, max_size = 3L)
print(hg)
summary(hg)
```

Description

Projects a hypergraph onto a network of pairs of its nodes. Two nodes are joined when they belong to a common hyperedge, and the weight of the edge depends on type. "clique" is the clique expansion: a hyperedge of size $|e|$ contributes the same amount to each of its $|e|(|e| - 1)/2$ pairs, so the weight of a pair is the number of hyperedges that contain both nodes (Zhou et al. 2006). "association" divides the contribution of each hyperedge by $|e| - 1$, following Coupette et al. (2024):

Usage

```
pairwise_network(
  hg,
  type = c("clique", "association", "citation"),
  weighted = NULL,
  duplicate_edges = c("count", "collapse"),
  self_association = FALSE,
  edge_source = NULL,
  directed = FALSE
)
```

Arguments

hg	A hypergraph (net_hg), such as one built by hypergraph() or text_hypergraph() .
type	The projection: "clique" (default), "association" or "citation".
weighted	type = "clique" only. TRUE (default) uses the incidence weights, FALSE their membership pattern. Setting it with another type is an error, because those projections are defined on membership.
duplicate_edges	For type = "association" or "citation": "count" (default) lets repeated hyperedges contribute repeatedly (the multi-hypergraph); "collapse" lets each distinct member set contribute once (the binary hypergraph).
self_association	For type = "association": add the source-to-member association term of Coupette et al. (2024). Requires one source node per hyperedge through edge_source or hg\$edge_data\$source. For source u, every membership of target v contributes $1 / \sum_e e $ over the hyperedges sourced by u.
edge_source	The node each hyperedge comes from: the name of a column of the hyperedge attributes, a vector of length n_hyperedges, a named vector keyed by hyperedge, or a data frame with columns edge and source (a hyperedge named twice with different sources raises hypergraphs_bad_input). NULL uses an attribute column named source. Used with self_association = TRUE and type = "citation".
directed	For type = "citation": keep the source-to-member direction. Default FALSE.

Details

$$w(\{u, v\}) = \sum_{e \supseteq \{u, v\}} \frac{1}{|e| - 1}$$

so the weight a hyperedge adds around each of its members is 1, and the weighted degree of a node equals the number of hyperedges of size at least two that contain it. A hyperedge of size one has no pairs and contributes nothing.

"citation" is the graph of a hypergraph whose hyperedges have sources: one edge from the source of every hyperedge to each of its members, so a hypergraph of citation blocks becomes the ordinary citation graph. duplicate_edges = "count" weights an edge by the number of blocks that repeat

it (the multi-graph); "collapse" keeps the binary graph. directed = TRUE keeps the source-to-member orientation; the default symmetrises.

The result is a network object that cograph plots and that `hypergraph()` reads as a network, so the cliques of the projection can be promoted back to hyperedges. Its edges are read with `hg_get()`.

Value

A `net_hg_pairwise`, which is also a `netobject` and a `cograph_network`: the weighted adjacency matrix of the projection (weights, zero diagonal, symmetric unless directed = TRUE), its nodes and edges. `hg_get(x)` returns the edges as a data frame with one row per pair of non-zero weight and columns from, to and weight, sorted by from then to.

Conditions

`hypergraphs_bad_input` for an argument that does not apply to type, and for `self_association` or type = "citation" without a source per hyperedge.

References

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. doi:10.1098/rsta.2023.0141

Zhou, D., Huang, J., & Schoelkopf, B. (2006). Learning with hypergraphs: clustering, classification, and embedding. *NeurIPS 19*, 1601-1608. doi:10.7551/mitpress/7503.003.0205

See Also

`hg_line_graph()` for the projection onto hyperedges.

Examples

```
meetings <- data.frame(
  person = c("Alice", "Bob", "Carol", "Alice", "Bob", "Dave", "Eve"),
  meeting = c("m1", "m1", "m1", "m2", "m2", "m3", "m3"))
meeting_hg <- hypergraph(meetings, node = "person", hyperedge = "meeting")
meeting_network <- pairwise_network(meeting_hg)
hg_get(meeting_network)
association_network <- pairwise_network(meeting_hg, type = "association")
hg_get(association_network)
```

plot.hypergraphs_bootstrap

Plot a memory-network bootstrap

Description

Forest plot of the rule-edge probabilities with their bootstrap percentile intervals, the most frequent edges first. Order is encoded by both colour (Okabe-Ito) and point shape.

Usage

```
## S3 method for class 'hypergraphs_bootstrap'
plot(x, top = 20L, ...)
```

Arguments

x	A hypergraphs_bootstrap object.
top	Integer. Number of edges to show (by descending count). Default 20.
...	Additional arguments (ignored).

Value

A ggplot object, invisibly.

```
plot.hypergraphs_comparison
  Plot a memory-network comparison
```

Description

Difference plot of the per-edge probability differences, the largest absolute differences first. Significance (BH-adjusted) is encoded by both colour and point shape.

Usage

```
## S3 method for class 'hypergraphs_comparison'
plot(x, top = 20L, ...)
```

Arguments

x	A hypergraphs_comparison object.
top	Integer. Number of edges to show (by descending absolute difference). Default 20.
...	Additional arguments (ignored).

Value

A ggplot object, invisibly.

plot.hypergraphs_memory_communities

Plot memory-network communities

Description

type = "physical" (**default**) every community as a pebble around the states it holds, through `plot.net_hg()` on the community hypergraph. A state shared by several communities lies inside each of their pebbles. Every state is a circle whose area follows its flow, with a triangle pointing at the state the walk most often moves to next, and every community has a title box with its flow.

type = "network" the network of states as a transition network analysis plot through `cograph::overlay_communities()`: every arrow is a transition with its probability (transitions below 0.05 hidden), every state a circle whose area follows its flow, coloured (Okabe-Ito) by the community that holds most of its flow, and every community a blob around its states, so a shared state lies in two blobs.

type = "states" the network of memory nodes drawn in the same way, laid out by the walk's transitions between them so that the nodes of a community sit together. Memory nodes with no flow are left out unless `show_zero_flow = TRUE`, and the title says how many. With more than 20 nodes the probabilities are not printed on the arrows.

Memory nodes with no flow (reached only by random jumps, see `hg_communities()`) carry no flow of their states and are never drawn in the physical view. The tables returned by `hg_get.hypergraphs_memory_communities()` are unaffected.

Usage

```
## S3 method for class 'hypergraphs_memory_communities'
plot(x, type = c("physical", "network", "states"), show_zero_flow = FALSE, ...)
```

Arguments

x	A hypergraphs_memory_communities object.
type	"physical" (default), "network" or "states".
show_zero_flow	Draw the memory nodes with no flow in the type = "states" view? Default FALSE.
...	For "physical", passed to <code>plot.net_hg()</code> (e.g. seed, label_size, arrow_style = "outside", layout); for "network" and "states", passed to <code>cograph::overlay_communities()</code> and on to <code>cograph::splot()</code> (e.g. layout, edge_label_style, threshold, blob_alpha). They override the defaults set here.

Value

For "physical", a ggplot object (print it to draw). For "network" and "states", x, invisibly (cograph draws with base graphics).

Conditions

hypergraphs_bad_input from `plot.net_hg()` for arguments passed through ... that it rejects.

References

Saqr, M., López-Pernas, S., Törmänen, T., Kaliisa, R., Misiejuk, K., & Tikka, S. (2025). Transition network analysis: A novel framework for modeling, visualizing, and identifying the temporal patterns of learners and learning processes. *Proceedings of the 15th International Learning Analytics and Knowledge Conference (LAK '25)*, 351-361. doi:10.1145/3706468.3706513

Examples

```
seqs <- list(c("a", "h", "b", "a", "h", "b", "a"),
             c("c", "h", "d", "c", "h", "d", "c"))
comm <- hg_communities(hon(seqs, max_order = 2L), trials = 2L)
plot(comm)
plot(comm, type = "network")
plot(comm, type = "states")
```

plot.net_hg

Plot a hypergraph with hyperedges as pebbles

Description

Draws every hyperedge as a smooth pebble around its members – the convex hull of the members, widened by padding and with its corners smoothed away – in a translucent fill parted from its neighbours by a white seam. Every member is guaranteed to lie inside its pebble. The default layout places the bipartite star expansion (nodes and hyperedges together), which gives each hyperedge a position of its own among its members; hulls fan out instead of piling onto shared nodes, and hyperedge labels sit at those positions. Hulls can be coloured by hyperedge size, by a column of the edge metadata, or by any value supplied per hyperedge, and outlined with a line type by the same kinds of selector. A hyperedge with a single member is drawn as its node alone.

Usage

```
## S3 method for class 'net_hg'
plot(
  x,
  layout = c("bipartite", "spring", "circle"),
  center = NULL,
  seed = 1L,
  color_by = NULL,
  linetype_by = NULL,
  labels = TRUE,
  label_size = NULL,
  edge_labels = FALSE,
  edge_label_size = 3,
```

```

dismantled = FALSE,
ncol = NULL,
node_size = 2.5,
detail = 5,
outline = "white",
alpha = 0.5,
linewidth = 1.1,
padding = 0.045,
legend_title = NULL,
node_sizes = NULL,
direction = NULL,
arrow_style = c("inside", "outside"),
node_fill = "#000000",
arrow_fill = NULL,
transitions = NULL,
size_title = NULL,
titles = NULL,
title_prefix = "",
notes = NULL,
unit = NULL,
pieces = NULL,
title_gap = 0.06,
group = NULL,
node_groups = NULL,
type = c("hulls", "incidence", "storyline"),
sort_by = NULL,
...
)

```

Arguments

x	A net_hg with at least one hyperedge of two or more members. Draw a part of a large hypergraph by passing <code>hg_subset()</code> first.
layout	"bipartite" (default; a Fruchterman-Reingold layout of the star expansion, nodes and hyperedges placed together), "spring" (the same on the clique projection), "circle", or a data.frame with node, x and y columns to reuse node coordinates across panels (hyperedges then sit at the centroid of their members). The two force-directed layouts place each connected component separately and pack the results; a layout table is used exactly as given.
center	Node names to place in the middle of the picture, such as the outcome states every hyperedge ends in. Under "bipartite" and "spring" they are held on a small ring at the centre while the rest of the layout arranges around them; under "circle" they sit inside the ring of the other nodes. Names that are not nodes of this hypergraph are skipped, so one vector can serve several subsets; an error is raised only when none of them is. NULL (default) places every node freely. Not used with a layout table.
seed	Seed for the force-directed layouts (default 1); the caller's random number stream is left untouched.

color_by	Colour of the hulls: NULL (the hyperedges' count attribute described above, the window counts of a <code>window_hypergraph()</code> , else one colour), "size" (hyperedge cardinality, one Okabe-Ito colour per size), "weight" (the hyperedge weights of a <code>window_hypergraph()</code> , the number of windows behind each hyperedge), the name of a column in the edge metadata (<code>x\$edge_data</code>), a vector named by hyperedge, or a vector with one value per hyperedge. Character or factor values get the Okabe-Ito palette; numeric values a sequential scale built from it.
linetype_by	Outline line type of the hulls, resolved like color_by; discrete only.
labels	TRUE (default) writes the node names, FALSE writes none, and a character vector named by node replaces the names shown.
label_size	Text size of the node labels, in mm. NULL (default) scales it to the number of labelled nodes n : 4.2 up to 12 labels, then $4.2 * \sqrt{12 / n}$, the side of the area each label can claim, down to a floor of 2.2 (reached at 44 labels). A number is used as given. With type = "incidence" it sizes the row names instead.
edge_labels	Name the hyperedges on the figure. FALSE (default) writes nothing, TRUE writes the hyperedge names, or pass a vector named by hyperedge to write something else. Under the bipartite layout each label sits at the hyperedge's own position; under the projection layouts, just outside the member furthest from the centre, clear of the crowded overlap.
edge_label_size	Text size for edge_labels (default 3), and for the panel titles when dismantled = TRUE. A title is never cut to fit its panel – a clipped hyperedge name reads as a different hyperedge – so lower this, or ncol, if long names collide.
dismantled	Draw one panel per hyperedge instead of one figure with every hull overlaid (default FALSE). All panels share the layout, so positions are comparable; each panel greys out the non-members and is titled with its hyperedge name. edge_labels does not apply.
ncol	Columns in the panel grid when dismantled = TRUE (default: roughly square).
node_size	Point size of the nodes (default 2.5).
detail	How much of the hull's shape the smoothing keeps: the width, in harmonics, of the Gaussian low-pass applied to the outline. The default 5 gives rounded, tapering petals; 3 is rounder still, 8 stays close to the members, and Inf draws the unsmoothed rounded hull.
outline	Outline colour: "white" (default) draws seams that part overlapping hyperedges; "fill" outlines each hyperedge in its own colour (pair it with a lower alpha, e.g. 0.15); any other colour is used as given.
alpha	Fill transparency (default 0.5); the outline is always drawn at full strength.
linewidth	Width of the outlines (default 1.1).
padding	Room around the member positions as a fraction of the layout extent (default 0.045). It also sets how far apart the layout packs disconnected components, so that two components' pebbles never touch and imply a member they do not share.

legend_title	Legend title for color_by; defaults to the selector's name.
node_sizes	Size every node by a value, such as how often the node occurs: a data.frame with a node column and a value column, or a numeric vector named by node. The columns are node and value when those names are present; otherwise the one character (or factor) column holds the node names and the one numeric column the values, so a table such as data.frame(state, trials) is read as it is. Every node of the hypergraph needs a non-negative value; names that are not nodes are skipped, so one table serves every hg_subset(). Without direction the nodes are points whose area follows the value (ggplot2::scale_size_area(), largest 10 mm), with a size legend, and each label sits just above its dot. With direction they are circles in data units (the figure has a fixed 1:1 aspect): the largest has a radius of 4% of the layout extent and none is smaller than 30% of that radius, so the rarest nodes stay visible, and a caption says what area and triangle show. NULL (default) draws plain points.
direction	Mark on each circle where the walk goes next: a data.frame with columns from, to and a weight – weight, or else the one other numeric column, such as trials (how often to follows from; repeated pairs are summed). Each node gets a triangle pointing at the other node that follows it with the largest weight, ties broken by node name; a node that nothing else follows (a sink, or only itself) gets no triangle. Rows naming a node that is not in the hypergraph are skipped. Requires node_sizes.
arrow_style	"inside" (default; a triangle inscribed in the circle, its apex on the rim facing the next node) or "outside" (the circle drawn out into a tip beyond its rim). An outside tip that points upward moves that node's label below the circle.
node_fill	Colour of the circles drawn by node_sizes with direction (default black).
arrow_fill	Colour of the direction triangles. NULL (default) is a dark slate ("#1F3A44") for "inside" and node_fill for "outside".
transitions	Draw the moves between nodes as curved arrows: a data.frame with from, to and a weight, read like direction. Line width follows the weight; a node followed by itself gets a small loop below it. Rows naming a node that is not in the hypergraph are skipped. The width legend is titled with the capitalised unit followed by "making the transition" when unit is given, and "Transition weight" otherwise.
size_title	What the node area shows (e.g. "trials with the event"): the caption reads "Circle area: " followed by it and the point size legend is titled with it. NULL (default) uses the unit followed by "with the event" when there is a unit, and otherwise says the area is proportional to the event value.
titles	Write a title box beside every hyperedge, just outside its pebble on the side facing away from the centre of the figure. NULL (default) writes the hyperedges' count attribute described above (not when dismantled = TRUE), else none; FALSE writes none; TRUE writes the hyperedge names; a selector resolved like color_by (an edge-metadata column, a vector named by hyperedge, or one value per hyperedge) must be numeric and adds a second line with that number and unit (e.g. "3,667 trials"). A hyperedge with a single member gets its box beside that node. Boxes are placed in order of decreasing value (name breaks ties) and a box that would sit on an earlier one moves below it.

title_prefix	Text written before each hyperedge name in its title box (default ""), such as "Mostly ".
notes	A further line for some title boxes: a character vector named by hyperedge, or a data.frame with a note column and the hyperedge names in a hyperedge, edge or group column. Hyperedges without a note get none; names that are not hyperedges are skipped. Requires titles.
unit	The word for what the values count (e.g. "trials", "steps"). It titles the color_by legend (capitalised, unless legend_title is given), follows the number in each title box, and names the node area ("trials with the event") and the transition widths. NULL (default) is the unit the hypergraph records ("sequences" for <code>group_hypergraph()</code> of a clustering), else the name of the hyperedges' count attribute described above, else those texts stay generic.
pieces	How the bipartite layout places a hypergraph that falls into disconnected pieces: "packed" (default; pieces are laid out one by one and packed into a square frame, so each takes room in proportion to its size) or "row" (every piece is laid out in a frame of its own, as if drawn alone, and the frames are set side by side 1.4 frame widths apart, in order of the piece's most prominent hyperedge – see titles – with a piece of one node at the middle of its frame). Only the bipartite layout has pieces; "row" given with another layout is an error.
title_gap	Distance of each title box beyond its pebble, in layout units (the layout spans 0 to 1; default 0.06).
group	Draw one group only: the name of a value of the hyperedges' group attribute (a hypergraph of counted sets with a single group is drawn as that group without it), such as "Cluster 1" of a <code>group_hypergraph()</code> built from a clustering of sequences. Only that group's hyperedges and their members are drawn; for clustered sequences each node is sized by the sequences of the group containing it (unless node_sizes is given), each set is named by the states it adds to those every drawn set shares (listed in the subtitle), the figure is titled with the group and its number of sequences, titles defaults to FALSE and the legends are stacked. A name that is not a group, or a hypergraph without a group attribute, raises <code>hypergraphs_bad_input</code> . NULL (default) draws every hyperedge.
node_groups	Colour and shape every node by its group, such as its community: an <code>hg_communities()</code> , <code>hg_mmsbm()</code> or <code>hg_topics()</code> fit, a <code>hg_cluster()</code> or <code>hg_classify()</code> result, or a data.frame with a node column and a community, cluster, predicted, topic, block or label column (read as <code>hg_agreement()</code> reads a labeling). Groups take the Okabe-Ito colours in their natural order and a shape each; a node without a group is plotted as a small grey point. Without color_by the hulls are then grey, so the node colours carry the grouping. NULL (default) plots every node alike. Raises <code>hypergraphs_bad_input</code> with node_sizes, direction, transitions, dismantled = TRUE or outline = "fill".
type	"hulls" (default) plots each hyperedge as a pebble around its members; "incidence" plots the incidence matrix; "storyline" plots the hyperedges in order as columns and the nodes as lines (see Details).
sort_by	Column order of type = "incidence": NULL (default; by highest member row, then name), "size", the name of a column in the edge metadata such as a date, a vector named by hyperedge, or a vector with one value per hyperedge. Numbers

sort largest first, as in `hg_get()`; dates, text and a clock column (start, end, time, ... as `temporal_hypergraph()` names them, numeric in a snapshot) sort ascending. Ties fall back to the default order. For type = "storyline" it is the order of the columns, always ascending; NULL keeps the stored order of the hyperedges.

... For type = "storyline": top (the number of nodes with the most hyperedges drawn as lines, default 8, NULL for all), spacing ("even" or "strength"), width_by (NULL or "degree") and point_size, as in `plot.net_temporal_hypergraph()`. Otherwise unused.

Details

A hypergraph whose hyperedges carry one numeric attribute – such as a trials count that `group_hypergraph()` kept because it is constant within each group – is drawn by that attribute with no further argument: it colours the pebbles, writes a title box with the count beside each one, and names the unit of the legends. Of several numeric attributes, the one that varies between hyperedges is read when only one does (a membership share of 1 on every hyperedge is passed over); otherwise none is. `color_by`, `titles` and `unit` override it.

Node labels are bold with a white halo, the legends sit below the plot (a colour bar 2.5 cm per key, discrete legends at most four keys to a row) and the plot has wide margins (40, 130, 30 and 130 pt) so title boxes outside the pebbles are not cut. The look of earlier versions is partly available through arguments: `alpha = 0.45` and `label_size = 3`; the legend position and margins through `ggplot2::theme()` added to the result.

A hypergraph whose hyperedges repeat the same members, such as the trials of an event log, is drawn as its distinct sets: each set once, coloured by its number of copies, with each node sized by the hyperedges that contain it. A call that styles or labels the hyperedges one by one (`titles`, `color_by`, `linetype_by`, `edge_labels`, `notes`, `dismantled`), or groups the nodes (`node_groups`), draws every hyperedge.

A hypergraph that falls into several disconnected pieces is laid out one piece at a time and the pieces are then packed into a roughly square frame. A force-directed layout has no force at all between two disconnected components – they repel and nothing pulls back – so laying the whole hypergraph out at once lets a stray hyperedge drift to the edge of the picture and set the scale for everything else, leaving the connected structure a speck in the middle. Packing keeps every component at its own size and the frame spent on structure.

type = "storyline" plots a hypergraph whose hyperedges have an order, such as the windows of one sequence from `window_hypergraph(collapse = FALSE)`, as a storyline (Tanahashi and Ma 2012): each hyperedge is a column, in stored order or by `sort_by`, and each of the busiest nodes a line that its hyperedges gather, laid out as for `plot.net_temporal_hypergraph()`. It takes `sort_by`, `edge_labels` (FALSE drops the column names) and the storyline arguments in ...

type = "incidence" plots the incidence matrix instead, after UpSet (Lex et al. 2014). Each node is a row and each hyperedge a column; a vertical bar joins the members of a hyperedge. Rows are ordered by decreasing hyperdegree, which a bar at the right of each row also shows, and columns by `sort_by`, then by their highest member row, so hyperedges that share the busiest nodes stand together. A bar above each column shows the hyperedge's size when sizes differ. Empty cells are grey points up to 20,000 cells. Every hyperedge is a column, repeated sets included. The figure grows linearly with the number of hyperedges, so it stays legible where hulls overlap into one shape. It takes `color_by` (discrete values also get a point shape each), `legend_title`, `labels`,

edge_labels (the column names), label_size and edge_label_size (row and column text in mm; defaults 8 and 7 pt).

Value

A ggplot object (with dismantled = TRUE, one facet per hyperedge). With node_sizes and direction the nodes are polygon layers in data units and the caption names what circle area and triangle show; with node_sizes alone they are points with a size legend. Title boxes are the last layer.

Conditions

hypergraphs_bad_input for an invalid selector or layout, a node_sizes that misses a node or holds a negative value, a direction or transitions table without from, to and one non-negative numeric weight, a direction without node_sizes, titles that are not numeric, notes without titles, an invalid unit, title_prefix or title_gap, pieces = "row" with a layout other than "bipartite", or node overlays or titles combined with dismantled = TRUE; with type = "incidence", any argument of the hull plot other than those listed in Details, and sort_by with type = "hulls".

References

- Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. doi:10.1098/rsta.2023.0141
- Fruchterman, T. M. J., & Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11), 1129-1164. doi:10.1002/spe.4380211102
- Tanahashi, Y., & Ma, K.-L. (2012). Design considerations for optimizing storyline visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 18(12), 2679-2688. doi:10.1109/TVCG.2012.212
- Lex, A., Gehlenborg, N., Strobel, H., Vuillemot, R., & Pfister, H. (2014). UpSet: Visualization of intersecting sets. *IEEE Transactions on Visualization and Computer Graphics*, 20(12), 1983-1992. doi:10.1109/TVCG.2014.2346248

Examples

```
dat <- data.frame(
  member = c("a", "b", "c", "b", "c", "d", "d", "e", "f"),
  event = c("e1", "e1", "e1", "e2", "e2", "e2", "e3", "e3", "e4")
)
hg <- group_hypergraph(dat, "member", "event")
plot(hg, color_by = "size", edge_labels = TRUE)
plot(hg, detail = Inf, outline = "fill", alpha = 0.15)
plot(hg, center = c("b", "c"))
plot(hg, dismantled = TRUE)
plot(hg, type = "incidence", edge_labels = TRUE)

# the windows of one sequence, in order, as a storyline
steps <- list(c("plan", "monitor", "discuss", "plan", "adapt", "monitor",
               "discuss", "consensus"))
windows <- window_hypergraph(steps, window = 3, collapse = FALSE)
plot(windows, type = "storyline", top = NULL)
```

```

# tribunals in claims against Argentina, one column per tribunal by date
tribunals <- group_hypergraph(icsid_tribunals, node = "arbitrator",
                             hyperedge = "case")
argentina <- hg_subset(tribunals,
                      where = list(respondent = "Argentine Republic"),
                      component = "largest")
plot(argentina, type = "incidence", color_by = "economic_sector",
      sort_by = "constituted")

# disconnected pieces are laid out separately and packed
apart <- group_hypergraph(
  data.frame(member = c("a", "b", "c", "b", "c", "d", "x", "y", "z"),
             event = c("e1", "e1", "e1", "e2", "e2", "e2", "e3", "e3", "e3")),
  "member", "event"
)
plot(apart, color_by = "size")

# circles sized by how often each node occurs, pointing where it goes next
counts <- c(a = 40, b = 25, c = 12, d = 30, e = 5, f = 2)
moves <- data.frame(from = c("a", "b", "c", "d", "d", "e"),
                    to = c("b", "c", "a", "e", "b", "f"),
                    weight = c(20, 9, 4, 8, 3, 1))
plot(hg, node_sizes = counts, direction = moves,
     size_title = "occurrences of the node")
plot(hg, node_sizes = counts, direction = moves, transitions = moves,
     arrow_style = "outside")

# groups of events with their trial counts: the count colours the pebbles
# and writes a title box per group; points sized by how many trials
# contain each event
members <- data.frame(
  group = c("Hint", "Hint", "Hint", "Question", "Question", "Question"),
  state = c("Wrong", "Hint", "Retry", "Wrong", "Question", "Retry"),
  trials = c(120, 120, 120, 80, 80, 80)
)
trial_groups <- group_hypergraph(members, node = "state", hyperedge = "group")
event_trials <- data.frame(state = c("Wrong", "Hint", "Question", "Retry"),
                          trials = c(200, 120, 80, 190))
plot(trial_groups, node_sizes = event_trials)

```

plot.net_hg_cluster *Plot method for net_hg_cluster*

Description

Two diagnostic panels. "spectrum": scree plot of the Laplacian spectrum with the k used for clustering marked - the eigengap after k supports (or questions) the choice of k . "embedding": the nodes in the first two spectral-embedding dimensions, labelled, coloured and shaped by cluster, sized by stationary probability - the geometry k -means actually clustered. "both" (default) arranges the two side by side (via gridExtra when available, base grid viewports otherwise).

Usage

```
## S3 method for class 'net_hg_cluster'
plot(x, what = c("both", "spectrum", "embedding"), n_values = NULL, ...)
```

Arguments

x	A net_hg_cluster object.
what	Character. "both" (default), "spectrum", or "embedding".
n_values	Integer. How many smallest eigenvalues to show in the spectrum panel (default: $\min(3 * k, n_nodes)$).
...	Additional arguments (ignored).

Value

For "spectrum"/"embedding", the ggplot object. For "both", the arranged gtable when gridExtra is installed (drawn on the current device), otherwise the two panels are drawn via grid viewports and the list of the two ggplots is returned invisibly.

plot.net_hg_pairwise *Plot the pairwise network of a hypergraph*

Description

Plots the network returned by `pairwise_network()` with `cograph::splot()`: a circle layout, which keeps every label apart however strongly the events are tied, edges whose width grows with the square root of their weight, and nodes whose area grows with their strength, the sum of the weights of their edges.

Usage

```
## S3 method for class 'net_hg_pairwise'
plot(x, ...)
```

Arguments

x	A net_hg_pairwise from <code>pairwise_network()</code> .
...	Arguments passed to <code>cograph::splot()</code> (e.g. layout, seed, minimum, node_fill); they override the defaults set here.

Value

x, invisibly. cograph plots with base graphics.

Examples

```
meetings <- data.frame(
  person = c("Alice", "Bob", "Carol", "Alice", "Bob", "Dave", "Eve"),
  meeting = c("m1", "m1", "m1", "m2", "m2", "m3", "m3"))
meeting_network <- pairwise_network(hypergraph(meetings, node = "person",
                                                hyperedge = "meeting"))

plot(meeting_network)
```

```
plot.net_hg_transduction
```

Plot method for net_hg_transduction

Description

Heatmap of the full node-by-class score matrix: rows are nodes (grouped by predicted class), columns are classes, tile shading and printed values are the spreading scores. Seed nodes (given labels) carry a black tile border, and each node's winning class is marked with a dot, so agreement between seeds, scores, and decisions is visible in one panel. Rows whose winning and runner-up scores are close (small margin) are the assignments to distrust.

Usage

```
## S3 method for class 'net_hg_transduction'
plot(x, ...)
```

Arguments

```
x          A net_hg_transduction object.
...        Additional arguments (ignored).
```

Value

A ggplot object, invisibly printable.

```
plot.net_temporal_hypergraph
```

Plot a temporal hypergraph

Description

Plots the snapshot that `hg_snapshot()` takes at `at` as a hypergraph, with `plot.net_hg()`: the hyperedges active then, as hulls by default or as the incidence matrix with `type = "incidence"`. The snapshot keeps the hyperedge attributes and the data's names for nodes and hyperedges, so `color_by` can name an attribute and the legends use the data's words.

Usage

```
## S3 method for class 'net_temporal_hypergraph'
plot(
  x,
  at = NULL,
  mode = c("active", "cumulative"),
  type = c("hulls", "incidence", "storyline"),
  top = 8L,
  start = NULL,
  end = NULL,
  method = NULL,
  ...
)
```

Arguments

x	A temporal_hypergraph() .
at	Snapshot time; defaults to the end of observation. Not used by the storyline.
mode	Snapshot mode passed to hg_snapshot() . Not used by the storyline.
type	"hulls" (default) or "incidence" plot the snapshot at at (see plot.net_hg()); "storyline" plots the whole history.
top	For type = "storyline": the number of nodes with the most hyperedges to draw as lines (default 8, one Okabe-Ito colour each; ties broken by name), or NULL for every node.
start, end	For type = "storyline": draw only the hyperedges that begin in this period, as dates for a calendar hypergraph or numbers on its clock. NULL (default) leaves the period open.
method	Deprecated. The snapshot used to be projected to a pairwise network; <code>plot(pairwise_network(hg_snapshot(at), type = method))</code> plots that network.
...	For "hulls" and "incidence", arguments passed to plot.net_hg() , such as <code>color_by</code> or <code>labels</code> . For "storyline", <code>edge_labels</code> (TRUE writes the hyperedge and its start under each column), <code>point_size</code> (size of the points, default 2.5) and <code>spacing</code> : "even" (default) puts neighbouring lines one row apart, "strength" brings two neighbouring lines closer the more plotted hyperedges their nodes share: a full row apart for none, 0.35 of a row for the largest number shared by any pair in the plot, and linearly in between, so lines that often meet run together. <code>Spacing</code> never changes the order of the lines. <code>width_by = "degree"</code> draws each line with a width that grows linearly with its node's number of hyperedges in the period, the count top ranks by, and adds a width legend; NULL (default) draws every line alike.

Details

type = "storyline" plots the whole history instead (Tanahashi and Ma 2012). Each of the top nodes with the most hyperedges is a line, from its first hyperedge to its last, and each hyperedge is a column, in order of its start, where a grey bar gathers the lines of its members among the plotted

nodes. The columns are spaced by order, not by elapsed time, and are labelled with the hyperedge and its start. Lines are ordered by the barycentre rule of Sugiyama, Tagawa and Toda (1981): along the columns the members of each hyperedge move to the median of their current rows, and the ordering is refined over repeated sweeps, keeping the one with the fewest line crossings. Each line takes an Okabe-Ito colour and its points a shape, and the legend names the nodes in decreasing order of their number of hyperedges. Eight colours and nine shapes cycle, so up to 72 lines differ in their pair of colour and shape.

Value

A ggplot object.

Conditions

hypergraphs_bad_input when the snapshot has no active nodes, when no hyperedge begins between start and end, for an invalid top, for at, mode or an argument of `plot.net_hg()` with `type = "storyline"`, for top, start or end with another type, and as `plot.net_hg()` raises it; hypergraphs_deprecated (a warning) for method.

References

- Tanahashi, Y., & Ma, K.-L. (2012). Design considerations for optimizing storyline visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 18(12), 2679-2688. doi:10.1109/TVCG.2012.212
- Sugiyama, K., Tagawa, S., & Toda, M. (1981). Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2), 109-125. doi:10.1109/TSMC.1981.4308636

Examples

```
seats <- data.frame(
  case = rep(c("A", "B", "C"), each = 3),
  arbitrator = c("p1", "a1", "a2", "p2", "a1", "a3", "p1", "a4", "a5"),
  constituted = rep(c(1, 2, 4), each = 3),
  concluded = rep(c(4, 3, 6), each = 3)
)
thg <- temporal_hypergraph(seats, node = "arbitrator", hyperedge = "case",
  start = "constituted", end = "concluded")
plot(thg, at = 2.5)
plot(thg, at = 2.5, type = "incidence", edge_labels = TRUE)
plot(thg, type = "storyline")
```

Description

Reuses the trained network, vocabulary, preprocessing and topic keywords. Words absent from the fitted vocabulary are ignored. A document with no remaining words receives missing predictions and a warning, preserving its row and id. The winning softmax probability is a model score, not a calibrated probability of correctness.

Usage

```
## S3 method for class 'hg_hypergat'
predict(
  object,
  newdata = NULL,
  column = NULL,
  id = NULL,
  labels = NULL,
  type = c("predictions", "probabilities"),
  ...
)
```

Arguments

object	A fitted hg_hypergat.
newdata	A character vector or data frame of new documents. NULL returns the original predictions.
column, id	Text and id columns of a data frame. Defaults to the columns used when fitting. An unnamed character vector receives sequential document ids.
labels	Optional observed labels for newdata, in the same formats as hg_hypergat() . Supplying them evaluates the predictions through hg_get.hg_classification() and plot() . These labels never update the model. Unscorable labelled documents count as errors.
type	"predictions" (default), a document table, or "probabilities", a matrix with one column per fitted class.
...	Unused.

Value

A plain data frame (node, label, predicted, score, margin) or a probability matrix, in input order. No fitting occurs.

References

Ding, K., Wang, J., Li, J., Li, D., & Liu, H. (2020). Be more with less: Hypergraph attention networks for inductive text classification. *EMNLP 2020*, 4927-4936.

print.hg_communities *Tables of a hypergraph community ensemble*

Description

Reads, prints and plots the result of `hg_communities()` on a hypergraph.

Usage

```
## S3 method for class 'hg_communities'
print(x, n = 10L, ...)

## S3 method for class 'hg_communities'
hg_get(
  x,
  what = c("medoid", "partitions", "runs", "sizes", "ami", "ari", "nmi", "weights"),
  ...,
  converged = NULL,
  sort_by = NULL,
  top = NULL
)

## S3 method for class 'hg_communities'
plot(x, ...)
```

Arguments

<code>x</code>	An <code>hg_communities</code> object.
<code>n</code>	Number of rows of the default table to print. Default 10.
<code>...</code>	For <code>plot()</code> , additional arguments passed to <code>cograph::splot()</code> ; otherwise unused.
<code>what</code>	Table to return: "medoid" (default: one row per node with its community in the AMI-medoid run), "partitions", "runs", "sizes", "ami", "ari", "nmi", or (IRMM fits only) "weights". The three similarity tables have one row per distinct pair of runs (run_a, run_b, and the similarity), without the diagonal.
<code>converged</code>	For <code>what = "runs"</code> of an IRMM fit: TRUE keeps the runs whose weights settled, FALSE the runs that reached max_iter. NULL (default) keeps every run.
<code>sort_by</code>	For <code>what = "runs"</code> : a numeric column of the runs table ("modularity", "n_communities" or "iterations") to order the runs by, largest first, ties broken by run number. NULL (default) keeps run order.
<code>top</code>	NULL (default, every row) or the number of first rows of any table to return, applied after <code>converged</code> and <code>sort_by</code> .

Value

hg_get(): a base data.frame. Raises hypergraphs_bad_input for converged or sort_by with a table other than "runs", or converged with an Infomap fit. print(): x, invisibly. plot(): the cograph plot of the projection, coloured by the medoid communities.

Examples

```
dat <- data.frame(
  member = c("a", "b", "c", "a", "b", "c", "x", "y", "z", "x", "y", "z"),
  edge = rep(paste0("e", 1:4), each = 3)
)
h <- group_hypergraph(dat, "member", "edge")
if (requireNamespace("igraph", quietly = TRUE)) {
  fit <- hg_communities(h, n_runs = 2, trials = 2, seeds = 1:2)
  hg_get(fit, what = "runs")
  hg_get(fit, what = "runs", sort_by = "n_communities", top = 1)
}
```

```
print.hypergraphs_bootstrap
```

Print method for hypergraphs_bootstrap

Description

Print method for hypergraphs_bootstrap

Usage

```
## S3 method for class 'hypergraphs_bootstrap'
print(x, n = 10L, ...)
```

Arguments

x	A hypergraphs_bootstrap object.
n	Number of rows of the default table to print. Default 10.
...	Additional arguments (ignored).

Value

The input x, invisibly.

```
print.hypergraphs_comparison
```

Print method for hypergraphs_comparison

Description

Print method for hypergraphs_comparison

Usage

```
## S3 method for class 'hypergraphs_comparison'
print(x, n = 10L, ...)
```

Arguments

x	A hypergraphs_comparison object.
n	Number of rows of the default table to print. Default 10.
...	Additional arguments (ignored).

Value

The input x, invisibly.

```
print.hypergraphs_memory_communities
```

Print a memory-network community result

Description

Print a memory-network community result

Usage

```
## S3 method for class 'hypergraphs_memory_communities'
print(x, n = 10L, ...)
```

Arguments

x	A hypergraphs_memory_communities object.
n	Number of rows of the default table to print. Default 10.
...	Ignored.

Value

x, invisibly.

Examples

```
seqs <- list(c("a", "h", "b", "a", "h", "b", "a"),
             c("c", "h", "d", "c", "h", "d", "c"))
print(hg_communities(hon(seqs, max_order = 2L), trials = 2L))
```

```
print.hypergraphs_memory_group
```

Tables of a group model of memory networks

Description

Returns the requested table of every group's memory network, stacked, with a group column first.

Usage

```
## S3 method for class 'hypergraphs_memory_group'
print(x, n = 10L, ...)

## S3 method for class 'hypergraphs_memory_group'
hg_get(x, ...)

## S3 method for class 'hypergraphs_bootstrap_group'
print(x, n = 10L, ...)

## S3 method for class 'hypergraphs_bootstrap_group'
hg_get(x, ...)
```

Arguments

x	A hypergraphs_memory_group from hon() with group.
n	Number of rows of the default table to print. Default 10.
...	Passed to hg_get() on each group's network (what, order_min, sort_by, top, ...).

Value

print() returns x invisibly.

A data.frame: group, then the columns of the requested table.

Examples

```
seqs <- list(c("a", "b", "c", "a", "b"), c("a", "b", "c", "a", "b"),
             c("x", "b", "d", "x", "b"), c("x", "b", "d", "x", "b"))
by_group <- hon(seqs, group = c("g1", "g1", "g2", "g2"), max_order = 2)
by_group
hg_get(by_group, top = 3)
```

print.net_hg_cluster *Print method for net_hg_cluster*

Description

Print method for net_hg_cluster

Usage

```
## S3 method for class 'net_hg_cluster'
print(x, ...)
```

Arguments

x	A net_hg_cluster object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

print.net_hg_measures *Structural measures for a hypergraph*

Description

Computes a comprehensive structural-statistics suite for a [net_hg](#): node-level, hyperedge-level, and global measures. All measures are derived in a few BLAS calls on the incidence matrix.

Usage

```
## S3 method for class 'net_hg_measures'
print(x, ...)

hg_measures(
  hg,
  what = c("nodes", "edges", "overlap", "summary", "distribution", "components"),
  measure = c("hyperdegree", "strength", "n_neighbors", "size")
)
```

Arguments

x	A hg_measures object.
...	Additional arguments (ignored).
hg	A text_hypergraph() (or any hypergraphs net_hg).
what	Which table: "nodes" (default; one row per node with hyperdegree, strength, max_edge_size and n_neighbors, the distinct nodes it shares a hyperedge with), "edges" (one row per hyperedge with its size), "overlap" (one row per hyperedge pair with overlap, overlap_coefficient, jaccard), "summary" (one row per scalar measure), "distribution" (the empirical distribution of measure, as a hypergraphs_distribution table whose plot() draws the CCDF), or "components" (one row per connected component through shared hyperedges, with its n_nodes, n_edges, share of nodes and diameter).
measure	For what = "distribution": "hyperdegree" (default), "strength", "n_neighbors" (node measures) or "size" (hyperedge cardinality).

Details

what = selects which table is returned, as a tidy data.frame.

All measures are computed via standard matrix operations on the binary incidence $B = (b_{ij})$ where $b_{ij} = 1$ iff node i is in hyperedge j :

- hyperdegree = rowSums(B), edge_sizes = colSums(B)
- co_degree = tcrossprod(B) (with zero diagonal)
- edge_pairwise_overlap = crossprod(B) (with zero diagonal)
- overlap_coefficient[i, j] = overlap[i, j] / min(edge_sizes[i], edge_sizes[j])
- jaccard[i, j] = overlap[i, j] / (edge_sizes[i] + edge_sizes[j] - overlap[i, j])

Empty hypergraph (n_hyperedges == 0) returns trivial zeros and empty matrices.

Value

The input x invisibly.

A base data.frame, one row per node, edge, edge pair, measure, distinct value, or component according to what.

References

- Lee, G., Bu, F., Eliassi-Rad, T., & Shin, K. (2025). A survey on hypergraph mining: patterns, tools, and generators. *ACM Computing Surveys*, 57(8), 203. doi:10.1145/3719002
- Do, M. T., Yoon, S., Hooi, B., & Shin, K. (2020). Structural patterns and generative models of real-world hypergraphs. arXiv:2006.07060.

See Also

[network_hypergraph\(\)](#), [group_hypergraph\(\)](#), [pairwise_network\(\)](#).

Examples

```
hg <- text_hypergraph(c(
  a = "salt and soup and onions",
  b = "soup and salt",
  c = "stars and sky"
))
hg_measures(hg)
hg_measures(hg, what = "summary")
hg_measures(hg, what = "components")
```

```
print.net_hg_transduction
```

Print method for net_hg_transduction

Description

Print method for net_hg_transduction

Usage

```
## S3 method for class 'net_hg_transduction'
print(x, ...)
```

Arguments

x	A net_hg_transduction object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

```
print.text_hypergraph Print a text hypergraph
```

Description

Print a text hypergraph

Usage

```
## S3 method for class 'text_hypergraph'
print(x, n = 10L, ...)
```

Arguments

x	A <code>text_hypergraph()</code> object.
n	Number of rows of the default table to print. Default 10.
...	Unused.

Value

x, invisibly.

random_hypergraph	<i>Random hypergraphs</i>
-------------------	---------------------------

Description

Samples a hypergraph from one of four random models, chosen with type.

Usage

```
random_hypergraph(
  type = c("uniform", "regular", "gnp", "sbm"),
  ...,
  seed = NULL
)
```

Arguments

type	The model: "uniform" (default), "regular", "gnp" or "sbm".
...	The arguments of the model, by name:
n	Number of nodes (all types; for "sbm", NULL uses <code>sum(block_sizes)</code>).
m	Number of hyperedges ("uniform", "regular", "gnp"; for "gnp", NULL draws it from a Poisson distribution).
k	Hyperedge size ("uniform") or node degree ("regular").
prob	Sampling weights, length n for "uniform" and length m for "regular"; NULL is equal weights.
p	Incidence probability in $[0, 1]$ ("gnp").
lambda	Poisson mean of m ("gnp").
P	Symmetric block-to-block pair probability matrix ("sbm").
block_sizes	Positive integer block sizes ("sbm").
d	Hyperedge size, recycled across sampled pairs; with <code>variable_size = TRUE</code> , the Poisson mean of the size minus two ("sbm").
impurity	Number of added members replaced by nodes outside the blocks of the pair ("sbm", default 0).
variable_size	Draw sizes as $2 + \text{Poisson}(d)$ ("sbm", default FALSE).
absolute_purity	Replacements come only from outside the blocks of the pair ("sbm", default TRUE).
seed	Optional seed; the random number state of the caller is restored on exit.

Details

"gnp" is the Erdos–Renyi-style model: n nodes and m hyperedges, with every incidence drawn independently as Bernoulli(p). If m is omitted it is Poisson with mean λ , or $n * p$ when λ is also omitted. Empty and singleton hyperedges are retained, since they are outcomes of the model.

"uniform" makes every hyperedge contain exactly k distinct nodes, and "regular" makes every node belong to exactly k distinct hyperedges. Sampling weights `prob` can be unequal.

"sbm" samples a graph stochastic block model and extends every sampled pair of nodes to a hyperedge with further members from the blocks of the pair; `impurity` of the added members can then be replaced by nodes outside those blocks.

Each type takes its own arguments, given by name; an argument that the chosen type does not take is an error.

Value

A `net_hg` with binary incidence and the model parameters in `$params`; for "sbm", `$blocks` records the planted block of each node.

Conditions

`hypergraphs_bad_input` for an argument the chosen type does not take, an argument given without a name, or a count (n , m , k , `impurity`, `seed`) that is not one whole number in range – fractions, NA, Inf and values beyond the integer range are refused, never truncated.

References

Marchette, D. J. (2021). HyperG: Hypergraphs in R. R package version 1.0.0.

Examples

```
uniform <- random_hypergraph("uniform", n = 20, m = 8, k = 3, seed = 1)
hg_measures(uniform, what = "distribution", measure = "size")
regular <- random_hypergraph("regular", n = 20, m = 8, k = 2, seed = 1)
hg_measures(regular, what = "distribution", measure = "hyperdegree")
bernoulli <- random_hypergraph("gnp", n = 20, m = 8, p = 0.2, seed = 1)
bernoulli
blocks <- matrix(c(0.5, 0.05, 0.05, 0.5), 2, 2)
planted <- random_hypergraph("sbm", P = blocks, block_sizes = c(10, 10),
                             d = 3, seed = 1)
hg_get(planted, what = "nodes")
```

Description

The true community of every node a second-order network built from [ring_sequences](#) can hold. A node "u -> v" belongs to the one group holding both u and v; the first-order node of an action that is not shared ("p1_1") belongs to that action's group. First-order nodes of the shared actions ("s1", ...) have no single group and are not listed.

Usage

ring_communities

Format

A data frame with 56 rows (one per node) and 2 columns:

node Character. A node label as [hon\(\)](#) writes it.

community Integer. The planted group, 1 to 4.

Source

Built with [ring_sequences](#) by data-raw/ring_sequences.R.

Examples

ring_communities

ring_sequences	<i>Simulated sequences with planted memory modules</i>
----------------	--

Description

A process whose communities can only be seen with memory. Four groups of four actions sit on a ring and neighbouring groups share one action (s1, ..., s4); the other two actions of group *k* ($p_{k>1}$, $p_{k>2}$) are its own. A walk starts in a random group at a random action of it and each step moves to another action of the current group, chosen at random; on reaching a shared action the walk switches to the neighbouring group with probability 0.05. At a shared action a first-order network sees both groups equally, while a second-order state such as "p1_1 -> s2" knows it came from group 1. The planted group of every state is in [ring_communities](#).

Usage

ring_sequences

Format

A list of 200 character vectors, one walk each, 40 actions long, over the 12 actions s1-s4 and p1_1-p4_2.

Source

Simulated with seed 1 by data-raw/ring_sequences.R.

See Also

[hg_communities\(\)](#), which recovers the four groups from these sequences.

Examples

```
length(ring_sequences)
ring_hon <- hon(ring_sequences, max_order = 2L)
ring_hon
```

simplicial

Simplicial complex from sequences, a network or a memory network

Description

Builds a simplicial complex: the sets of actions that occur together in windows of consecutive actions (type = "window"), every clique of a thresholded weighted network (type = "clique"), every higher-order pathway of a memory network (type = "pathway"), or the Vietoris-Rips complex of a distance matrix (type = "vr"). Every face of every simplex is included.

Usage

```
simplicial(
  x,
  type = "clique",
  threshold = 0,
  max_dim = 10L,
  max_pathways = NULL,
  anomaly = c("all", "over", "under"),
  max_scale = NULL,
  verify = TRUE,
  direction = c("either", "both"),
  window = 3L,
  min_count = 1L,
  validate = FALSE,
  alpha = 0.05,
  null = c("swap", "hypergeometric"),
  n_null = NULL,
  seed = NULL,
  action = NULL,
  actor = NULL,
  time = NULL,
  session = NULL,
  time_threshold = 900,
```

```

    timezone = "UTC",
    ...
)

## S3 method for class 'hypergraphs_simplicial'
plot(
  x,
  y,
  dismantled = FALSE,
  top = NULL,
  type = c("simplices", "summary"),
  ...
)

## S3 method for class 'hypergraphs_simplicial'
hg_get(x, ...)

## S3 method for class 'hypergraphs_simplicial'
print(x, n = 10L, ...)

```

Arguments

<code>x</code>	Sequences for <code>type = "window"</code> , in any form described in sequence-input ; a square weighted matrix, a network object carrying one (<code>netobject</code> , <code>tna</code>) or sequences for <code>type = "clique"</code> ; a memory network (<code>hon()</code> , <code>hupa()</code> on sequences, <code>mogen()</code>) for <code>type = "pathway"</code> ; or for <code>type = "vr"</code> a distance matrix, a <code>dist</code> object, or points: a table or matrix with one row per point and one numeric column per coordinate (not square), or a data frame of numeric coordinates, whose one character column, if present, names the points. Points are turned into Euclidean distances. A square numeric matrix is read as distances.
<code>type</code>	For <code>plot()</code> : <code>"simplices"</code> (default) draws the maximal simplices; <code>"summary"</code> draws the face counts, the Betti numbers and the simplicial degree. <code>dismantled</code> and <code>top</code> apply to <code>"simplices"</code> only.
<code>threshold</code>	For <code>type = "clique"</code> : the smallest absolute edge weight kept (default 0, every non-zero edge).
<code>max_dim</code>	Integer. Highest simplex dimension (a k -simplex has $k + 1$ nodes). Default 10.
<code>max_pathways</code>	For <code>type = "pathway"</code> : keep at most this many pathways, ranked by count (HON) or ratio (HYPA). NULL keeps all.
<code>anomaly</code>	For a HYPA pathway complex: <code>"all"</code> (default), <code>"over"</code> or <code>"under"</code> .
<code>max_scale</code>	For <code>type = "vr"</code> : the longest edge admitted. NULL uses the largest distance.
<code>verify</code>	Logical. Check a clique complex on construction (see Details). Default TRUE; set FALSE to skip the check on very large complexes.
<code>direction</code>	For <code>type = "clique"</code> on a directed network: <code>"either"</code> (default, a pair is joined when the larger of its two weights reaches threshold) or <code>"both"</code> (when both weights do, as in <code>tna::cliques()</code>).
<code>window</code>	For <code>type = "window"</code> : the number of consecutive actions in a window. Default 3.

min_count	For type = "window": the number of windows in which a set of actions must occur to become a simplex. Default 1. Not used with validate = TRUE.
validate	For type = "window": TRUE keeps only the sets of actions that pass the statistical validation described in Details. Default FALSE.
alpha	For validate = TRUE: the false discovery rate. Default 0.05.
null	For validate = TRUE: "swap" (default) or "hypergeometric" (see Details).
n_null	For null = "swap": the number of shuffled copies. The smallest possible p-value is $1 / (n_null + 1)$, and a set can pass on its own only if this is at most alpha divided by the number of possible sets of its size. NULL (default) takes the smallest number that meets this for every size, and at least 999. A number given here that does not meet it raises the warning <code>hypergraphs_low_resolution</code> with the number that would.
seed	For null = "swap": a seed for the shuffles, applied locally so the global random state is restored. NULL (default) uses the current random state.
action, actor, time, session, time_threshold, timezone	For type = "window": long-format arguments (see sequence-input); leave the column names NULL for wide or list input.
...	For type = "pathway" on a network object: arguments passed to the memory-network builder.
y	Unused.
dismantled	For <code>plot()</code> : TRUE draws each maximal simplex in its own panel; FALSE (default) draws them together on one layout.
top	For <code>plot()</code> : the number of maximal simplices to draw, in the order described under Value. NULL (default) draws all of them.
n	Number of rows of the default table to print. Default 10.

Details

type = "window" is a co-occurrence complex (Salnikov et al. 2018) whose unit of co-occurrence is a window of window consecutive actions in a sequence (Ding et al. 2020). The distinct actions of each window form a set; a set observed in at least min_count windows becomes a simplex, together with all its subsets. A window of window actions yields simplices of dimension at most window - 1.

With validate = TRUE a window complex needs no count threshold. A set of actions becomes a simplex when it occurs in more windows than a null model predicts. With null = "swap" (default) the null is swap randomization (Gionis et al. 2007): the windows and their actions are shuffled by checkerboard swaps (Gotelli 2000) so that every window keeps its number of actions and every action its number of windows, and only which actions share a window changes. The p-value of a set is the share of n_null shuffled copies in which the set occurs at least as often as observed, counting the observed data (Phipson & Smyth 2010). With null = "hypergeometric" the null is the statistically validated hypergraph of Musciotto, Battiston & Mantegna (2021): a set of n actions is tested against the windows with exactly n distinct actions, each action falls into these windows independently of the others, and the p-value is exact (their Eq. 3). That null ignores that a window holds exactly n actions. It suits sparse co-occurrence, where most possible sets of actions are rare or absent and each is expected in less than one window, as with many actors who meet in small groups. On dense data, such as a few actions that all occur together often, it expects more co-occurrence

than the windows can hold and validates little; when the median tested set of a size is expected in one window or more it raises the warning `hypergraphs_dense_cooccurrence`. For either null the p-values of the sets of each size are corrected with the Benjamini-Hochberg false discovery rate over all possible sets of that size, and a set is validated when its adjusted p-value is at most `alpha`. Every action remains a vertex. The tests are read with `hg_get(x, what = "validation")`.

`type = "clique"` builds the clique (flag) complex of a network: every clique of the thresholded network becomes a simplex (Giusti, Ghrist & Bassett 2016). A directed network is made undirected by direction: with `"either"` (default) a pair is joined when the larger of its two weights reaches threshold, and with `"both"` when both weights do. `"both"` gives the cliques of `tna::cliques()` with the same threshold (Tikka, Lopez-Pernas & Saqr 2025): the triangles of the complex are its cliques of size three, and so on. Given sequences instead of a network, the clique complex is built on their transition network: the relative transition probabilities that the `tna` family estimates from the same sequences.

A clique complex is checked on construction (`verify = TRUE`): its simplices must be exactly the cliques `igraph` enumerates on the same thresholded graph (when `igraph` is installed), and its Euler characteristic from the face counts must equal the alternating sum of its Betti numbers (the Euler-Poincare identity). A failed check raises the warning `hypergraphs_simplicial_unverified`; the complex is returned either way.

Value

A `simplicial_complex` object. Read it with `hg_get()`: one row per simplex (`what = "simplices"`, the default), the face counts (`"f_vector"`), the per-node simplicial degree (`"degree"`) or, for a validated window complex, one row per tested set of actions (`"validation"`), and the Betti numbers (`"betti"`, with rational coefficients as in `hg_betti()`). `plot()` draws the maximal simplices, those no larger simplex contains, as regions around their nodes (`cograph::plot_simplicial()`; its arguments pass through `...`): the largest simplices first and, among simplices of one size, the largest validation z-score first for a validated window complex, the most frequent first for a window complex, the closest first for a Vietoris-Rips complex, and by node names otherwise. With `type = "summary"`, `plot()` draws the summary of the complex instead: the face counts by dimension, the Betti numbers, the simplicial degree of each node and its degree by dimension. `plot()` returns the figure. For the `plot()` method, `x` is the complex.

References

- Hatcher, A. (2002). *Algebraic Topology*. Cambridge University Press.
- Giusti, C., Ghrist, R., & Bassett, D. S. (2016). Two's company, three (or more) is a simplex. *Journal of Computational Neuroscience*, 41, 1-14. doi:[10.1007/s1082701606086](https://doi.org/10.1007/s1082701606086)
- Tikka, S., Lopez-Pernas, S., & Saqr, M. (2025). `tna`: An R package for transition network analysis. *Applied Psychological Measurement*, 49, 326-328. doi:[10.1177/01466216251348840](https://doi.org/10.1177/01466216251348840)
- Salnikov, V., Cassese, D., Lambiotte, R., & Jones, N. S. (2018). Co-occurrence simplicial complexes in mathematics: identifying the holes of knowledge. *Applied Network Science*, 3, 37. doi:[10.1007/s4110901800743](https://doi.org/10.1007/s4110901800743)
- Ding, K., Wang, J., Li, J., Li, D., & Liu, H. (2020). Be more with less: Hypergraph attention networks for inductive text classification. *Proceedings of EMNLP 2020*, 4927-4936. doi:[10.18653/v1/2020.emnlpmain.399](https://doi.org/10.18653/v1/2020.emnlpmain.399)

- Gionis, A., Mannila, H., Mielikainen, T., & Tsaparas, P. (2007). Assessing data mining results via swap randomization. *ACM Transactions on Knowledge Discovery from Data*, 1(3), 14. doi:[10.1145/1297332.1297338](https://doi.org/10.1145/1297332.1297338)
- Gotelli, N. J. (2000). Null model analysis of species co-occurrence patterns. *Ecology*, 81(9), 2606-2621. doi:[10.1890/00129658\(2000\)081\[2606:NMAOSC\]2.0.CO;2](https://doi.org/10.1890/00129658(2000)081[2606:NMAOSC]2.0.CO;2)
- Phipson, B., & Smyth, G. K. (2010). Permutation p-values should never be zero. *Statistical Applications in Genetics and Molecular Biology*, 9(1), 39. doi:[10.2202/15446115.1585](https://doi.org/10.2202/15446115.1585)
- Musciotto, F., Battiston, F., & Mantegna, R. N. (2021). Detecting informative higher-order interactions in statistically validated hypergraphs. *Communications Physics*, 4, 218. doi:[10.1038/s42005021007104](https://doi.org/10.1038/s42005021007104)
- Battiston, F., Cencetti, G., Iacopini, I., Latora, V., Lucas, M., Patania, A., Young, J.-G., & Petri, G. (2020). Networks beyond pairwise interactions: Structure and dynamics. *Physics Reports*, 874, 1–92. doi:[10.1016/j.physrep.2020.05.004](https://doi.org/10.1016/j.physrep.2020.05.004)

See Also

[hg_betti\(\)](#), [hg_euler\(\)](#), [hg_qanalysis\(\)](#), [hg_homology\(\)](#)

Examples

```
adj <- matrix(c(0, 1, 1, 0,
               1, 0, 1, 1,
               1, 1, 0, 1,
               0, 1, 1, 0), 4, 4, dimnames = list(letters[1:4], letters[1:4]))
sc <- simplicial(adj)
hg_get(sc)
hg_get(sc, what = "f_vector")
```

stop_words_en

English function-word stop list

Description

A fixed list of English function words for the stop_words argument of [text_hypergraph\(\)](#) and [clean_text\(\)](#). type = "minimal" (the default) is a small list of articles, prepositions, conjunctions and auxiliaries, deliberately minimal and versioned with the package. type = "snowball" is the English stop list of the Snowball project (Porter 2001), 174 words that add the personal pronouns, the forms of "do" and the contractions ("i", "you", "don't", "i'm"), which chat messages and other informal text are full of. Corpus-specific boilerplate (e.g. "study", "results" in an abstract corpus) is added by the caller, as in `c(stop_words_en(), "study", "results")`.

Usage

```
stop_words_en(type = c("minimal", "snowball"))
```

Arguments

type "minimal" (default) or "snowball".

Value

A sorted character vector of lowercase English function words. Raises `hypergraphs_bad_input` for an unknown type.

References

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. doi:[10.1017/CBO9780511809071](https://doi.org/10.1017/CBO9780511809071)

Porter, M. F. (2001). Snowball: A language for stemming algorithms. <https://snowballstem.org/texts/introduction.html>

Examples

```
hg <- text_hypergraph(
  c(a = "the salt and the soup", b = "the soup and the stars"),
  stop_words = stop_words_en()
)
hg_get(hg, what = "vocabulary")
length(stop_words_en(type = "snowball"))
```

```
summary.net_hg_cluster
```

Summary method for net_hg_cluster

Description

Summary method for `net_hg_cluster`

Usage

```
## S3 method for class 'net_hg_cluster'
summary(object, ...)
```

Arguments

<code>object</code>	A <code>net_hg_cluster</code> object.
<code>...</code>	Additional arguments (ignored).

Value

A data.frame, one row per cluster: cluster, size, share. Returned **invisibly**: `summary(x)` prints the summary and nothing else; assign the result to keep the table.

```
summary.net_hg_transduction
```

Summary method for net_hg_transduction

Description

Summary method for net_hg_transduction

Usage

```
## S3 method for class 'net_hg_transduction'
summary(object, ...)
```

Arguments

`object` A net_hg_transduction object.
`...` Additional arguments (ignored).

Value

A data.frame, one row per class: class, n_labeled, n_predicted, mean_margin (mean winning margin among the nodes predicted into the class). Returned **invisibly**: summary(x) prints the summary and nothing else; assign the result to keep the table.

```
temporal_hypergraph    Temporal hypergraph from an edge list or co-occurrence data
```

Description

Builds a temporal hypergraph the way a temporal network is defined from data in this ecosystem (DyNet's `dynet()`): one row per relation, with the columns that name its ends and its clock. Two input shapes are accepted. An **edge list** names from and to, and every row is a hyperedge of size two, so an ordinary temporal network is the same object. **Membership data** name a node and a hyperedge: every node sharing one value of hyperedge (a case, a citation block, a seminar) belongs to one hyperedge. Two clocks are understood, and the one you name selects the format:

Usage

```
temporal_hypergraph(
  data,
  from = NULL,
  to = NULL,
  node = NULL,
  hyperedge = NULL,
  time = NULL,
```

```

start = NULL,
end = NULL,
weight = NULL,
nodes = NULL,
time_unit = "auto",
observation_start = NULL,
observation_end = NULL,
sparse = FALSE,
separator = NULL,
cooccur_by = NULL
)

```

Arguments

<code>data</code>	A data frame with one row per relation, or a sequence table (a wide data frame of states, a list of character vectors, or a tna / netobject model); see Details.
<code>from, to</code>	Column names of a pairwise edge list. Detected from the alias table when neither is given and node/hyperedge are not named.
<code>node, hyperedge</code>	Column names of membership data: the node, and the grouping whose shared values bind nodes into one hyperedge. Naming either selects the membership format; the other is then detected by alias if not given. When no column is named and no edge list is detected, both are detected by alias (node, actor, person, ... and hyperedge, group, event, ...), as <code>group_hypergraph()</code> does.
<code>time</code>	Column with the instant of a contact hyperedge.
<code>start, end</code>	Columns with the interval on which a hyperedge is active. <code>start</code> without an <code>end</code> column is read as a contact clock.
<code>weight</code>	Optional membership-weight column.
<code>nodes</code>	Optional node universe: a character vector of node names, or a data frame whose first column holds the names and whose <code>start</code> , <code>time</code> or <code>date</code> column, if present, gives the time each node enters (on the same clock as <code>data</code>). Nodes that never appear in a hyperedge are then kept as zero-degree nodes of every snapshot, and <code>hg_growth()</code> counts a node from its own start. Every observed node must be in the universe.
<code>time_unit</code>	Unit for calendar times: "auto" (default), "seconds", "minutes", "hours", "days" or "weeks". Numeric times are left alone and reported as "step".
<code>observation_start, observation_end</code>	Optional bounds of the observation window, as numbers on the stored clock or as dates for a calendar hypergraph. Either may be omitted; the corresponding limit of the data is then used.
<code>sparse</code>	Store every snapshot's incidence as a sparse Matrix? Default FALSE.
<code>separator</code>	Split the node column on this string, one row per member, before building. Bibliographic exports ship a hyperedge's members as a single delimited cell – EUR-Lex citationcelex and eurovoc, Scopus and Web of Science reference and keyword fields – so <code>separator = ";"</code> replaces the caller's own split, trim and drop-empties. Members empty after trimming are dropped, and a row left with no member contributes no hyperedge.

`cooccur_by` Deprecated name of group; using it warns with a `hypergraphs_deprecated` condition.

Details

interval start and end: the hyperedge is active on the closed interval between them. A missing end means it stays active through the end of observation.

contact time: the hyperedge is an instantaneous event, as in a contact log – a citation, a message, a meeting on one day. It is present at that instant only. The cumulative view in which every hyperedge stays once it has appeared (the point-aggregation model of Coupette et al. 2024) is a snapshot mode, not a property of the data: ask for it with `mode = "cumulative"` in [hg_snapshot\(\)](#), [hg_growth\(\)](#) and [hg_edges\(\)](#).

A third shape is a **sequence table**: one row per session and one column per position holding the state at that step (the wide format of `tna` and `TraMineR`), a list of character vectors, or a `tna / netobject` model built from one. It is recognised when no relational column is named or detected and every column is categorical. Each session is one hyperedge whose members are the states it contains, and a state's membership is a contact at its position, 1 to the session's length, on a "step" clock: the simple co-occurrence reading in which time is order.

In any shape a membership may carry its own time, as when a log has one row per attendance rather than one time per group. The hyperedge then spans from its first to its last membership, each membership is present on its own spell only, and a snapshot keeps the memberships present in its window: `mode = "cumulative"` at step `t` is what each session had shown by `t`, and `window = 3` at `t` is what it showed on steps `t` to `t + 2`. When every membership carries its hyperedge's time, as a tribunal or a citation block does, nothing changes.

Column names are resolved case-insensitively from the same alias table Dynet uses, so `Sender/Receiver`, `source/target`, `onset/terminus` and `timestamp` are understood without being spelled out; a name you give explicitly must exist as written. Times may be numeric, `Date`, `POSIXct` or character date-time strings. Numeric times are kept as they are and reported in "step" units; calendar input is converted to elapsed time since the earliest time in the data, in a unit chosen for the span ("days" beyond three days) or given as `time_unit`, and the unit and origin are stored in the object and shown by `print()`. Every time you pass later – `at`, `start`, `end`, the observation bounds – may be a date for a calendar hypergraph or a number on that clock, and every time column in a result is on that clock.

`observation_start` and `observation_end` declare the study window when it is known independently of the log, with Dynet's meaning: they bound the snapshot times and the measurement grid, and an open-ended hyperedge is active through `observation_end`, but the stored memberships are never rewritten and `hg_get()` returns the original spells. Without them the window is the span of the data.

Every other column that is constant within a hyperedge is kept as a hyperedge attribute in the edge metadata, where `plot()` can colour by it and where [pairwise_network\(\)](#) finds the source a citation block belongs to. Columns that vary within a hyperedge, such as the seat an arbitrator held, are not attributes of the hyperedge and are left out.

Value

A `net_temporal_hypergraph` holding the membership table (node, edge, start, end, weight), the edge metadata (edge, start, end and the hyperedge attributes), the node universe with entry

times, the sorted event times, format ("interval" or "contact"), time_unit, origin and the observation bounds. `hg_get(x, what = "memberships" | "edges" | "nodes")` returns the three tables; `summary()` the one-row description.

References

Coupette, C., Hartung, D., & Katz, D. M. (2024). Legal hypergraphs. *Philosophical Transactions of the Royal Society A*, 382(2270), 20230141. doi:10.1098/rsta.2023.0141

Examples

```
seats <- data.frame(
  case = c("A", "A", "A", "B", "B", "B"),
  arbitrator = c("p1", "a1", "a2", "p2", "a1", "a3"),
  constituted = c(1, 1, 1, 2, 2, 2), concluded = c(4, 4, 4, 5, 5, 5),
  sector = c("oil", "oil", "oil", "gas", "gas", "gas")
)
thg <- temporal_hypergraph(seats, node = "arbitrator", hyperedge = "case",
  start = "constituted", end = "concluded")

thg
hg_snapshot(thg, at = 3)

# a contact log on a calendar: instants at each date, cumulative on request
contacts <- data.frame(from = c("a", "b", "c"), to = c("b", "c", "a"),
  date = as.Date(c("2024-01-01", "2024-01-05", "2024-01-09")))
calls <- temporal_hypergraph(contacts, from = "from", to = "to", time = "date")
calls
hg_snapshot(calls, at = as.Date("2024-01-05"))
hg_snapshot(calls, at = as.Date("2024-01-05"), mode = "cumulative")
```

text_hypergraph

Build a weighted hypergraph from a text corpus

Description

Tokenizes a corpus (base R, deterministic) and builds a weighted hypergraph (a `group_hypergraph()` object with a text layer), in one of three constructions:

Usage

```
text_hypergraph(
  x,
  column = NULL,
  id = NULL,
  construction = c("bag", "window", "sentence", "knn"),
  nodes = c("doc", "word"),
  weight = c("n", "tfidf"),
  stop_words = NULL,
  min_count = 1L,
```

```

    max_words = Inf,
    coverage = 1,
    min_chars = 1L,
    lowercase = TRUE,
    window = 3L,
    window_mode = c("sliding", "tumbling"),
    k = 10L,
    embeddings = NULL,
    model = NULL,
    sparse = NULL,
    separator = NULL
)

```

Arguments

<code>x</code>	A character vector of documents, or a <code>data.frame</code> containing a text column.
<code>column</code>	Name of the text column when <code>x</code> is a <code>data.frame</code> .
<code>id</code>	Optional name of an ID column when <code>x</code> is a <code>data.frame</code> ; its values (unique, non-missing) become the document identifiers. Defaults to the names of <code>x</code> when it is a named character vector, otherwise <code>"doc_1"</code> , <code>"doc_2"</code> , ...
<code>construction</code>	<code>"bag"</code> (default), <code>"window"</code> , <code>"sentence"</code> or <code>"knn"</code> – see Details.
<code>nodes</code>	Which entity is the vertex set for the bag construction: <code>"doc"</code> (default) or <code>"word"</code> . Ignored by <code>"window"</code> (vertices are words) and <code>"knn"</code> (vertices are documents).
<code>weight</code>	Term weighting for the bag construction: <code>"n"</code> (raw count, default) or <code>"tfidf"</code> . The window construction always weights by window counts, the sentence construction by in-sentence counts, the knn construction by cosine similarity.
<code>stop_words</code>	Optional character vector of words to drop after tokenization (compared after lowercasing when <code>lowercase = TRUE</code>); see stop_words_en() . Not applicable to <code>"knn"</code> .
<code>min_count</code>	Minimum total corpus count for a word to be kept (default 1L, keep everything). Not applicable to <code>"knn"</code> .
<code>max_words</code>	Keep at most this many words, the most frequent first (ties broken alphabetically). <code>Inf</code> , the default, keeps every word. Pruning the long tail is the usual way to make a large corpus tractable: the rare words carry little signal but dominate the vocabulary, and the incidence has one hyperedge per word.
<code>coverage</code>	Keep the fewest most-frequent words whose combined corpus count reaches this share of all tokens, a number in $(0, 1]$. 1, the default, keeps every word; 0.99 keeps the words carrying 99% of the tokens. Applied together with <code>max_words</code> and <code>min_count</code> , the strictest wins. Documents left with no tokens are dropped with a <code>hypergraphs_dropped_documents</code> warning.
<code>min_chars</code>	Minimum number of characters for a word to be kept (default 1L, keep everything). Raise it to drop the single letters and short fragments that initials, enumerations and hyphenated line breaks leave behind. Not applicable to <code>"knn"</code> .
<code>lowercase</code>	Lowercase the text before tokenization (default TRUE).
<code>window</code>	Window size in tokens for construction = <code>"window"</code> (default 3L).

window_mode	"sliding" (default) or "tumbling", for construction = "window".
k	Number of nearest neighbors per hyperedge for construction = "knn" (default 10L).
embeddings	Numeric matrix of document embeddings for construction = "knn": one row per document, either rownames matching the document IDs or rows in document order. NULL (default) encodes the text with sbert.
model	Passed to <code>sbert::encode()</code> when embeddings are computed (NULL = sbert's default model).
sparse	Store the incidence as a <code>Matrix::dgCMatrix</code> (bag and sentence constructions). Default NULL chooses: sparse when the document-by-word incidence would have a million cells or more, dense otherwise. Sparse hypergraphs scale to tens of thousands of documents; <code>hg_cluster()</code> , <code>hg_classify()</code> , <code>hg_pagerank()</code> , and <code>hg_measures()</code> use sparse operator paths that agree with the dense engines (tested), while tensor centralities and the null test currently require the dense representation.
separator	NULL (default) splits the text into words. A string such as ";" reads the column as a delimited field of terms instead, such as the author keywords of a bibliographic export: each term is the whole phrase between separators ("higher education"), trimmed, and becomes one hyperedge, so a paper is bound to the keywords its authors gave it. This is the keyword incidence of co-word analysis (Callon et al. 1983) read as a hypergraph. <code>stop_words</code> then names whole terms to drop and <code>min_count</code> the fewest occurrences a term needs. Bag construction only.

Details

- construction = "bag" (default): the document-word incidence. With nodes = "doc", documents are vertices and each word is a hyperedge over the documents containing it (the orientation for document clustering and transductive classification, Hayashi et al. 2020); with nodes = "word" the orientation reverses (the HyperGAT sentence-as-hyperedge setup generalized to whole documents).
- construction = "window": words are vertices and every token window is a hyperedge – the sliding-window sequential construction of Ding et al. (2020, Sec. 3.2), extended with weights: each hyperedge is a distinct window content (the sorted set of words co-occurring in a window) and its incidence weight is the number of windows with that content. Windows never cross document boundaries. `window_mode = "sliding"` moves one token at a time (a document shorter than window forms one whole-document window); `"tumbling"` uses consecutive chunks, trailing partial chunk included. This is the text front end of `window_hypergraph()`: on token sequences with no repeated word inside a window, the two constructions produce the same incidence matrix (tested), with the hyperedges here named by their word content. They differ only at the edges of the definition – a document shorter than window, a trailing partial chunk, or a word repeated inside one window, which a set-valued hyperedge collapses.
- construction = "sentence": words are vertices and every sentence is a hyperedge – HyperGAT's sentence-as-hyperedge construction (Ding et al. 2020, Sec. 3.2) over the whole corpus. Documents are split at ., !, ? and ;; each sentence with at least one surviving token is one hyperedge named `<doc>#<sentence>`, and its incidence weight is the word's occurrence count in that sentence. Two words are then related when they share a sentence, not merely a document – the scope `hg_keywords()` uses for `type = "sentence centrality"`.

- `construction = "knn"`: documents are vertices and each document plus its `k` nearest neighbors in an embedding space is one hyperedge, weighted by cosine similarity (see [knn_hypergraph\(\)](#)). Pass a precomputed embeddings matrix, or leave it `NULL` to encode the text with the `sbert` package (if installed; models download only on explicit user confirmation, per `sbert`'s policy).

Tokens are maximal runs of letters (with internal apostrophes); digits and punctuation are separators. `weight = "tfidf"` (bag construction only) uses the smoothed formula $tf * (\log((1 + N) / (1 + df)) + 1)$ with `tf` the raw count, `N` the number of documents, and `df` the word's document frequency – the `smooth_idf` variant of Manning, Raghavan and Schütze (2008) as popularized by `scikit-learn`, which is never zero and so never silently disconnects a vertex. `stop_words` and `min_count` filtering happen before windowing, so removing a token closes the gap it leaves in the sequence.

Value

An object of class `c("text_hypergraph", "net_hg")` – a [group_hypergraph\(\)](#) hypergraph accepted by every hypergraphs hypergraph verb and by [hg_measures\(\)](#), [hg_centralty\(\)](#), [hg_cluster\(\)](#), and [hg_classify\(\)](#) – with a `text` field recording the corpus tables. Use [hg_get.text_hypergraph\(\)](#) for the tidy weight table, and its `what` argument for the document and vocabulary tables.

Conditions

Raises `hypergraphs_bad_input` (broken argument contract, including bag-only arguments passed to other constructions, and a `data.frame` column other than `column` and `id` named like a column of the document table: `doc`, and for the token-based constructions also `n_tokens` or `n_types`), `hypergraphs_empty_corpus` (no document survives tokenization and filtering), `hypergraphs_missing_embeddings` (`construction = "knn"` with neither embeddings nor the `sbert` package), and warns with `hypergraphs_dropped_document` when some documents end up empty.

References

- Callon, M., Courtial, J.-P., Turner, W. A., & Bauin, S. (1983). From translations to problematic networks: An introduction to co-word analysis. *Social Science Information*, 22(2), 191-235. [doi:10.1177/053901883022002003](#)
- Ding, K., Wang, J., Li, J., Li, D., & Liu, H. (2020). Be more with less: Hypergraph attention networks for inductive text classification. *EMNLP 2020*. [doi:10.18653/v1/2020.emnlpmain.399](#)
- Hayashi, K., Aksoy, S. G., Park, C. H., & Park, H. (2020). Hypergraph random walks, Laplacians, and clustering. *CIKM 2020*. [doi:10.1145/3340531.3412034](#)
- Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Examples

```
corpus <- c(
  cooking_1 = "Simmer the soup with onions and carrots",
  cooking_2 = "This soup recipe needs a pinch of salt",
  space_1   = "The telescope revealed a distant galaxy",
  space_2   = "Astronomers aimed the telescope at the night sky"
)
hg <- text_hypergraph(corpus, weight = "tfidf")
```

```
hg
hg_get(hg)

win <- text_hypergraph(corpus, construction = "window", window = 3)
win

# knn from precomputed embeddings (sbert-free, offline)
emb <- matrix(c(1, 0, 0.9, 0.1, 0, 1, 0.1, 0.9),
              nrow = 4, byrow = TRUE,
              dimnames = list(names(corpus), NULL))
text_hypergraph(corpus, construction = "knn", k = 1, embeddings = emb)
```

topic_network	<i>Network of topics</i>
---------------	--------------------------

Description

Builds the topic-by-topic network of a document hypergraph, one node per topic and one undirected weighted edge per pair of related topics. The topics come either from a partition of the documents or from a mixed-membership topic model.

Usage

```
topic_network(
  hg,
  clusters = NULL,
  topics = NULL,
  threshold = NULL,
  cutoff = 0.01,
  similarity = c("none", "association", "cosine", "jaccard", "inclusion", "equivalence"),
  what = c("edges", "network")
)
```

Arguments

hg	The document hypergraph the topics were computed on.
clusters	A partition: the tidy table returned by hg_cluster() (columns node, cluster), a named vector of cluster labels, or the name of a column of the hypergraph's document table.
topics	A mixed-membership topic model of hg fitted by hg_topics() . Give clusters or topics.
threshold	Topic model only: the share at which a topic counts as present in a document, one number in (0, 1]. NULL (default) relates topics by the correlation of their shares instead.
cutoff	Topic model without threshold only: the correlation a pair must exceed to be kept (default 0.01, as in <code>stm::topicCorr()</code>).

similarity	"none" (default, the raw count), "association", "cosine", "jaccard", "inclusion" or "equivalence". Count networks only.
what	"edges" (default) for the edge list, "network" for a <code>cograph_network</code> built from it with <code>cograph::as_cograph()</code> , whose node table carries each topic's size.

Details

For a partition (clusters), the network is the co-occurrence network of the bibliometric kind: the strength between two topics is the sum, over all words, of the product of the number of documents in each topic that contain the word (full counting, the aggregation bibnets uses for keyword co-occurrence).

For a topic model fitted by `hg_topics()` (topics), two topics are related when the same documents draw on both. Without threshold, the weight is the correlation of the two topics' shares over the documents, and the pairs with a correlation above cutoff are kept, the "simple" topic correlation of `stm::topicCorr()` (Roberts et al. 2019). With threshold, a topic counts as present in a document when its share is at least threshold, and the weight of a pair is the number of documents in which both are present (Abuhay et al. 2017; Cassi et al. 2017).

A count network, from a partition or from a thresholded topic model, can be normalised by the similarity measures of bibnets' `normalize()`, with each topic's total on the diagonal (its documents with the word, or its documents with the topic present). With D_i the diagonal entry of topic i and A_{ij} the raw count: "association" is $A_{ij} / (D_i D_j)$, "cosine" is $A_{ij} / \sqrt{D_i D_j}$, "jaccard" is $A_{ij} / (D_i + D_j - A_{ij})$, "inclusion" is $A_{ij} / \min(D_i, D_j)$ and "equivalence" is $A_{ij}^2 / (D_i D_j)$ (van Eck & Waltman 2009).

The pairwise network of topics is the projection of the topic combinations that `group_hypergraph()` builds from the same topic model, in which every document binds all the topics it contains at once.

Value

For `what = "edges"`: a base data.frame, one row per pair of topics with a positive weight, columns source, target, weight, pairs in the topics' natural order. For `what = "network"`: a `cograph_network` with one node per topic (label, name, size), a topic without any edge included, and one undirected weighted edge per pair; size is the number of documents of a cluster, the documents in which a topic is present, or (without threshold) a topic's expected number of documents. Raises `hypergraphs_bad_input` for unknown node names, for neither or both of clusters and topics, for a topic model not fitted on hg, for an invalid threshold or cutoff, and for a similarity measure on a correlation network. A topic whose share is the same in every document has no correlation with any other: it is kept without edges, with a `hypergraphs_constant_topics` warning.

References

- van Eck, N. J., & Waltman, L. (2009). How to normalize cooccurrence data? An analysis of some well-known similarity measures. *Journal of the American Society for Information Science and Technology*, 60(8), 1635–1651.
- Roberts, M. E., Stewart, B. M., & Tingley, D. (2019). stm: An R package for structural topic models. *Journal of Statistical Software*, 91(2), 1-40. doi:10.18637/jss.v091.i02

Abuhay, T. M., Kovalchuk, S. V., Bochenina, K., Kamps, G., Krzhizhanovskaya, V. V., & Lees, M. H. (2017). Analysis of computational science papers from ICCS 2001-2016 using topic modeling and graph theory. *Procedia Computer Science*, 108, 7-17. doi:10.1016/j.procs.2017.05.183

Cassi, L., Lahatte, A., Rafols, I., Sautier, P., & de Turckheim, E. (2017). Improving fitness: Mapping research priorities against societal needs on obesity. *Journal of Informetrics*, 11(4), 1095-1113. doi:10.1016/j.joi.2017.09.010

Examples

```
hg <- text_hypergraph(c(
  cooking_1 = "simmer the soup with onions and carrots",
  cooking_2 = "this soup recipe needs salt on a cold night",
  space_1 = "the telescope revealed a distant galaxy and stars",
  space_2 = "astronomers aimed the telescope at the stars all night"
), stop_words = c("the", "with", "and", "a", "this", "at", "on", "all"))
topics <- hg_cluster(hg, k = 2, seed = 1)
topic_network(hg, clusters = topics)
topic_network(hg, clusters = topics, similarity = "cosine")

corpus <- c(
  a = "soup salt onion soup broth", b = "salt soup broth onion",
  c = "stars sky moon night", d = "sky stars night moon moon",
  e = "soup stars salt sky night broth", f = "onion salt stars broth")
corpus_hg <- text_hypergraph(corpus)
model <- hg_topics(corpus_hg, k = 2, nstart = 2)
topic_network(corpus_hg, topics = model, threshold = 0.2)
```

tutoring_events

Problem steps of learners with a tutor

Description

The events of 13,309 problem steps that learners worked through with an AI tutor, one row per event. A step begins with the task, the learner attempts an answer, and the answer is correct or incorrect. After an incorrect answer the tutor responds with guidance, a question, an order, a refutation, a prompt to reflect or comfort, and the learner reattempts or gives up. A step ends Completed when an answer is correct, or Stopped when it ends without a correct answer. A trial is the part of a step between two answers: a new trial starts at the event after each Incorrect, so the tutor's response to an incorrect answer belongs to the reattempt it leads to. The events are real; each is renamed to a word of the same meaning, similar events are merged, and learner and skill identifiers are removed. Rebuilt by data-raw/tutoring_events.R.

Usage

```
tutoring_events
```

Format

A data frame with 84,356 rows and 5 columns:

step Problem step number (integer).

trial Trial identifier, unique over the data ("**<step>.<n>**").

position Position of the event within its step (integer).

event The event: the step's Begin and Task, the learner's Attempt, Reattempt and GiveUp, the answers Correct and Incorrect, the step's end Completed or Stopped, and the tutor's Guidance, Question, Order, Refute, Reflect and Comfort.

outcome "completed" for a step that ends with a correct answer, "stopped" for one that ends without.

Source

Renamed by data-raw/tutoring_events.R.

Examples

```
step_sets <- group_hypergraph(tutoring_events, node = "event",
                              hyperedge = "trial", group = "outcome", top = 4)
step_sets
```

window_hypergraph	<i>Windowed Sequence Hyperedges</i>
-------------------	-------------------------------------

Description

Builds a hypergraph from categorical sequence data: a fixed-size window moves over each sequence and the distinct states observed inside one window form one hyperedge (Ding et al. 2020). Windows whose distinct state sets coincide collapse into a single hyperedge whose weight is the number of such windows (`window_counts`); the incidence cells hold the total within-window occurrences of each state, so the incidence matrix carries edge-dependent vertex weights (Chitra & Raphael 2019) that `hg_cluster()` with `type = "random_walk"` uses directly. The window counts are the default hyperedge weights of the whole Laplacian family (`hg_laplacian()`, `hg_cluster()`, `hg_classify()`).

Usage

```
window_hypergraph(
  data,
  window = 3L,
  step = 1L,
  action = NULL,
  actor = NULL,
  time = NULL,
  session = NULL,
```

```

    time_threshold = 900,
    timezone = "UTC",
    min_size = 1L,
    min_weight = 1L,
    collapse = TRUE
  )

```

Arguments

data	Sequences in any form described in sequence-input : a long event table, a wide data.frame or character matrix (one sequence per row, trailing NAs stripped), or a list of character vectors. A list's names, when it has them, identify its sequences and must be unique and non-empty (hypergraphs_bad_input otherwise).
window	Integer ≥ 2 . Window size in sequence positions.
step	Integer ≥ 1 . Offset between consecutive window starts: 1 slides, window tumbles.
action, actor, time, session, time_threshold, timezone	Long-format arguments, as in sequence-input .
min_size	Integer ≥ 1 . Drop hyperedges with fewer distinct states after collapsing. The default 1 keeps everything.
min_weight	Integer ≥ 1 . Drop hyperedges observed in fewer than min_weight windows, keeping only recurrent state combinations (the role min_freq plays in rule extraction). The default 1 keeps everything. The total dropped by min_size and min_weight together is recorded in params\$n_dropped; with both at their defaults, sum(window_counts) equals the number of non-empty full windows.
collapse	TRUE (default) merges windows with the same set of states into one hyperedge weighted by its number of windows. FALSE keeps every window as a hyperedge of its own, in sequence order and in order of position, named by its positions ("1-3", or "sequence_2:1-3" when there are several sequences); the edge metadata (edge_data) then records each window's sequence, start and end position, every window count is 1, and min_weight does not apply. The ordered windows of one sequence are the input of plot(type = "storyline").

Details

step = 1 (default) gives sliding windows; step = window gives tumbling (non-overlapping) windows. Only full windows are formed: a sequence shorter than window contributes none and is counted in params\$n_short_sequences. NA states are excluded from a window's set; a window containing only NAs is skipped and counted in params\$n_empty_windows.

Whole-sequence hyperedges (each complete sequence as one hyperedge) are the special case already covered by [group_hypergraph\(\)](#) on long-format data with member = action and group = actor; use this verb when the hyperedges should be local in time.

Value

A net_hg object (as from [network_hypergraph\(\)](#) and [group_hypergraph\(\)](#)): a list with hyperedges (list of sorted node index vectors), incidence (numeric node x hyperedge matrix of within-window

occurrence totals), nodes, n_nodes, n_hyperedges, window_counts (integer, one weight per hyperedge: the number of windows collapsed into it), size_distribution, and params (source = "window_hypergraph", window, step, min_size, n_sequences, n_short_sequences, n_windows, n_empty_windows, n_dropped). Use `hg_get()` for the tidy one-row-per-hyperedge table.

References

Ding, K., Wang, J., Li, J., Li, D., & Liu, H. (2020). Be more with less: Hypergraph attention networks for inductive text classification. *Proceedings of EMNLP 2020*, 4927-4936. doi:10.18653/v1/2020.emnlpmain.399

Chitra, U., & Raphael, B. J. (2019). Random walks on hypergraphs with edge-dependent vertex weights. *Proceedings of the 36th International Conference on Machine Learning*, PMLR 97, 1172-1181.

See Also

`network_hypergraph()`, `group_hypergraph()`, `hg_measures()`, `hg_cluster()`, `pairwise_network()`

Examples

```
hg <- window_hypergraph(human_long, action = "code",
                        actor = "session_id", time = "timestamp",
                        window = 3L)

hg
edges <- hg_get(hg)
head(edges)

# Tumbling windows over wide-format sequences
wide <- data.frame(
  t1 = c("plan", "code"), t2 = c("code", "test"),
  t3 = c("test", "code"), t4 = c("plan", "debug")
)
window_hypergraph(wide, window = 2L, step = 2L)
```

Index

* datasets

- covid_abstracts, 8
 - covid_embeddings, 9
 - covid_sample, 9
 - forum_posts, 11
 - gfcc_citations, 12
 - gfcc_decisions, 13
 - icsid_tribunals, 152
 - long-data, 154
 - ring_communities, 187
 - ring_sequences, 188
 - tutoring_events, 204
- ai_long (long-data), 154
- clean_text, 6
- clean_text(), 193
- cograph::as_cograph(), 203
- cograph::overlay_communities(), 165
- cograph::splot(), 165, 174, 179
- covid_abstracts, 8, 9
- covid_embeddings, 9
- covid_sample, 9
- dual_hypergraph, 10
- dual_hypergraph(), 103
- forum_posts, 11
- gfcc_citations, 12, 13
- gfcc_decisions, 12, 13
- ggplot2::scale_size_area(), 169
- group_hypergraph, 14
- group_hypergraph(), 36, 71, 101, 107, 129, 139, 150, 151, 153, 155, 170, 171, 184, 196, 198, 201, 203, 206, 207
- heterogeneous_hgat, 18
- hg_agreement, 19
- hg_agreement(), 170
- hg_allset, 21
- hg_assortativity, 22
- hg_assortativity(), 51, 141
- hg_betti, 24
- hg_betti(), 57, 192, 193
- hg_bootstrap, 25
- hg_bootstrap(), 34, 47, 48, 59, 64, 70, 121, 123, 143, 144
- hg_bottleneck, 27
- hg_bottleneck(), 91
- hg_centrality, 28
- hg_centrality(), 29, 32, 98, 99, 144, 153, 201
- hg_centrality.net_hg, 29
- hg_centrality.net_hg(), 28
- hg_centrality.net_hon, 32
- hg_centrality.net_hon(), 28
- hg_classification, 35, 94
- hg_classify, 34
- hg_classify(), 19, 52, 86, 121, 129, 153, 170, 200, 201, 205
- hg_cluster, 36
- hg_cluster(), 19, 30, 39, 56, 97, 98, 105, 106, 118, 121–123, 126, 127, 130, 131, 135, 138, 153, 170, 200–202, 205, 207
- hg_cocluster, 38
- hg_communities, 40
- hg_communities(), 20, 41, 46, 48, 59, 70, 110, 111, 144, 165, 170, 179, 189
- hg_communities.net_hg, 41
- hg_communities.net_hg(), 40
- hg_communities.net_hon, 43
- hg_communities.net_hon(), 40
- hg_community_quality, 45
- hg_community_quality(), 49
- hg_compare, 46
- hg_compare(), 26, 59, 65, 70, 121, 143, 144
- hg_compare_communities, 48
- hg_compare_communities(), 70

- hg_degree, 50
- hg_degree(), 85
- hg_degree_correlation, 51
- hg_degree_correlation(), 24
- hg_dictionary, 52
- hg_edge Centrality, 53
- hg_edge Centrality(), 55, 129
- hg_edges, 54
- hg_edges(), 129, 197
- hg_embed, 56
- hg_euler, 57
- hg_euler(), 25, 193
- hg_get, 58
- hg_get(), 26, 35, 40, 45, 47, 68–70, 91, 94, 101, 104, 119, 138, 143, 144, 147, 156–158, 171, 182, 192, 207
- hg_get.hg_classification, 60
- hg_get.hg_classification(), 63, 178
- hg_get.hg_communities
 - (print.hg_communities), 179
- hg_get.hg_communities(), 59
- hg_get.hg_hypergat, 62
- hg_get.hg_hypergat(), 59, 94
- hg_get.hypergraphs_bootstrap, 63
- hg_get.hypergraphs_bootstrap_group
 - (print.hypergraphs_memory_group), 182
- hg_get.hypergraphs_community_comparison
 - (hg_compare_communities), 48
- hg_get.hypergraphs_comparison, 64
- hg_get.hypergraphs_hypa, 65
- hg_get.hypergraphs_memory_communities, 66
- hg_get.hypergraphs_memory_communities(), 45, 165
- hg_get.hypergraphs_memory_group
 - (print.hypergraphs_memory_group), 182
- hg_get.hypergraphs_motifs (hg_motifs), 112
- hg_get.hypergraphs_result, 68
- hg_get.hypergraphs_simplicial
 - (simplicial), 189
- hg_get.hypergraphs_summary, 69
- hg_get.net_hg, 70
- hg_get.net_hg(), 59
- hg_get.net_hg_cluster, 72
- hg_get.net_hg_measures, 72
- hg_get.net_hg_mmsbm (hg_mmsbm), 106
- hg_get.net_hg_topics (hg_topics), 136
- hg_get.net_hg_transduction, 73
- hg_get.net_hon, 74
- hg_get.net_honem, 75
- hg_get.net_hypa
 - (hg_get.hypergraphs_hypa), 65
- hg_get.net_markov_order, 76
- hg_get.net_markov_order(), 77
- hg_get.net_markov_order_group, 77
- hg_get.net_markov_stability, 77
- hg_get.net_mogen, 79
- hg_get.net_path_dependence, 80
- hg_get.net_temporal_hypergraph, 81
- hg_get.persistence_landscape, 82
- hg_get.persistent_homology, 83
- hg_get.q_analysis, 84
- hg_get.simplicial_complex, 84
- hg_get.text_hypergraph, 86
- hg_get.text_hypergraph(), 59, 201
- hg_growth, 87
- hg_growth(), 196, 197
- hg_hnbn, 89
- hg_homology, 90
- hg_homology(), 24, 25, 27, 59, 68, 70, 83, 100, 101, 142, 193
- hg_hypergat, 91
- hg_hypergat(), 19, 52, 121, 178
- hg_hypergc, 95
- hg_joint_cluster, 96
- hg_keywords, 97
- hg_keywords(), 129–131, 200
- hg_landscape, 100
- hg_landscape(), 59, 68, 70, 82, 91
- hg_laplacian, 101
- hg_laplacian(), 35, 36, 118, 205
- hg_line_graph, 102
- hg_line_graph(), 56, 163
- hg_markov_stability, 104
- hg_markov_stability(), 59, 68, 70, 78
- hg_measures (print.net_hg_measures), 183
- hg_measures(), 31, 56, 115, 141, 153, 200, 201, 207
- hg_membership, 105
- hg_membership(), 110
- hg_mmsbm, 106
- hg_mmsbm(), 20, 70, 170
- hg_modularity, 110

- hg_modularity(), [42, 43](#)
- hg_motifs, [112](#)
- hg_motifs(), [70, 129](#)
- hg_neural, [113](#)
- hg_neural(), [19, 22, 90, 96, 121](#)
- hg_null_test, [115](#)
- hg_null_test(), [148, 150](#)
- hg_pagerank, [117](#)
- hg_pagerank(), [200](#)
- hg_qanalysis, [119](#)
- hg_qanalysis(), [59, 68, 70, 84, 193](#)
- hg_representations, [120](#)
- hg_seeds, [121](#)
- hg_sequences, [121](#)
- hg_sequences(), [11](#)
- hg_snapshot, [124](#)
- hg_snapshot(), [129, 175, 176, 197](#)
- hg_snapshots, [125](#)
- hg_snapshots(), [54, 55, 113](#)
- hg_stability, [126](#)
- hg_subset, [128](#)
- hg_subset(), [17, 167, 169](#)
- hg_topic_quality, [130](#)
- hg_topic_quality(), [133, 134, 138](#)
- hg_topic_search, [133](#)
- hg_topic_sizes, [135](#)
- hg_topics, [136](#)
- hg_topics(), [16, 20, 109, 122, 131, 133, 134, 151, 170, 202, 203](#)
- hg_transitivity, [139](#)
- hg_transitivity(), [24](#)
- hg_wasserstein, [141](#)
- hg_wasserstein(), [27, 91](#)
- hon, [142](#)
- hon(), [25, 26, 28, 32, 34, 40, 44, 47, 48, 59, 68, 70, 74, 121, 123, 144, 145, 182, 188, 190](#)
- honem, [144](#)
- honem(), [59, 68, 70, 75, 144](#)
- human_long (long-data), [154](#)
- hypa, [145](#)
- hypa(), [59, 65, 66, 70, 121, 146, 148, 190](#)
- hypa.default, [146](#)
- hypa.default(), [145, 148, 150](#)
- hypa.net_hg, [148](#)
- hypa.net_hg(), [145, 147](#)
- hypergraph, [150](#)
- hypergraph(), [162, 163](#)
- icsid_tribunals, [152](#)
- knn_hypergraph, [153](#)
- knn_hypergraph(), [10, 55, 103, 115, 117, 201](#)
- long-data, [154](#)
- markov_order, [155](#)
- markov_order(), [26, 48, 59, 68, 70, 76, 77, 105, 121, 157, 159](#)
- memory, [156](#)
- memory(), [34, 59, 68, 70, 80, 156, 159](#)
- mogen, [158](#)
- mogen(), [59, 68, 70, 79, 121, 156, 157, 190](#)
- net_hg, [14, 29, 139, 183](#)
- network_hypergraph, [159](#)
- network_hypergraph(), [16, 17, 31, 101, 139, 151, 184, 206, 207](#)
- pairwise_network, [161](#)
- pairwise_network(), [17, 30, 31, 41, 46, 103, 120, 141, 174, 184, 197, 207](#)
- plot.hg_classification
(hg_get.hg_classification), [60](#)
- plot.hg_communities
(print.hg_communities), [179](#)
- plot.hg_hypergat (hg_get.hg_hypergat), [62](#)
- plot.hypergraphs_bootstrap, [163](#)
- plot.hypergraphs_community_comparison
(hg_compare_communities), [48](#)
- plot.hypergraphs_comparison, [164](#)
- plot.hypergraphs_keywords
(hg_keywords), [97](#)
- plot.hypergraphs_membership
(hg_membership), [105](#)
- plot.hypergraphs_memory_communities, [165](#)
- plot.hypergraphs_motifs (hg_motifs), [112](#)
- plot.hypergraphs_result
(hg_get.hypergraphs_result), [68](#)
- plot.hypergraphs_series (hg_growth), [87](#)
- plot.hypergraphs_simplicial
(simplicial), [189](#)
- plot.hypergraphs_topic_quality
(hg_topic_quality), [130](#)
- plot.hypergraphs_topic_search
(hg_topic_search), [133](#)

`plot.hypergraphs_topic_sizes`
 (`hg_topic_sizes`), 135
`plot.net_hg`, 166
`plot.net_hg()`, 165, 166, 175–177
`plot.net_hg_cluster`, 173
`plot.net_hg_mmsbm` (`hg_mmsbm`), 106
`plot.net_hg_pairwise`, 174
`plot.net_hg_topics` (`hg_topics`), 136
`plot.net_hg_transduction`, 175
`plot.net_temporal_hypergraph`, 175
`plot.net_temporal_hypergraph()`, 171
`predict.hg_hypergat`, 177
`predict.hg_hypergat()`, 94
`print.hg_classification`
 (`hg_get.hg_classification`), 60
`print.hg_communities`, 179
`print.hg_hypergat` (`hg_get.hg_hypergat`),
 62
`print.hypergraphs_bootstrap`, 180
`print.hypergraphs_bootstrap_group`
 (`print.hypergraphs_memory_group`),
 182
`print.hypergraphs_community_comparison`
 (`hg_compare_communities`), 48
`print.hypergraphs_comparison`, 181
`print.hypergraphs_hypa` (`hypa.default`),
 146
`print.hypergraphs_keywords`
 (`hg_keywords`), 97
`print.hypergraphs_memory_communities`,
 181
`print.hypergraphs_memory_group`, 182
`print.hypergraphs_result`
 (`hg_get.hypergraphs_result`), 68
`print.hypergraphs_simplicial`
 (`simplicial`), 189
`print.hypergraphs_summary`
 (`hg_get.hypergraphs_summary`),
 69
`print.net_hg` (`network_hypergraph`), 159
`print.net_hg_cluster`, 183
`print.net_hg_measures`, 183
`print.net_hg_mmsbm` (`hg_mmsbm`), 106
`print.net_hg_topics` (`hg_topics`), 136
`print.net_hg_transduction`, 185
`print.text_hypergraph`, 185

`random_hypergraph`, 186
`random_hypergraph()`, 71, 101

`result-summary`
 (`hg_get.hypergraphs_summary`),
 69
`ring_communities`, 187, 188
`ring_sequences`, 188, 188

`sequence-input`, 26, 104, 143, 145–147,
 156–158, 190, 191, 206
`simplicial`, 189
`simplicial()`, 24, 50, 57, 59, 66, 70, 74, 79,
 85, 90, 119, 155, 160, 161
`stats::p.adjust`, 147–149
`stats::p.adjust()`, 149
`stop_words_en`, 193
`stop_words_en()`, 93, 199
`summary.hg_classification`
 (`hg_get.hg_classification`), 60
`summary.hg_communities`
 (`hg_get.hypergraphs_summary`),
 69
`summary.hg_hypergat`
 (`hg_get.hg_hypergat`), 62
`summary.hypergraphs_bootstrap`
 (`hg_get.hypergraphs_summary`),
 69
`summary.hypergraphs_bootstrap_group`
 (`hg_get.hypergraphs_summary`),
 69
`summary.hypergraphs_community_comparison`
 (`hg_get.hypergraphs_summary`),
 69
`summary.hypergraphs_comparison`
 (`hg_get.hypergraphs_summary`),
 69
`summary.hypergraphs_hypa`
 (`hg_get.hypergraphs_summary`),
 69
`summary.hypergraphs_memory_communities`
 (`hg_get.hypergraphs_summary`),
 69
`summary.hypergraphs_memory_group`
 (`hg_get.hypergraphs_summary`),
 69
`summary.hypergraphs_motifs`
 (`hg_get.hypergraphs_summary`),
 69
`summary.hypergraphs_result`
 (`hg_get.hypergraphs_summary`),
 69

`summary.hypergraphs_simplicial`
 (`hg_get.hypergraphs_summary`),
 69

`summary.net_hg(network_hypergraph)`, 159

`summary.net_hg_cluster`, 194

`summary.net_hg_mmsbm`
 (`hg_get.hypergraphs_summary`),
 69

`summary.net_hg_topics`
 (`hg_get.hypergraphs_summary`),
 69

`summary.net_hg_transduction`, 195

`temporal_hypergraph`, 195

`temporal_hypergraph()`, 14, 17, 53, 55, 81,
 112, 120, 124, 125, 129, 150–152,
 171, 176

`text_hypergat(hg_hypergat)`, 91

`text_hypergat()`, 18

`text_hypergraph`, 198

`text_hypergraph()`, 6, 7, 10, 20, 22, 28, 29,
 35, 36, 38, 39, 52, 55, 86, 89, 93, 95,
 97, 103, 107, 109, 114, 115, 117,
 121, 122, 126, 129, 130, 137, 139,
 151, 162, 184, 186, 193

`topic_network`, 202

`tutoring_events`, 204

`window_hypergraph`, 205

`window_hypergraph()`, 28, 30, 42, 70, 71,
 102, 107, 108, 111, 121–123, 129,
 150, 151, 168, 200