Package 'birddog'

April 4, 2026

Title Sniffing Emergence and Trajectories in Academic Papers and Patents

Version 1.0.4

Description Provides a unified set of methods to detect scientific emergence and technological trajectories in academic papers and patents. The package combines citation network analysis with community detection and attribute extraction, also applying natural language processing (NLP) and structural topic modeling (STM) to uncover the contents of research communities. It implements metrics and visualizations of community trajectories, including novelty indicators, citation cycle time, and main path analysis, allowing researchers to map and interpret the dynamics of emerging knowledge fields. Applications of the method include: Souza et al. (2022) <doi:10.1002/bbb.2441>, Souza et al. (2022) <doi:10.14211/ibjesb.e1742>, Matos et al. (2023) <doi:10.1007/s43938-023-00036-3>, Maria et al. (2023) <doi:10.3390/su15020967>, Biazatti et al. (2024) <doi:10.1016/j.envdev.2024.101074>, Felizardo et al. (2025) <doi:10.1007/s12649-025-03136-z>, and Miranda et al. (2025) <doi:10.1016/j.ijhydene.2025.01.089>.

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.3

Imports dplyr, ggraph, ggplot2, plotly, igraph, tidygraph, tidyr, tibble, Matrix, purrr, readr, rlang, glue, openalexR, RColorBrewer, scales, stringr

VignetteBuilder knitr

Suggests benchmarkme, knitr, rmarkdown, cli, ggHoriPlot, ggrepel, ggthemes, janitor, gt, testthat (>= 3.0.0), tictoc, viridis, zoo, stm, tidytext, udpipe

Config/testthat/edition 3

Depends R (>= 4.1.0)

URL http://roneyfraga.com/birddog/, https://github.com/roneyfraga/birddog

BugReports https://github.com/roneyfraga/birddog/issues

NeedsCompilation no

Author Roney Fraga Souza [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-5750-489X>),
Luis Felipe de Souza Rodrigues [ctb]

Maintainer Roney Fraga Souza <roneyfraga@gmail.com>

Repository CRAN

Date/Publication 2026-04-04 19:00:03 UTC

Contents

birddog-package *birddog: sniffing emergence and trajectories in academic papers and patents*

Description

Tools to detect emergence and trace technological/scientific trajectories in papers and patents. It reads OpenAlex and Web of Science data, builds citation-based networks, identifies groups, and summarizes their dynamics.

Links

- Website: http://roneyfraga.com/birddog/
- GitHub: https://github.com/roneyfraga/birddog
- Issues: https://github.com/roneyfraga/birddog/issues

Author(s)

Maintainer: Roney Fraga Souza <roneyfraga@gmail.com> (ORCID) [copyright holder]

Other contributors:

- Luis Felipe de Souza Rodrigues <lfsouza25@gmail.com> [contributor]

See Also

Useful links:

- Report bugs at https://github.com/roneyfraga/birddog/issues

attach_docs_to_vertices *Attach document IDs to graph vertices*

Description

Adds document ID lists to each vertex in the graph based on the group-document mapping.

Usage

```
attach_docs_to_vertices(g, docs_tbl)
```

Arguments

g	igraph object
docs_tbl	Tibble with columns group_id and document_id

Value

Modified igraph with `doc_ids` vertex attribute

`detect_main_trajectories` *Detect main temporal trajectories in group-year DAG*

Description

Identifies the most significant temporal trajectories within a group's evolution over time by building a directed acyclic graph (DAG) from similarity data and extracting highest-scoring disjoint paths using dynamic programming.

Usage

```
detect_main_trajectories(
  groups_cumulative_trajectories,
  group,
  jaccard_min = 0.05,
  intra_min = 0.1,
  k_out = 2,
  alpha = 1,
  beta = 0.1,
  top_M = 5,
  min_len = 3,
  use_docs_per_group = TRUE
)
```

Arguments

`groups_cumulative_trajectories`
: List containing three components:

- `groups_similarity`: Nested list with similarity data for each group, containing edges with `from`, `to`, `weight` (Jaccard), and `documents`
- `groups_attributes`: Nested list with node attributes for each group, containing `quantity_papers`, `prop_tracked_intra_group`, `tracked_documents`, `PY.sd`, and `network_until`
- `docs_per_group`: Data frame mapping group IDs to document IDs for accurate unique document counting

`group`
: Character ID of the group to analyze (e.g., "component1_g01")

`jaccard_min`
: Minimum Jaccard similarity for edges (default: 0.05). Higher values create sparser graphs with stronger connections.

`intra_min`
: Minimum proportion of tracked documents within group for nodes (default: 0.10). Higher values filter out weaker nodes.

`k_out` Maximum number of outgoing edges to keep per node (default: 2). Controls graph sparsity - lower values create simpler backbone structures.

`alpha` Weight for edge strength in path scoring (default: 1). Higher values emphasize transition strength over node quality.

`beta` Per-step persistence bonus in path scoring (default: 0.1). Higher values encourage longer trajectories.

`top_M` Maximum number of disjoint trajectories to extract (default: 5)

`min_len` Minimum number of distinct years for valid trajectory (default: 3)

`use_docs_per_group` Whether to use document IDs for accurate unique document counting (default: TRUE). If FALSE, uses approximation.

Details

This function implements a comprehensive pipeline for detecting significant temporal trajectories in research group evolution:

Algorithm Overview:

1. **Build Temporal DAG**: Constructs a directed acyclic graph where:
   - Nodes represent group-year combinations filtered by `intra_min` quality threshold
   - Edges represent transitions between consecutive years filtered by `jaccard_min`
   - Graph is sparsified to top `k_out` edges per node
2. **Score Components**: Computes node and edge scores:
   - Node score: $s_v = \log(1 + \text{quantity\_papers}_v \times \text{prop\_tracked\_intra\_group}_v)$
   - Edge score: $s_e = \text{weight}_e \times \log(1 + \text{documents}_e)$
3. **Extract Trajectories**: Uses dynamic programming to find heaviest paths:
   - Path score: $\text{best}(v) = \max\left(s_v, \max_{u \to v}\left(\text{best}(u) + s_v + \alpha \cdot s_{(u,v)} + \beta\right)\right)$
   - Iteratively extracts top `top_M` disjoint trajectories
   - Trajectories must span at least `min_len` distinct years
4. **Count Documents**: Calculates unique document coverage:
   - If `use_docs_per_group = TRUE`: Exact count via set union of document IDs
   - Otherwise: Approximation: $\sum$ node documents $-$ $\sum$ edge documents

Parameter Tuning Guidance:

- For **smoother, longer trajectories**: Increase `beta` (persistence bonus)
- For **transition-focused scoring**: Increase `alpha` (edge weight)
- For **denser connectivity**: Lower `jaccard_min` or increase `k_out`
- For **higher quality nodes**: Increase `intra_min`
- For **exact document counts**: Ensure `use_docs_per_group = TRUE` and provide `docs_per_group` data

Value

A list with two components:

- `graph`: An igraph object representing the temporal DAG with scoring attributes and optional document IDs
- `trajectories`: A tibble of detected trajectories sorted by score, with columns:
  - `traj_id`: Trajectory identifier ("tr1", "tr2", ...)
  - `start`, `end`: First and last year of the trajectory
  - `length`: Number of distinct years in the trajectory
  - `nodes`: List of node names along the path (e.g., "y2009g03")
  - `score`: Total path score from dynamic programming
  - `mean_w`: Mean edge score along the path
  - `sum_docs`: Count of unique documents covered by the path
  - `mean_size`: Mean node size (quantity_papers × proportion tracked)
  - `mean_PYsd`: Mean publication year standard deviation

See Also

`filter_trajectories()` for post-processing detected trajectories, `plot_group_trajectories_lines_2d()` and `plot_group_trajectories_lines_3d()` for visualization

Examples

```
## Not run:
# Basic usage with default parameters
trajectories <- detect_main_trajectories(
  groups_cumulative_trajectories = my_data,
  group = "component1_g01"
)

# Tuned for longer, transition-focused trajectories
trajectories <- detect_main_trajectories(
  groups_cumulative_trajectories = my_data,
  group = "component1_g01",
  jaccard_min = 0.03, # More permissive connectivity
  k_out = 3, # Denser backbone
  alpha = 1.5, # Emphasize edge strength
  beta = 0.2, # Encourage longer paths
  top_M = 8, # Extract more trajectories
  min_len = 4 # Require longer trajectories
)

# Access results
graph <- trajectories$graph
trajectory_data <- trajectories$trajectories

# Plot the top trajectory
top_trajectory <- trajectory_data[1, ]
```

```
## End(Not run)
```

`filter_trajectories` *Filter and rank detected trajectories*

Description

Applies post-processing filters and ranking to trajectory data based on score, length, and other criteria. This function helps refine the output from `detect_main_trajectories()` by keeping only the most relevant trajectories according to user-specified constraints.

Usage

```
filter_trajectories(tr_tbl, top_n = 3, min_score = NULL, min_length = NULL)
```

Arguments

`tr_tbl`	A tibble of trajectories from `detect_main_trajectories()$trajectories`. Must contain at least `traj_id`, `score`, and `length` columns.
`top_n`	Maximum number of trajectories to keep after filtering and sorting (default: 3). If `NULL`, keeps all trajectories that meet the filter criteria.
`min_score`	Minimum score threshold for trajectories (default: `NULL`). Trajectories with score less than `min_score` are discarded. Useful for removing weak or noisy paths.
`min_length`	Minimum trajectory length in distinct years (default: `NULL`). Trajectories shorter than `min_length` are discarded. Ensures only trajectories spanning a meaningful temporal horizon are kept.

Details

This function provides a straightforward way to refine trajectory detection results by applying quality filters and ranking. The filtering process occurs in three steps:

1. **Quality Filtering**: Remove trajectories that don't meet minimum quality standards
   - `min_score`: Filters by the dynamic programming path score (higher = better)
   - `min_length`: Filters by temporal span in distinct years
2. **Ranking**: Sort remaining trajectories by descending score to prioritize the most significant paths
3. **Selection**: Keep only the top `top_n` trajectories after filtering and sorting

Typical Use Cases:

- **Focus on strongest signals**: Use `min_score` to remove low-confidence trajectories
- **Ensure temporal significance**: Use `min_length` to require multi-year evolution
- **Limit visualization complexity**: Use `top_n` to focus on the most important paths
- **Progressive refinement**: Chain multiple calls with different criteria

Value

A filtered and sorted trajectory tibble with the same structure as input, containing only trajectories that meet all criteria, sorted by descending score. Returns an empty tibble if no trajectories meet the criteria.

See Also

`detect_main_trajectories()` for generating the trajectory data, `plot_group_trajectories_lines_2d()` and `plot_group_trajectories_lines_3d()` for visualizing filtered trajectories

Examples

```
## Not run:
# Get trajectories first
traj_data <- detect_main_trajectories(
  groups_cumulative_trajectories = my_data,
  group = "component1_g01"
)

# Basic: Keep top 3 trajectories by score
top_trajectories <- filter_trajectories(traj_data$trajectories)

# Keep top 5 trajectories with minimum quality standards
quality_trajectories <- filter_trajectories(
  tr_tbl = traj_data$trajectories,
  top_n = 5,
  min_score = 10,
  min_length = 4
)

# Keep all trajectories meeting minimum length (no top_n limit)
long_trajectories <- filter_trajectories(
  tr_tbl = traj_data$trajectories,
  top_n = NULL,
  min_length = 5
)

# Very strict filtering for high-quality, long trajectories
strict_trajectories <- filter_trajectories(
  tr_tbl = traj_data$trajectories,
  top_n = 3,
  min_score = 15,
  min_length = 6
)

# Use filtered trajectories for visualization
plot_group_trajectories_lines_2d(
  traj_data = traj_data,
  traj_filtered = quality_trajectories
)

## End(Not run)
```

get_openalex_fields *Get Fields from OpenAlex for Work IDs*

Description

Retrieves specified fields for OpenAlex work IDs using the OpenAlex API. Processes data in batches to avoid API rate limits.

Usage

```
get_openalex_fields(
  openalex_ids,
  variables = "publication_year",
  batch_size = 50,
  save_dir = NULL
)
```

Arguments

openalex_ids	Character vector of OpenAlex work IDs (format: "W1234567890") or a data frame/tibble containing a column named "CR" with OpenAlex IDs. IDs can be semicolon-separated strings which will be split automatically.
variables	Character vector of variable names to fetch from OpenAlex. Options include: "publication_year", "doi", "type", "source_display_name", or any valid OpenAlex work field. Default is "publication_year".
batch_size	Number of IDs to process per API call (default: 50). Smaller batches help avoid API rate limits.
save_dir	Optional path to directory where intermediate results should be saved as RDS files. If NULL (default), no saving occurs. Directory will be created if it doesn't exist.

Details

This function:

1. Accepts either a character vector of IDs or a data frame with a "CR" column
2. Splits semicolon-separated ID strings into individual IDs
3. Validates IDs against the pattern "^W\d+$"
4. Fetches specified variables from OpenAlex API in batches
5. Optionally saves each batch to disk as it's processed
6. Handles API errors gracefully with informative messages
7. Includes delays between batches to respect API rate limits

Value

A tibble with the following columns:

- `id`: The OpenAlex work ID
- One column for each requested variable (e.g., "publication_year", "doi", "type")

Rows without valid OpenAlex IDs or where API calls fail will have NA values.

Note

The OpenAlex API has rate limits. This function implements:

- Batch processing to reduce number of API calls
- 0.5 second delays between batches
- Error handling for failed API requests
- Progress messages to track execution
- Optional disk saving for data persistence

If you encounter rate limiting errors, consider reducing batch_size or implementing longer delays.

Examples

```
## Not run:
# From a character vector
ids <- c("W2261389918", "W1548650423", "W1504492735")
result <- get_openalex_fields(ids)

# Fetch multiple variables
result <- get_openalex_fields(
  ids,
  variables = c("publication_year", "doi", "type")
)

# From a data frame with CR column
oa_data <- data.frame(CR = c("W123;W456", "W789"))
result <- get_openalex_fields(oa_data)

# Save intermediate results while downloading
result <- get_openalex_fields(
  ids,
  variables = c("publication_year", "source_display_name"),
  save_dir = tempdir()
)

## End(Not run)
```

`mixed_sort` *Natural sort for alphanumeric strings*

Description

Sort character vectors containing embedded numbers in natural numeric order rather than lexicographic order. For example, `"c1g2"` comes before `"c1g10"`, which standard `sort()` would not guarantee.

Usage

```
mixed_sort(x)
```

Arguments

`x`	Character vector to sort.

Details

The function extracts all numeric segments from each string using `stringr::str_extract_all()` and sorts by them sequentially (first number, then second). Designed for two-segment identifiers such as `"c1g1"`, `"c2g10"`, etc.

Value

A character vector sorted in natural numeric order.

See Also

`sort()` for standard lexicographic sorting.

Examples

```
mixed_sort(c("c1g10", "c1g2", "c1g1", "c1g9"))

mixed_sort(c("group12", "group1", "group3"))
```

plot_group_trajectories_2d *Visualize 2D Technological Trajectories from Group Evolution*

Description

Creates a 2D visualization of technological trajectories based on group similarity metrics, showing the evolution of research groups over time with node size representing group importance and color representing publication-year deviation.

Usage

```
plot_group_trajectories_2d(
  groups_cumulative_trajectories,
  group = "c1g1",
  jaccard_similarity = 0.01,
  prop_tracked_intra_group_treshold = 0.2,
  label_type = "size",
  label_vertical_position = 0,
  label_horizontal_position = 0,
  label_angle = 0,
  time_span = NA,
  show_legend = TRUE
)
```

Arguments

groups_cumulative_trajectories	A list with components groups_similarity and groups_attributes, typically produced by plot_groups_trajectories(). The groups_similarity element must be a named list of edge tables (one per group) with at least from, to, and weight; the groups_attributes element must be a named list of node tables containing, among others, network_until, quantity_papers, prop_tracked_intra_group, tracked_documents, and PY.sd.
group	The specific group to visualize (default: "c1g1").
jaccard_similarity	Minimum Jaccard similarity threshold for connections (default: 0.1).
prop_tracked_intra_group_treshold	Minimum proportion of tracked intra-group documents for nodes to be included (default: 0.2).
label_type	Type of labels to display on nodes ("size" for weighted size or "id" for group IDs).
label_vertical_position	Vertical adjustment for node labels (default: 0).
label_horizontal_position	Horizontal adjustment for node labels (default: 0).

label_angle	Angle for node labels (default: 0).
time_span	Optional vector of years to display; if NA, shows all (default: NA).
show_legend	Logical indicating whether to show the color legend (default: TRUE).

Value

A ggplot2 object visualizing the technological trajectories.

Examples

```
## Not run:
# Compute trajectories first
traj_data <- plot_groups_trajectories(groups_cumulative)

# Visualize a specific group (pass the whole object; the function extracts what it needs internally)
plot_group_trajectories_2d(
  groups_cumulative_trajectories = traj_data,
  group = "c1g5",
  jaccard_similarity = 0.3
)

## End(Not run)
```

---

plot_group_trajectories_3d

Visualize 3D Technological Trajectories from Group Evolution

Description

Creates an interactive 3D visualization of technological trajectories showing the evolution of research groups over time with node size representing group importance and color representing publication year deviation.

Usage

```
plot_group_trajectories_3d(
  groups_cumulative_trajectories,
  group = "component1_g01",
  jaccard_similarity = 0.1,
  prop_tracked_intra_group_treshold = 0.2,
  label_type = "size",
  label_vertical_position = 0,
  label_horizontal_position = 0,
  label_angle = 0,
  time_span = NA,
  show_legend = TRUE,
  last_year_keywords = NULL,
```

```
  log_scale = FALSE
)
```

Arguments

groups_cumulative_trajectories	A list containing two components: • groups_similarity: Similarity data between groups • groups_attributes: Attribute data for each group
group	The specific group to visualize (default: "component1_g01")
jaccard_similarity	Minimum Jaccard similarity threshold for connections (default: 0.1)
prop_tracked_intra_group_treshold	Minimum proportion of tracked intra-group documents for nodes to be included (default: 0.2)
label_type	Type of labels to display on nodes ("size" for weighted size or "id" for group IDs)
label_vertical_position	Vertical adjustment for node labels (default: 0)
label_horizontal_position	Horizontal adjustment for node labels (default: 0)
label_angle	Angle for node labels (default: 0)
time_span	Optional vector specifying the time span to display (default: NA shows all years)
show_legend	Logical indicating whether to show the color legend (default: TRUE)
last_year_keywords	Optional keywords description for the last year (default: NULL)
log_scale	Whether to apply log transformation to the z-axis size values (default: FALSE). Uses `log1p()` (i.e., log(1 + x)) to compress large differences between node sizes.

Value

A plotly 3D visualization object

Examples

```
## Not run:
# First get trajectory data
traj_data <- sniff_groups_trajectories(groups_cumulative)

# Visualize a specific group in 3D
plot_group_trajectory_3d(
  groups_cumulative_trajectories = traj_data,
  group = "component1_g05",
  jaccard_similarity = 0.2
)
```

```
## End(Not run)
```

---

plot_group_trajectories_lines_2d *Plot 2D trajectories as variable-width lines*

Description

Creates a 2D line plot showing research trajectories over time, with highlighted trajectories displayed as variable-width lines and optional background trajectories shown in lowlight style. Edge widths grow along each highlighted trajectory based on cumulative paper counts, and labels are placed at trajectory endpoints.

Usage

```
plot_group_trajectories_lines_2d(
  traj_data,
  traj_filtered,
  title = "Main trajectories",
  width_range = c(0.8, 6),
  width_by_traj_size = TRUE,
  use_raw_papers = FALSE,
  label_nudge_x = 0.3,
  label_size = 4,
  show_only_highlighted = FALSE,
  lowlight_width = 0.9,
  lowlight_alpha = 0.22,
  lowlight_color = "#9AA5B1"
)
```

Arguments

`traj_data`	List containing trajectory data generated by `detect_main_trajectories()` with components: • `graph`: igraph object containing nodes and edges across years • `trajectories`: tibble of all candidate trajectories (traj_id + nodes list)
`traj_filtered`	Filtered trajectories tibble from `filter_trajectories()` containing the subset to emphasize. Must contain columns: • `traj_id`: trajectory identifiers • `nodes`: list of character vectors (ordered by time or orderable)
`title`	Plot title (default: "Main trajectories")
`width_range`	Range for edge widths of highlighted trajectories (default: c(0.8, 6.0)). Width at each segment is scaled by cumulative paper count up to the next node.

`width_by_traj_size`
: Whether to scale each highlighted trajectory's line width proportionally to its total paper count (default: TRUE). When TRUE, the trajectory with the most papers uses the full `width_range`, while smaller trajectories are proportionally thinner (minimum 25% of full range).

`use_raw_papers`
: Whether to use raw paper counts for width scaling (default: FALSE). If TRUE, uses raw `quantity_papers`; if FALSE, uses weighted size: `quantity_papers * prop_tracked_intra_group`.

`label_nudge_x`
: Horizontal nudge for trajectory end labels to prevent overlap with nodes (default: 0.30)

`label_size`
: Text size for trajectory end labels (default: 4)

`show_only_highlighted`
: Whether to show only highlighted trajectories (default: FALSE). If TRUE, hides all non-highlighted trajectory lines; if FALSE, draws lowlight background.

`lowlight_width`
: Line width for lowlight (background) edges (default: 0.9)

`lowlight_alpha`
: Transparency for lowlight edges (default: 0.22; smaller values = more transparent)

`lowlight_color`
: Color for lowlight edges (default: "#9AA5B1" - neutral gray)

Details

This function visualizes research trajectories as variable-width lines:

- **Highlighted trajectories** (`traj_filtered`) are colored lines with widths proportional to cumulative paper counts (raw or weighted)
- **Background trajectories** (when `show_only_highlighted = FALSE`) are shown as thin, transparent lines
- **Trajectory labels** are placed at the end of each highlighted trajectory
- The **x-axis represents publication years** using a Sugiyama layout
- The **y-axis shows vertical positions** from the layout (no intrinsic meaning)
- Colors are assigned only to highlighted trajectories present in the plot

When `traj_data$trajectories` is available and `show_only_highlighted = FALSE`, the lowlight layer shows only edges that belong to any trajectory but not the highlighted set. Otherwise, it shows the entire graph minus highlighted edges.

Value

A ggplot object displaying the trajectory network

Examples

```
## Not run:
# Detect main trajectories first
traj_data <- detect_main_trajectories(your_graph_data)

# Filter trajectories of interest
```

```
filtered_traj <- filter_trajectories(traj_data$trajectories,
                                     min_papers = 10)

# Create the plot
plot_group_trajectories_lines_2d(
  traj_data = traj_data,
  traj_filtered = filtered_traj,
  title = "Key Research Trajectories",
  width_range = c(1, 8),
  show_only_highlighted = FALSE
)

## End(Not run)
```

`plot_group_trajectories_lines_3d` *Plot 3D trajectories as variable-width lines*

Description

Creates an interactive 3D plot showing research trajectories with time on the x-axis, route separation on the y-axis, and cumulative paper counts on the z-axis. Highlighted trajectories are displayed as growing-thickness lines, with optional background trajectories and network context in lowlight style.

Usage

```
plot_group_trajectories_lines_3d(
  traj_data,
  traj_filtered,
  width_range_hi = c(4, 12),
  width_range_lo = c(1.2, 3),
  use_raw_papers = TRUE,
  log_scale = FALSE,
  connect_only_existing_edges = TRUE,
  show_labels = TRUE,
  show_only_highlighted = FALSE,
  label_size = 18,
  hover_font_size = 12,
  lowlight_width = 1,
  lowlight_alpha = 0.9,
  lowlight_color = "#9AA5B1",
  width_by_traj_size = TRUE,
  group_id = NULL
)
```

Arguments

traj_data	List containing trajectory data generated by detect_main_trajectories() with components: • graph: igraph object containing nodes and edges across years • trajectories: tibble of all candidate trajectories (traj_id + nodes list)
traj_filtered	Filtered trajectories tibble from filter_trajectories() containing the subset to emphasize. Must contain columns: • traj_id: trajectory identifiers • nodes: list of character vectors (ordered by time or orderable)
width_range_hi	Width range for highlighted trajectory segments (default: c(4, 12)). Segment widths scale with cumulative paper counts.
width_range_lo	Baseline width range used to compute constant lowlight width (default: c(1.2, 3)). The mean of this range determines lowlight width.
use_raw_papers	Whether to use raw paper counts for z-axis scaling (default: TRUE). If TRUE, uses raw quantity_papers; if FALSE, uses weighted size: quantity_papers * prop_tracked_intra_group.
log_scale	Whether to apply log transformation to cumulative document counts on the z-axis (default: FALSE). Uses log1p() (i.e., log(1 + x)) to handle zero values. Useful when trajectories have very different growth rates, compressing the z-axis to better visualize relative trajectory shapes.
connect_only_existing_edges	Whether to draw only edges that exist in the graph (default: TRUE). If FALSE, draws all consecutive node pairs in trajectories regardless of graph edges.
show_labels	Whether to add end-of-trajectory labels inside the 3D plot (default: TRUE)
show_only_highlighted	Whether to show only highlighted trajectories (default: FALSE). If TRUE, hides all background network and lowlight trajectories.
label_size	Font size for trajectory end labels (default: 18)
hover_font_size	Font size for hover tooltips (default: 12)
lowlight_width	Line width for lowlight trajectories and background network (default: 1)
lowlight_alpha	Transparency for lowlight elements (default: 0.9)
lowlight_color	Color for lowlight elements (default: "#9AA5B1" - neutral gray)
width_by_traj_size	Whether to scale each highlighted trajectory's line width proportionally to its total paper count (default: TRUE). When TRUE, the trajectory with the most papers uses the full width_range_hi, while smaller trajectories are proportionally thinner (minimum 25% of full range).
group_id	Optional group identifier to display in the plot title (default: NULL). When provided, the title becomes "3D Trajectories - group_id".

Details

This function creates an interactive 3D visualization of research trajectories:

- **X-axis**: Publication year (parsed from vertex names like "y2007g05")
- **Y-axis**: "Route" (Sugiyama layout coordinate to separate trajectories vertically)
- **Z-axis**: Cumulative documents (raw or weighted) along each trajectory

Key features:

- **Highlighted trajectories** (`traj_filtered`) are colored lines with widths that grow proportionally to cumulative paper counts
- **Lowlight trajectories** (when `show_only_highlighted = FALSE`) show other trajectories as constant-width lines
- **Background network** (when `show_only_highlighted = FALSE`) provides context with thin gray edges
- **Hover tooltips** show detailed information at each trajectory point
- **End labels** identify highlighted trajectories (when `show_labels = TRUE`)
- **Edge validation** (when `connect_only_existing_edges = TRUE`) ensures only actual graph edges are drawn

The function uses a Sugiyama layout for the y-axis coordinates and cumulative sums of paper counts for the z-axis values. Colors for highlighted trajectories are assigned using RColorBrewer's Set2 palette (for <=8 trajectories) or a hue-based palette (for more trajectories).

Value

A plotly interactive 3D plot object

Examples

```
## Not run:
# Detect main trajectories first
traj_data <- detect_main_trajectories(your_graph_data)

# Filter trajectories of interest
filtered_traj <- filter_trajectories(traj_data$trajectories,
                                     min_papers = 10)

# Create interactive 3D plot
plot_group_trajectories_lines_3d(
  traj_data = traj_data,
  traj_filtered = filtered_traj,
  width_range_hi = c(3, 10),
  use_raw_papers = FALSE,
  show_labels = TRUE
)

# Log-scale z-axis for trajectories with very different growth rates
plot_group_trajectories_lines_3d(
```

```
  traj_data = traj_data,
  traj_filtered = filtered_traj,
  log_scale = TRUE
)

# Minimal view with only highlighted trajectories
plot_group_trajectories_lines_3d(
  traj_data = traj_data,
  traj_filtered = filtered_traj,
  show_only_highlighted = TRUE,
  label_size = 16
)

## End(Not run)
```

`read_openalex` *Read and Process OpenAlex data*

Description

Parse datasets exported from **OpenAlex** in two ways: (1) a CSV file exported in the browser, or (2) a data frame obtained via the {openalexR} API helpers. The function standardizes fields to common bibliographic tags (e.g., AU, SO, CR, PY, DI) and returns a tidy tibble.

Usage

```
read_openalex(file, format = "csv")
```

Arguments

`file`	For `format = "csv"`, a character string with a local path or an HTTP(S) URL to a CSV export. For `format = "api"`, a data frame produced by {openalexR} for the **works** entity.
`format`	Either `"csv"` (CSV export) or `"api"` (data frame from {openalexR}).

Details

CSV mode (`format = "csv"`)**:**

- If `file` is a URL, it is downloaded to a temporary file before parsing (a progress message is printed).
- Selected fields are mapped to standardized tags: `id_short` (short OpenAlex ID), SR (= `id_short`), PY (= `publication_year`), TI (= `title`), DI (= `doi`), DT (= `type`), DE (= `keywords.display_name`), AB (= `abstract`), AU (= `authorships.author.display_name`), SO (= `locations.source.display_name`), C1 (= `authorships.countries`), TC (= `cited_by_count`), SC (= `primary_topic.field.display_name`), CR (= `referenced_works`, with the `https://openalex.org/` prefix stripped), and DB = `"openalex_csv"`.

- PY is coerced to numeric; a helper column DI2 (uppercase, punctuation-stripped variant of DI) is added; columns with all-caps tags are placed first and DI2 is relocated after DI.

API mode (format = "api"):

- file must be a data frame containing at least column id; typically this is returned by openalexR::oa_request() + openalexR::oa2df() or similar.
- Records are filtered to type %in% c("article","review") and deduplicated by id.
- The function derives:
  - id_short (= id without the https://openalex.org/ prefix) and SR (= id_short);
  - CR: concatenated short IDs from referenced_works (semicolon-separated);
  - DE: concatenated keyword names (lower case) from keywords;
  - AU: concatenated author names (upper case) from authorships;
  - plus core fields PY (= publication_year), TC (= cited_by_count), TI (= title), AB (= abstract), DI (= doi), and DB = "openalex_api".
- The result keeps one row per id and may include original columns from the input (via a right join), after constructing the standardized fields above.

Value

A tibble with standardized bibliographic columns. Typical output includes: id_short, AU, DI, CR, SO, DT, DE, AB, C1, TC, SC, SR, PY, and DB (source flag: "openalex_csv" or "openalex_api"). See **Details**.

Supported inputs

- format = "csv" — a local path or an HTTP(S) URL to an OpenAlex **CSV** export.
- format = "api" — a **data frame** produced by {openalexR} for the **works** entity (with the usual OpenAlex columns, including list-columns such as keywords, authorships, and referenced_works).

See Also

OpenAlex R client: oa_request, oa2df. Importers for Web of Science: read_wos.

Examples

```
## Not run:
## CSV export (local path)
x <- read_openalex("openalex-works.csv", format = "csv")

## Using the API with openalexR
library(openalexR)
url_api <- "https://api.openalex.org/works?page=1&filter=primary_location.source.id:s121026525"
df_api  <- openalexR::oa_request(query_url = url_api) |>
  openalexR::oa2df(entity = "works")
y <- read_openalex(df_api, format = "api")

## End(Not run)
```

read_wos *Read Web of Science exported files*

Description

Parse Web of Science (WoS) export files in multiple formats and return a tidy table. The function automatically dispatches to a specialized parser based on the `format` argument and can also **download from a URL** if `file` points to an `http://` or `https://` resource.

Usage

```
read_wos(file, format = "bib", normalized_names = TRUE)
```

Arguments

`file`	Character scalar or vector. Path(s) to a WoS export file, or a single URL (`http://` or `https://`) pointing to a WoS export.
`format`	Character scalar. Export format; one of `"bib"`, `"ris"`, `"txt-plain-text"`, or `"txt-tab-delimited"`.
`normalized_names`	Logical. If `TRUE` (default), use standardized column names when possible; if `FALSE`, keep original WoS field tags.

Details

- `file` may be a single path/URL or a **vector of paths**; multiple files will be combined row-wise when applicable.
- When `file` is a URL, the file is downloaded to a temporary path before parsing (a progress message is printed).
- If `normalized_names = TRUE`, selected WoS tags are mapped to standardized names (e.g., `AU` -> `author`, `TI` -> `title`, `PY` -> `year`, `DI` -> `doi`, `DE` -> `keywords`, `SR` -> `unique_id`, etc.; the exact mapping depends on the format). Otherwise, original field tags are preserved.
- The output includes:
  - `DI2`: an uppercase, punctuation-stripped variant of `DI` (if present),
  - `PY`: coerced to numeric (when present),
  - `DB`: a provenance flag indicating the source/format and whether names were normalized.
- Columns with ALL-CAPS tags (e.g., `AU`, `TI`, `PY`) are placed first, followed by other columns, and `DI2` is relocated just after `DI`.

Value

A tibble with the parsed WoS records. See **Details** for notes on added/coerced columns (`DI2`, `PY`, `DB`) and column ordering.

Supported formats

- "bib" — BibTeX export
- "ris" — RIS export
- "txt-plain-text" — Plain-text export
- "txt-tab-delimited" — Tab-delimited export

See Also

Internal parsers used by this function: read_wos_bib, read_wos_ris, read_wos_plain, read_wos_tab.

Examples

```
bib_file <- system.file("extdata", "sample_wos.bib", package = "birddog")
M <- read_wos(bib_file, format = "bib", normalized_names = TRUE)
head(M)

## Not run:
# load data from a URL
M <- read_wos("https://example.com/savedrecs.bib", format = "bib")

## End(Not run)
```

---

sniff_citations_cycle_time

Calculate Citation Cycle Time (CCT) indicator

Description

Calculates the Citation Cycle Time (CCT) to measure the pace of scientific or technological progress in a publication network. Based on Kayal (1999), the indicator measures the median age of cited publications, where lower values indicate faster knowledge replacement cycles.

Usage

```
sniff_citations_cycle_time(
  network,
  scope = "groups",
  start_year = NULL,
  end_year = NULL,
  tracked_cr_py = NULL,
  batch_size = 50,
  min_papers_per_year = 3,
  rolling_window = NULL
)
```

Arguments

network	Required. Network object containing publication data. For scope = "groups": object returned by sniff_groups(). For scope = "network": network object (tbl_graph or igraph).
scope	Analysis scope. Either "groups" (default) for separate group analysis or "network" for complete network analysis.
start_year, end_year	Start and end years for temporal analysis. If not specified, uses minimum and maximum years found in the data.
tracked_cr_py	Pre-processed citation year data (optional). A tibble with columns CR (OpenAlex work ID) and CR_PY (publication year). If provided, skips fetching data from OpenAlex API. Useful for avoiding repeated API calls.
batch_size	For OpenAlex data: number of IDs to process per API call (default: 50). Smaller batches help avoid API rate limits, larger batches process data faster but may trigger rate limiting.
min_papers_per_year	Minimum number of papers required in a given year to compute CCT. Years with fewer papers are reported as NA (default: 3).
rolling_window	Optional integer for rolling window smoothing. If provided, CCT values are smoothed using a centered moving average of the specified width (e.g., 3 for a 3-year window). Default is NULL (no smoothing).

Details

The Citation Cycle Time (CCT) is calculated following Kayal (1999):

1. Extract citation IDs from the network's CR column
2. Fetch publication years for cited works from OpenAlex API using get_openalex_fields()
3. For each publication, calculate the age of each cited reference (PY - CR_PY)
4. Calculate the median citation age per publication
5. For each year, calculate the median of per-publication medians across all publications in that year (annual mode)

Lower CCT values indicate that publications are citing more recent work, suggesting a faster pace of knowledge replacement. A sudden drop in CCT within a group signals potential scientific emergence.

The function automatically handles:

- Splitting semicolon-separated citation IDs
- Batch processing of OpenAlex API requests
- Filtering invalid citations (where cited work was published after citing work)
- Skipping years with too few papers (min_papers_per_year)
- Optional rolling window smoothing for noisy time series
- Creating temporal plots for each group

Value

A list with the following components:

`data`	Tibble with CCT data containing columns: group, year, index
`plots`	Named list of plotly objects showing temporal evolution of CCT for each group. Each plot shows both absolute CCT values and year-over-year differences.
`years_range`	Named vector with start_year and end_year used in the analysis
`tracked_cr_py`	Citation year data with columns CR and CR_PY. Can be saved and reused in subsequent analyses to avoid repeated API calls.

References

Kayal AA, Waters RC. An empirical evaluation of the technology cycle time indicator as a measure of the pace of technological progress in superconductor technology. IEEE Transactions on Engineering Management. 1999;46(2):127-31. doi:10.1109/17.759138

See Also

`sniff_groups()`, `get_openalex_fields()`, `indexes_plots()`

Examples

```
## Not run:
# Group analysis
results <- sniff_citations_cycle_time(network_groups, scope = "groups")

# Network analysis
results_network <- sniff_citations_cycle_time(complete_network, scope = "network")

# With rolling window smoothing
results_smooth <- sniff_citations_cycle_time(
  network_groups,
  scope = "groups",
  rolling_window = 3
)

# Accessing results
cct_data <- results$data
plots <- results$plots
plots$c1g1  # View plot for specific group

# Reuse citation data to avoid repeated API calls
saved_citations <- results$tracked_cr_py
results2 <- sniff_citations_cycle_time(
  network_groups,
  tracked_cr_py = saved_citations
)

## End(Not run)
```

`sniff_components` *Identify and Analyze Network Components*

Description

Detects connected components in a citation network and computes summary statistics for each component. Returns both the component information and an updated network with component labels.

Usage

```
sniff_components(net)
```

Arguments

`net`	A network object (tbl_graph or igraph) generated by `sniff_network()`

Value

A list with two elements:

components A tibble with component statistics containing:

- `component`: Component identifier (e.g., "c1")
- `quantity_publications`: Number of publications in component
- `average_age`: Mean publication year of component

network The input network with added component labels

Examples

```
## Not run:
# Create a network first
data <- read_wos("savedrecs.txt")
net <- sniff_network(data)

# Analyze components
result <- sniff_components(net)

# Access component information
result$components

# Get network with component labels
component_net <- result$network

## End(Not run)
```

sniff_entropy *Calculate Entropy Based on Keywords Over Time*

Description

Computes the normalized Shannon entropy of keyword distributions from scientific publications over a specified time range. Entropy measures the diversity and evenness of keyword usage within research groups or the entire network.

Usage

```
sniff_entropy(
  network,
  scope = "groups",
  start_year = NULL,
  end_year = NULL,
  mode = "rolling",
  window_size = 5
)
```

Arguments

network	A network object to analyze. For `scope = "groups"`, this should be the output of `sniff_groups()`. For `scope = "network"`, this should be a `tbl_graph` or `igraph` object from `sniff_network()`.
scope	Character specifying the analysis scope: "groups" for multiple groups or "network" for the entire network (default: "groups").
start_year	Starting year for entropy calculation. If NULL, uses the minimum publication year found in the network data.
end_year	Ending year for entropy calculation. If NULL, uses the maximum publication year found in the network data.
mode	Character specifying the temporal mode for entropy calculation: **"annual"** Uses only publications from each specific year (default). **"cumulative"** Uses all publications from the start up to each year. **"rolling"** Uses a sliding window of `window_size` years ending at each year.
window_size	Integer specifying the rolling window size in years (default: 5). Only used when `mode = "rolling"`.

Details

The function calculates the normalized Shannon entropy (Pielou's evenness index) based on Shannon's information theory (Shannon, 1948). The temporal scope of keyword data depends on the `mode` parameter:

- **annual**: entropy from keywords published in each specific year. Values tend to be high (near 1) since within-year distributions are typically even.

- **cumulative**: entropy from all keywords published up to each year. Shows long-term trends in thematic concentration as the keyword pool grows.
- **rolling**: entropy from a sliding window of recent years. Balances sensitivity to recent shifts with enough data for stable estimates.

The normalized entropy (Pielou's J') is calculated as:

$$J' = \frac{H}{H_{max}} = \frac{-\sum_{i=1}^{n} p_i \log_2 p_i}{\log_2 n}$$

where p_i is the relative frequency of keyword i, n is the number of unique keywords, and $H_{max} = \log_2 n$ is the maximum possible entropy for n categories.

Entropy values range from 0 to 1, where:

- 0 indicates minimal diversity (one dominant keyword)
- 1 indicates maximal diversity (all keywords equally frequent)

A sudden increase in entropy may signal the emergence of new research topics, while a decrease suggests thematic convergence.

Value

A list with three components:

`data`	A tibble containing entropy values for each group and year
`plots`	A list of plotly objects visualizing entropy trends for each group
`years_range`	A vector with the start_year and end_year used in calculations

References

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3), 379-423. doi:10.1002/j.15387305.1948.tb01338.x

Pielou, E. C. (1966). The measurement of diversity in different types of biological collections. *Journal of Theoretical Biology*, 13, 131-144.

See Also

`sniff_groups`, `sniff_network`, `indexes_plots`

Examples

```
## Not run:
# Rolling window (default: 5 years)
entropy_results <- sniff_entropy(groups_data, scope = "groups")

# Cumulative mode
entropy_results <- sniff_entropy(groups_data, mode = "cumulative")

# Annual mode
entropy_results <- sniff_entropy(groups_data, mode = "annual")
```

```
# Rolling window with custom size
entropy_results <- sniff_entropy(groups_data, mode = "rolling", window_size = 3)

# Access results
entropy_data <- entropy_results$data
entropy_plots <- entropy_results$plots

## End(Not run)
```

`sniff_groups` *Detect and analyze groups in a scientific network*

Description

This function identifies and analyzes groups (communities) within scientific networks created from articles and patents data. It can apply different clustering algorithms to detect technological trajectories and emerging scientific fields.

Usage

```
sniff_groups(
  comps,
  min_group_size = 10,
  keep_component = c("c1"),
  cluster_component = c("c1"),
  algorithm = "fast_greedy",
  seed = 888L
)
```

Arguments

`comps`	A list containing network components, typically generated by `sniff_components()`. Must include a network object with 'component' and 'PY' (publication year) vertex attributes.
`min_group_size`	Minimum size for a group to be included in results (default = 10). Groups with fewer members will be filtered out.
`keep_component`	Character vector specifying which network components to process (default = "c1"). Can include multiple components.
`cluster_component`	Character vector specifying which components should be clustered (default = "c1"). Components not listed here will be treated as single groups.
`algorithm`	Community detection algorithm to use (default = "fast_greedy"). Options include: "louvain", "walktrap", "edge_betweenness", "fast_greedy", or "leiden".
`seed`	Random seed for reproducible results (default = 888L). Only applies to algorithms that use random initialization like Louvain.

Details

The function first validates the input network, then applies the specified clustering algorithm to detect communities within the network. It calculates statistics for each detected group and returns the results along with the augmented network. The function can handle multiple network components simultaneously, applying clustering only to specified components.

Value

A list with three elements:

- `aggregate`: A data frame with group statistics including group name, number of papers, and average publication year
- `network`: The input network with added group attributes
- `pubs_by_year`: Publication counts by group and year

See Also

`sniff_components()` for creating the input network components

Examples

```
## Not run:
# Assuming 'comps' is output from sniff_components()
groups <- sniff_groups(comps,
  min_group_size = 15,
  algorithm = "leiden",
  seed = 888L
)

# Access group statistics
groups$aggregate
groups$network
groups$pubs_by_year

## End(Not run)
```

---

sniff_groups_attributes

Calculate and Visualize Group Attributes from Scientific Networks

---

Description

This function analyzes publication growth rates and other attributes for research groups identified in scientific networks. It calculates growth rates using exponential models, creates horizon plots for visualization, and generates summary tables.

Usage

```
sniff_groups_attributes(
  groups,
  growth_rate_period = 2010:2022,
  horizon_plot = TRUE,
  show_results = TRUE,
  assign_result = NULL
)
```

Arguments

groups	A list containing network data with publications by year and group information. Must include elements: network, pubs_by_year, and aggregate.
growth_rate_period	Numeric vector of years to use for growth rate calculation (default: 2010:2024).
horizon_plot	Logical indicating whether to include horizon plots in the output table (default: TRUE).
show_results	Logical indicating whether to print results to console (default: TRUE).
assign_result	Character string specifying a variable name to assign the results to in the global environment (default: NULL).

Details

The function performs the following steps:

1. Calculates growth rates using exponential models for each group
2. Processes publication age and doubling time metrics
3. Optionally creates horizon plots for each group's publication trend
4. Generates a comprehensive summary table

Value

A list with two components:

- attributes_table: A gt table showing group attributes including growth rates
- regression: A list of model summaries for each group's growth rate calculation

Examples

```
## Not run:
# Assuming groups is output from sniff_groups()
groups_attributes <- sniff_groups_attributes(groups,
  growth_rate_period = 2010:2022,
  horizon_plot = TRUE
)

# View the results table
print(groups_attributes$attributes_table)
```

```
# Access model summaries
groups_attributes$regression

## End(Not run)
```

---

sniff_groups_cumulative

Analyze Cumulative Network Groups Over Time

Description

Performs cumulative community detection on a network over specified time spans, returning group statistics and keyword analysis for each time period.

Usage

```
sniff_groups_cumulative(
  comps,
  time_span = NULL,
  min_group_size = 10,
  keep_component = c("c1"),
  cluster_component = c("c1"),
  top_n_keywords = 10,
  algorithm = "fast_greedy",
  seed = 888L
)
```

Arguments

comps	A list containing network components, typically generated by sniff_components(). Must include a network object with 'component' and 'PY' (publication year) vertex attributes.
time_span	Numeric vector of years to analyze (default: 2000:2024).
min_group_size	Minimum size for a cluster to be retained (default = 10).
keep_component	Character vector specifying which network components to process (default = "c1"). Can include multiple components.
cluster_component	Character vector specifying which components should be clustered (default = "c1"). Components not listed here will be treated as single groups.
top_n_keywords	Number of top keywords to extract per group (default = 10).
algorithm	Community detection algorithm to use. One of: "louvain", "walktrap", "edge_betweenness", "fast_greedy" (default), or "leiden".
seed	Random seed for reproducible results (default = 888L). Only applies to algorithms that use random initialization like Louvain.

Value

A named list (by year) where each element contains:

groups A tibble with group statistics and top keywords

documents A tibble mapping documents to groups

network The cumulative network up to that year

Examples

```
## Not run:
# Typical pipeline:
data <- read_wos("savedrecs.txt")
net <- sniff_network(data)
comps <- sniff_components(net)

# Cumulative analysis
groups_cumulative <- sniff_groups_cumulative(
  comps,
  time_span = 2010:2020,
  keep_component = c("c1", "c2"),
  cluster_component = c("c1"),
  algorithm = "leiden",
  seed = 888L
)

# Access results for 2015
groups_cumulative[["network_until_2015"]]$groups

## End(Not run)
```

---

sniff_groups_cumulative_citations

Calculate Cumulative Citations by Group and Year

Description

This function calculates cumulative citations for papers within research groups, tracking how citations accumulate over time for highly cited papers.

Usage

```
sniff_groups_cumulative_citations(groups, min_citations = 5)
```

Arguments

groups	A list containing network data with the following components: • network: A tidygraph network object • pubs_by_year: Publication counts by year • aggregate: Aggregate network statistics
min_citations	Minimum number of citations for a paper to be included in analysis (default: 10).

Details

For each research group, the function:

- Identifies papers with citations above the threshold
- Tracks citations to these papers year by year
- Calculates cumulative citation patterns
- Computes various growth metrics for citation analysis

Works with both Web of Science (WOS) and OpenAlex data formats.

Value

A named list (by research group) where each element contains a tibble with:

- group: Research group identifier
- SR: Paper identifier
- TC: Total citations
- PY: Publication year
- Ki: Total network citations
- citations_by_year: A tibble with annual citation counts (PY: year, citations: count)
- growth_power: Growth power score (0-100)
- growth_consistency: Percentage of years with citations
- peak_momentum: Highest 3-year rolling average citation count
- early_impact: Citations in first 5 years
- recent_momentum: Citations in last 3 years
- acceleration_factor: Ratio of late to early citations

Examples

```
## Not run:
# Assuming groups is output from sniff_groups()
# Calculate cumulative citations
groups_cumulative_citations <- sniff_groups_cumulative_citations(groups, min_citations = 5)
# View results for first group
head(groups_cumulative_citations[[1]])

## End(Not run)
```

`sniff_groups_hubs` *Identify Hub Papers in Research Groups*

Description

This function analyzes citation networks to identify hub papers within research groups based on their citation patterns. It calculates several metrics (Zi, Pi) to classify papers into different hub categories.

Usage

```
sniff_groups_hubs(groups, min_citations = 1)
```

Arguments

`groups`	A list containing network data with the following components: • network: A tidygraph network object • pubs_by_year: Publication counts by year • aggregate: Aggregate network statistics
`min_citations`	Minimum number of citations for a paper to be considered (default: 1)

Details

The function classifies papers into hub categories based on:

- R5: Knowledge hubs (Zi >= 2.5 and Pi <= 0.3)
- R6: Bridging hubs (Zi >= 2.5 and 0.3 < Pi <= 0.75)
- R7: Boundary-spanning hubs (Zi >= 2.5 and Pi > 0.75)

Value

A tibble containing:

- group: Research group identifier
- SR: Paper identifier
- TC: Total citations
- Ki: Total citations from all groups
- ki: Citations from within the same group
- Zi: Standardized within-group citation score
- Pi: Citation diversity index
- zone: Hub classification ("noHub", "R5", "R6", "R7")

Examples

```
## Not run:

# Assuming 'groups' is output from sniff_groups()

# Identify hub papers
hubs <- sniff_groups_hubs(groups, min_citations = 5)

# View results
head(hubs)

## End(Not run)
```

`sniff_groups_keywords` *Extract representative keywords from grouped nodes*

Description

This function processes nodes grouped in a network (typically by community detection), and extracts the most frequent and the most distinctive keywords (using TF-IDF) from a descriptor field such as keywords or subject terms.

Usage

```
sniff_groups_keywords(net_groups, n_terms = 15, min_freq = 1, sep = ";")
```

Arguments

`net_groups`	A list containing a `network` component of class `tbl_graph`, where each node has at least two attributes: `group` and `DE`.
`n_terms`	Integer. The number of top terms to return per group, both by frequency and by TF-IDF. Default is 15.
`min_freq`	Integer. Minimum frequency a term must have in a group to be considered. Default is 2.
`sep`	Character. Separator used in the `DE` field to split multiple terms. Default is `";"`.

Value

A tibble with one row per group, containing two columns:

- `term_freq`: the most frequent terms (with raw frequency).
- `term_tfidf`: the most distinctive terms (with TF-IDF scores).

Examples

```
## Not run:
# Assuming 'groups' is output from sniff_groups()
groups_keywords <- sniff_groups_keywords(groups)

## End(Not run)
```

`sniff_groups_stm_prepare` *Prepare Text Data and Analyze Topic Models*

Description

Processes text data for structural topic modeling and performs topic number selection analysis, returning both the processed data and diagnostic plots.

Usage

```
sniff_groups_stm_prepare(
  groups,
  group_to_stm = "g01",
  search_topics = c(5:40, 45, 50, 55, 60),
  seed = 1234,
  cores = 1
)
```

Arguments

`groups`	A list containing network data with a 'network' component
`group_to_stm`	Character string specifying which research group to process (default: 'g01')
`search_topics`	Numeric vector of topic numbers to evaluate (default: c(5:40, 45, 50, 55, 60))
`seed`	Random seed for reproducibility (default: 1234)
`cores`	Number of CPU cores to use (default: 1)

Value

A list containing:

- result: The searchK results object
- plots: A list containing two ggplot objects (p1: metrics by K, p2: exclusivity vs coherence)
- df_prep: Output from stm::textProcessor
- df_doc: Output from stm::prepDocuments
- df: Original filtered data

Examples

```
## Not run:
output <- sniff_groups_stm_prepare(network_data)
output$plots$p1 # View first plot
output$result # Access search results

## End(Not run)
```

`sniff_groups_stm_run` *Run Structural Topic Modeling Analysis*

Description

Performs structural topic modeling on prepared text data and returns topic proportions and top documents for each topic.

Usage

```
sniff_groups_stm_run(groups_stm_prepare, k_topics = 12, n_top_documents = 50)
```

Arguments

`groups_stm_prepare`	A prepared STM object from `sniff_groups_stm_prepare()`
`k_topics`	Number of topics to model (default: 12)
`n_top_documents`	Number of top documents to each topic (default: 50)

Details

This function:

- Fits an STM model with specified number of topics
- Identifies top terms for each topic
- Calculates topic proportions
- Identifies top documents for each topic

Value

A list containing:

- topic_proportion2: Data frame with topic proportions and top terms
- tab_top_documents: Data frame of top documents for each topic

Examples

```
## Not run:
# Prepare data first
stm_data <- sniff_groups_stm_prepare(network_data)

# Run topic modeling
stm_results <- sniff_groups_stm_run(stm_data, k_topics = 15)

# Access results
stm_results$topic_proportion2  # Topic proportions and terms
stm_results$tab_top_documents  # Top documents per topic

## End(Not run)
```

---

sniff_groups_terms *Extract and Analyze Key Terms from Research Groups*

Description

Identifies and extracts key terms from titles and abstracts of publications within different research groups using natural language processing techniques, and computes term statistics including TF-IDF scores.

Usage

```
sniff_groups_terms(
  net_groups,
  algorithm = "rake",
  phrase_pattern = "(A|N)*N(P+D*(A|N)*N)*",
  model_dir = tempdir(),
  n_cores = 1,
  show_progress = TRUE,
  n_terms = 15,
  min_freq = 2,
  digits = 4
)
```

Arguments

net_groups	A list containing network data with publication information. Must include elements: network (with vertex attributes 'group', 'TI', 'AB'), pubs_by_year, and aggregate.
algorithm	Term extraction algorithm to use. Options are: • "rake" - Rapid Automatic Keyword Extraction (default) • "pointwise" - Pointwise Mutual Information • "phrase" - Phrase pattern matching

phrase_pattern	Regular expression pattern for phrase extraction when algorithm = "phrase" (default: "(A|N)*N(P+D(A|N)*N)*")
model_dir	Directory where UDPipe models are stored (default: tempdir())
n_cores	Number of CPU cores to use for parallel processing (default: 1)
show_progress	Logical indicating whether to show progress bar (default: TRUE)
n_terms	Number of top terms to return in summary table (default: 15)
min_freq	Minimum frequency threshold for terms (default: 2)
digits	Number of decimal places to round numerical values (default: 4)

Details

This function performs the following steps:

1. Validates input structure and parameters
2. Loads the UDPipe language model from the specified directory
3. Processes text data (titles and abstracts) for each group
4. Applies the selected term extraction algorithm (RAKE, PMI, or phrase patterns)
5. Computes term frequencies and TF-IDF scores
6. Returns ranked terms for each research group with comprehensive statistics

The function uses UDPipe for tokenization, lemmatization and POS tagging before term extraction. For phrase extraction, the default pattern finds noun phrases.

Value

A list with two components:

- terms_by_group: A named list (by group) of data frames containing extracted terms with statistics
- terms_table: A summary tibble with top terms by frequency and TF-IDF for each group

Examples

```
## Not run:
# Assuming groups is output from sniff_groups()
terms <- sniff_groups_terms(groups, algorithm = "rake")

# View terms for first group
head(terms$terms_by_group[[1]])

# View summary table
print(terms$terms_table)

# Customized extraction with custom model directory
net_groups_terms <- sniff_groups_terms(net_groups,
  algorithm = "phrase",
  model_dir = tempdir(),
  n_terms = 10,
```

```
  min_freq = 3,
  n_cores = 4
)

## End(Not run)
```

`sniff_groups_trajectories` *Detect Technological Trajectories from Grouped Documents*

Description

This function analyzes the evolution of document groups over time to detect technological trajectories and scientific emergence patterns. It computes similarity measures between groups across time periods and tracks their attributes.

Usage

```
sniff_groups_trajectories(
  groups_cumulative,
  min_group_size = 10,
  top_n_keywords = 3
)
```

Arguments

`groups_cumulative`	A list of cumulative group data over time, typically produced by other functions in the birddog package. Each element should contain network, documents, and groups data.
`min_group_size`	Minimum number of documents required for a group to be considered (default: 10). Smaller groups will be filtered out.
`top_n_keywords`	Number of top keywords to consider when analyzing group characteristics (default: 3).

Value

A list with three components:

- groups_attributes: A list of data frames containing attributes for each tracked group
- groups_similarity: A list of data frames containing Jaccard similarity measures between groups across time periods
- docs_per_group: A data frame containing document IDs for all groups across time periods

Examples

```
## Not run:
# Assuming you have cumulative group data:
trajectories <- sniff_groups_trajectories(groups_cumulative, min_group_size = 15)

## End(Not run)
```

`sniff_key_route` *Identify Key Routes in Citation Networks*

Description

This function identifies and visualizes key citation routes within scientific networks by analyzing the most significant citation paths between publications. The algorithm implements the key-route search from the integrated main path analysis approach described in Liu & Lu (2012).

Usage

```
sniff_key_route(
  network,
  scope = "network",
  n_routes = 1,
  compact_gaps = FALSE,
  direction = "vertical"
)
```

Arguments

`network`	A network object of class `tbl_graph` or `igraph` containing citation data, or a list object generated by `sniff_groups()` when `scope = "groups"`
`scope`	Character string specifying the analysis scope. Must be either "network" (for full network analysis) or "groups" (for group-wise analysis of a grouped network)
`n_routes`	Positive integer specifying the number of key-route starting edges to use (default: 1). Each iteration selects the next-highest SPC edge as a new starting point and extends it into a full path. Higher values reveal auxiliary knowledge diffusion paths and divergence-convergence structures (see Liu & Lu, 2012, Figure 8).
`compact_gaps`	Logical. If TRUE, compresses the vertical axis by removing empty year gaps between publications. This produces shorter plots suitable for academic publications. Default is FALSE (chronological spacing).
`direction`	Character. Plot direction: `"vertical"` (default, top-to-bottom) or `"horizontal"` (left-to-right, oldest on the left).

Details

The function implements the key-route search from Liu & Lu (2012):

1. Computes Search Path Count (SPC) for each citation link using an efficient O(V+E) algorithm based on topological sort. SPC measures how many source-to-sink paths traverse each link.
2. Selects the key-route: the link with the highest SPC value.
3. Searches forward from the end node of the key-route, greedily following the outgoing link with the highest SPC, until a sink is reached.
4. Searches backward from the start node of the key-route, greedily following the incoming link with the highest SPC, until a source is reached.

When `n_routes > 1`, the procedure is repeated: each iteration selects the edge with the next-highest SPC as a new starting point and extends it forward and backward. Routes can share nodes and edges, producing a divergence-convergence-divergence structure that reveals how knowledge flows through multiple sub-streams.

The SPC is computed as `forward[u] * backward[v]` for each edge (u, v), where `forward[u]` counts paths from any source to u and `backward[v]` counts paths from v to any sink (Batagelj, 2003). This guarantees the most significant link is always included in the key-route path.

Value

A list containing for each group:

- `plot` - A ggplot2 object visualizing the key citation route(s)
- `data` - A tibble with publication details (name, TI, AU, PY) of nodes in the key route(s)

References

Liu JS, Lu LYY. An integrated approach for main path analysis: Development of the Hirsch index as an example. Journal of the American Society for Information Science and Technology. 2012;63(3):528-542. doi:10.1002/asi.21692

Batagelj V. Efficient algorithms for citation network analysis. University of Ljubljana, Institute of Mathematics, Physics and Mechanics, Department of Theoretical Computer Science, Preprint Series. 2003;41:897.

Examples

```
## Not run:
# Single key-route (default)
result <- sniff_key_route(my_network, scope = "network")

# Multiple key-routes for richer structure
result <- sniff_key_route(my_network, scope = "network", n_routes = 5)

# Group-wise analysis
grouped_network <- sniff_groups(data)
result <- sniff_key_route(grouped_network, scope = "groups", n_routes = 3)

# Access results for a specific group
```

```
result$group_name$plot
result$group_name$data

## End(Not run)
```

`sniff_network` *Create Citation Networks from Bibliographic Data*

Description

Constructs different types of citation networks from bibliographic data imported from Web of Science or OpenAlex using `birddog's` reading functions.

Usage

```
sniff_network(dataframe, type = "direct citation", external_references = FALSE)
```

Arguments

`dataframe`	A data frame imported via `read_openalex()` or `read_wos()`
`type`	Type of network to create. One of: • "direct citation": Direct citation links between documents • "bibliographic coupling": Documents linked by shared references
`external_references`	Logical indicating whether to include external references (references not in the original dataset) as nodes in the network

Value

A `tbl_graph` object from the tidygraph package representing the citation network. Node attributes include bibliographic information from the input data.

Examples

```
## Not run:
# Using OpenAlex data
oa_data <- read_openalex("works.csv", format = "csv")
net <- sniff_network(oa_data, type = "direct citation")

# Using WoS data
wos_data <- read_wos("savedrecs.txt")
net <- sniff_network(wos_data, type = "bibliographic coupling", external_references = TRUE)

## End(Not run)
```

Index