Package ‘ig.degree.betweenness’

November 11, 2024

Type Package

Title ``Smith-Pittman Community Detection Algorithm for 'igraph' Objects (2024)''

Version 0.1.0

Description Implements the ``Smith-Pittman'' community detection algorithm for network analysis using 'igraph' objects. This algorithm combines node degree and betweenness centrality measures to identify communities within networks, with a gradient evident in social partitioning. The package provides functions for community detection, visualization, and analysis of the resulting community structure. Methods are based on results from Smith, Pittman and Xu (2024) <doi:10.48550/arXiv.2411.01394>.

License MIT + file LICENSE

URL https://github.com/benyamindsmith/ig.degree.betweenness

BugReports https://github.com/benyamindsmith/ig.degree.betweenness/issues

Encoding UTF-8

Imports igraph, igraphdata, rlist, BBmisc, qgraph

RoxygenNote 7.3.2

Suggests knitr, rmarkdown

NeedsCompilation no

Author Benjamin Smith [aut, cre] (<https://orcid.org/0009-0007-2206-0177>), Tyler Pittman [aut] (<https://orcid.org/0000-0002-5013-6980>), Wei Xu [aut] (<https://orcid.org/0000-0002-0257-8856>)

Maintainer Benjamin Smith <benyamin.smith@mail.utoronto.ca>

Repository CRAN

Date/Publication 2024-11-11 12:00:01 UTC

Contents

`cluster_degree_betweenness` *Community structure detection based on node degree centrality and edge betweenness*

Description

Referred to as the "Smith-Pittman" algorithm in Smith et al (2024). This algorithm detects communities by calculating the degree centrality measures of nodes and edge betweenness.

Usage

```
cluster_degree_betweenness(graph)
```

Arguments

`graph`	The graph to analyze

Details

This can be thought of as an alternative version of `igraph::cluster_edge_betweeness()`.

The function iteratively removes edges based on their betweenness centrality and the degree of their adjacent nodes. At each iteration, it identifies the edge with the highest betweenness centrality among those connected to nodes with the highest degree.It then removes that edge and recalculates the modularity of the resulting graph. The process continues until all edges have been assessed or until no further subgraph can be created with the optimal number of communites being chosen based on maximization of modularity.

Value

An igraph "communities" object with detected communities via the Smith-Pittman algorithm.

References

Smith et al (2024) "Centrality in Collaboration: A Novel Algorithm for Social Partitioning Gradients in Community Detection for Multiple Oncology Clinical Trial Enrollments", <doi:10.48550/arXiv.2411.01394>

Examples

```
library(igraphdata)
data("karate")
ndb <- cluster_degree_betweenness(karate)
plot(
ndb,
karate,
main= "Degree-Betweenness Clustering"
)
```

```
ndb

# UNLABELED GRAPH EXAMPLE

data("UKfaculty")
# Making graph undirected so it looks nicer when its plotted
uk_faculty <- prep_unlabeled_graph(UKfaculty) |>
  igraph::as.undirected()

ndb <- cluster_degree_betweenness(uk_faculty)

plot(
  ndb,
  uk_faculty,
  main= "Smith-Pittman Clustering for UK Faculty"
)
```

`plot_simplified_edgeplot` *Plot Simplified Edgeplot*

Description

This function generates a simplified edge plot of an igraph object, optionally highlighting communities if provided.

Usage

```
plot_simplified_edgeplot(graph, communities = NULL, edge.arrow.size = 0.2, ...)
```

Arguments

`graph`	igraph object
`communities`	optional; A communities object
`edge.arrow.size`	edge.arrow size arg. See ?igraph::plot.igraph for more details
`...`	other arguments to be passed to the `plot()` function

Details

This function is ideally for networks with a low number of nodes having varying numbers of connection and self loops. See the example for a better visual understanding.

Value

No return value, called for side effects.

Examples

```
# Load the igraph package
library(igraph)
library(ig.degree.betweenness)
# Set parameters
num_nodes <- 15    # Number of nodes (adjust as needed)
initial_edges <- 1   # Starting edges for preferential attachment

# Create a directed, scale-free network using the Barabási-Albert model
g <- sample_pa(n = num_nodes, m = initial_edges, directed = TRUE)

# Introduce additional edges to high-degree nodes to accentuate popularity differences
num_extra_edges <- 350   # Additional edges to create more popular nodes
set.seed(123)           # For reproducibility

for (i in 1:num_extra_edges) {
  # Sample nodes with probability proportional to their degree (to reinforce popularity)
 from <- sample(V(g), 1, prob = degree(g, mode = "in") + 1)  # +1 to avoid zero probabilities
  to <- sample(V(g), 1)

  # Ensure we don't add the same edge repeatedly unless intended, allowing self-loops
  g <- add_edges(g, c(from, to))
}

# Add self-loops to a subset of nodes
num_self_loops <- 5
for (i in 1:num_self_loops) {
  node <- sample(V(g), 1)
  g <- add_edges(g, c(node, node))
}


g_ <- ig.degree.betweenness::prep_unlabeled_graph(g)

ig.degree.betweenness::plot_simplified_edgeplot(g_,main="Simulated Data")
```

`prep_unlabeled_graph` *Prepared Unlabeled Graph to work with Degree-Betweenness Algorithm*

Description

Presently, `cluster_degree_betweenness()` function only works with labeled graphs. `prep_unlabeled_graph()` is a utility function that gives an unlabeled graph labels which are string values of their vertices.

Usage

```
prep_unlabeled_graph(graph)
```

Arguments

graph	an unlabeled graph.

Value

An "igraph" object with named vertices.

See Also

[cluster_degree_betweenness()] which this function aids.

Examples

```
library(igraph)
library(igraphdata)
library(ig.degree.betweenness)
data("UKfaculty")
# Making graph undirected so it looks nicer when its plotted
uk_faculty <- prep_unlabeled_graph(UKfaculty) |>
  as.undirected()

ndb <- cluster_degree_betweenness(uk_faculty)

plot(
ndb,
uk_faculty,
main= "Node Degree Clustering"
)

ndb
```

Index