dNetFind

March 27, 2017

`dNetFind` *Function to find heuristically maximum scoring subgraph*

Description

`dNetFind` is supposed to find the maximum scoring subgraph from an input graph and scores imposed on its nodes. The input graph and the output subgraph are both of "igraph" or "graphNET" object. The input scores imposed on the nodes in the input graph can be divided into two parts: the positve nodes and the negative nodes. The searching for maximum scoring subgraph is deduced to find the connected subgraph containing the positive nodes as many as possible, but the negative nodes as few as possible. To this end, a heuristic search is used (see Note below).

Usage

```
dNetFind(g, scores)
```

Arguments

`g`	an object of class "igraph" or "graphNEL"
`scores`	a vector of scores. For each element, it must have the name that could be mapped onto the input graph. Also, the names in input "scores" should contain all those in the input graph "g", but the reverse is not necessary

Value

a subgraph with a maximum score, an object of class "igraph" or "graphNEL"

Note

The search procedure is heuristic to find the subgraph with the maximum score:

- i) transform the input graph into a new graph by collapsing connected positive nodes into a meta-node. As such, meta-nodes are isolated to each other but are linked via negative nodes (single-nodes). Clearly, meta-nodes have positive scores, and negative scores for the single-nodes.

- ii) append the weight attribute to the edges in the transformed graph. There are two types of edges: 1) the single-single edge with two single-nodes as two ends, and 2) single-meta edge with a single-node as one end and a meta-node as the other end. The weight for a single-single edge is the absolute sum of the scores in its two-end single-nodes but normalised by their degrees. The weight for a single-meta edge is simply the absolute score in its single-node end normalised by the degree. As such, weights are all non-negative.
- iii) find minimum spanning tree (MST) in the weighted transformed graph using Prim's greedy algorithm. A spanning tree of the weighted graph is a subgraph that is tree and connects all the node together. The MST is a spanning tree with the sum of its edge weights minimised amongst all possible spanning trees.
- iv) find all shortest paths between any pair of meta-nodes in the MST. Within the weighted transformed graph in ii), a subgraph is induced containing nodes (only occuring in these shortest paths) and all edges between them.
- v) within the induced subgraph, identify single-nodes that are direct neighbors of meta-nodes. For each of these single-nodes, also make sure it has the absolute scores no more than the sum of scores in its neighboring meta-nodes. These single-nodes meeting both criteria are called "linkers".
- vi) still within the induced subgraph in v), find the linker graph that contains only linkers and edges between them. Similarly to iii), find MST of the linker graph, called 'linker MST'. Notably, this linker MST serves as the scaffold, which only contains linkers but has meta-nodes being direcly attached to.
- vii) in linker MST plus its attached meta-nodes, find the optimal path that has the sum of scores of its nodes and attached meta-nodes maximised amongest all possible paths. Nodes along this optimal path plus their attached meta-nodes are called 'subgraph nodes'.
- viii) finally, from the input graph extract a subgraph (called 'subgraph') that only contains subgraph nodes and edges betwen them. This subgraph is the maximum scoring subgraph containing the positive nodes as many as possible, but the negative nodes as few as possible.

See Also

dNetFind

Examples

```
# 1) generate an vector consisting of random values from beta distribution
x <- rbeta(1000, shape1=0.5, shape2=1)

# 2) fit a p-value distribution under beta-uniform mixture model
fit <- dBUMfit(x, ntry=1, hist.bum=FALSE, contour.bum=FALSE)

# 3) calculate the scores according to the fitted BUM and fdr=0.01
# using "pdf" method
scores <- dBUMscore(fit, method="pdf", fdr=0.05, scatter.bum=FALSE)
names(scores) <- as.character(1:length(scores))

# 4) generate a random graph according to the ER model
g <- erdos.renyi.game(1000, 1/100)

# 5) produce the induced subgraph only based on the nodes in query
subg <- dNetInduce(g, V(g), knn=0)

# 6) find the subgraph with the maximum score
subgraph <- dNetFind(subg, scores)
```