Fast Pathfinder Network Scaling

Description

Network scaling algorithms such as the Pathfinder algorithm are used to prune many different kinds of networks, including citation networks, random networks, and social networks. However, this algorithm suffers from run time problems for large networks and online processing due to its $O(n^4)$ time complexity. Hence another alternative, Fast Pathfinder algorithm, is used which prunes the original network to get its PFNET(r,n-1) in just $O(n^3)$ time. Since it essentially employs different algorithm than MST based PAthfinder scaling its resultsmight differ from it.

The underlying idea of this approach is based on a classical algorithm in graph theory for shortest path computation - Floyd-Warshall's Shortest Path Algorithm.

Pros & Cons

There is a substantial speed increase due to the changes made in shortest path distance matrix computation. This leads to a reduction in the *time complexity* from $O(n^4)$ to $O(n^3)$. Also the *space complexity* is drastically reduced from (2 * n) - 1 matrices in the original algorithm to 2 matrices.

However, the user cannot control the ***q*** parameter, which constrains the number of indirect proximities examined in generating the network. The q parameter is an integer value between 2 and n-1, inclusive where n is the number of nodes or items. Value of q is fixed to n-1. Also we have found that networks that have a low standard deviation for edge weights or if many of the edge weights are equal to the minimum edge weight, then the network might not be scaled as much as expected. This causes unweighted networks to be not scaled at all.

If the networks being processed are undirected then MST based Pathfinder Network scaling algorithm can be used. This will give results in 30 times faster than Fast pathfinder algorithm. Also it does not face the disadvantages faced by Fast Pathfinder algorithm.

Implementation Details

Once the input file is validated the algorithm is started. It works as follows,

1. Initialize matrices for edge weights & minimum distance costs with dimensions being number of nodes x number of nodes. Initialize their default values as POSITIVE_INFINITY.
2. When reading through the nodes, map node ids to matrix indexes.
3. Next, when reading through the edges set weight & distance matrices with edge weights corresponding to a pair of nodes. Depending upon users choice of how edge weight should be represented, normalize the edge weight i.e. If they chose similarity, normalize edge weights to be dissimilarity based by inverting the edge weight.
4. Now run 3 nested "for" loops with indexes i, j, k such that it returns the shortest possible path from node i to node j using only vertices 1 through k as intermediate points along the way. Now, given this function, our goal is to find the shortest path from each node i to each node j using only nodes 1 through k.
    a. For this, either the true shortest path only uses nodes in the set (1...k); or there exists some path that goes from node i to node k, then from k to j that is better.
    b. Use Minkowski's distance, r parameter, to calculate the distance between the intermediate nodes.
5. Once the distances for all pairs of nodes are calculated, during generating the output file, output only those edges whose distance & weight matrix values are equal.

Usage Hints

The user has to provide 4 inputs; the file storing the information about the network, value of ***r*** parameter, the column name that represents the edge weight and how the edge weight should be considered (Dissimilarity or Similarity). The provided network should not contain edge weights of less than equal to 0 else a warning is generated and default edge weight is assumed for that edge. The value of ***r*** ranges from 1 to infinity. Since *r = infinity* is the most commonly used in network scaling, there is a check-box, which when checked indicates that value of ***r*** is infinity (as default it is checked). The algorithm assumes the edge weights to be a measure of dissimilarity i.e. More the weight, more is the dissimilarity. If the input edge weights are a measure of Similarity i.e. More the weight, less is the similarity then the user can select *Similarity* option from the drop-down box. Only in the case of user selecting *Similarity* the edge weights will be inverted. For example, in a co-authorship network, more number of articles authored by 2 subjects equates to a stronger relationship. In this case the user should select *Similarity* option. The output will be provided as a .NWB file with a pruned network.

Links

- Source Code

Acknowledgments

The original Pathfinder Network Scaling algorithm authored by Roger Schvaneveldt et al. was adapted with a Fast based approach by *Arnaud Quirin et al.* is implemented, integrated and documented by Chintan Tank. Also thanks to Micah Linnemeier and Russell Duhon for providing guidance during implementation. For the description I acknowledge Wikipedia.

References

1. Schvaneveldt, Roger. Pathfinder associative networks : Studies in Knowledge organization. 1990. Link
2. Quirin, Arnaud., Cordon, Oscar., Guerrero-Bote, Vicente., Vargas-Quesada, Benjamin., Moya-Anegnon, Felix. A quick MST-based algorithm to obtain Pathfinder networks ((infinity), n - 1). In *Journal of the American Society for Information Science and Technology*, volume 59, pages 1912-1924, 2008. Link
3. Quirin, Arnaud., Cordon, Oscar., Santamaria, J., Vargas-Quesada, Benjamin., Moya-Anegnon, Felix. A new variant of the Pathfinder algorithm to generate large visual science maps in cubic time. In *Information processing & management*, volume 44, pages 1611-1623, 2008. Link

See Also

The license could not be verified: License Certificate has expired! Generate a Free license now.