Network module detection using recursive local graph sparsification and clustering

Michael Banf

Max Planck Institute for Plant Breeding Research, Cologne, Germany
banf@mpipz.mpg.de

ABSTRACT

Here we present a fast and highly scalable community structure preserving network module detection that recursively integrates graph sparsification and clustering. Our algorithm, called *SparseClust*, participated in the most recent DREAM community challenge on disease module identification, an open competition to comprehensively assess module identification methods across a wide range of biological networks.

Keywords: Graph clustering, Unsupervised structure learning, Network module inference

Introduction

Many community detection algorithms for complex networks have been presented over the last decades. However, determining the quality of individual algorithms, in terms of accuracy and computing time, remains an open issue (Yang *et al*., 2016). In order to obtain optimal clustering performance with respect to both criteria, especially for densely connected and large networks, graph sparsification approaches should be considered as general preprocessing steps to any clustering approach. This leads to the critical problem to efficiently identify edges that are more likely to lie within rather than between clusters. Methodologies for sparsifying graphs include random edge sampling (Karger, 1994, Tumminello *et al*., 2005) or complex network backbone detection (Glattfelder and Battiston, 2009). However, as these approaches are designed to identify most interesting edges, or construct spanning-tree-like network backbones, they are less suited to preserve community structures (Satuluri*et al*., 2011). In addition, methods such as network backbone detection becomes increasingly computationally complex for large network structures (Hamann *et al*., 2016). Other approaches use edge centrality measures, such as edge betweenness centrality (Newman and Girvan, 2004). The betweenness centrality is proportional to the number of shortest paths between any two nodes in the network passing through an edge. As a consequence, edges with a high betweenness centrality can be seen as bottlenecks in the network. Therefore, they are more likely to represent inter-cluster edges, which should be filtered first. However one major disadvantage of the edge betweenness centrality measure is that it becomes prohibitively expensive for larger graphs (Newman and Girvan, 2004).

Here, we present a fast, highly scalable heuristic for iterative local graph sparsification and module detection (see figure 1). The main idea behind our sparsification algorithm is to retain edges that are likely to be part of the same cluster, i.e. intra-cluster edges, and, therefore, to remove those edges that connect modules, i.e. inter-cluster edges. Each sparsification step is followed by a clustering step. The procedure iterates until no more modules larger than a predefined size exist or the sparsification does not change the clustering any further.

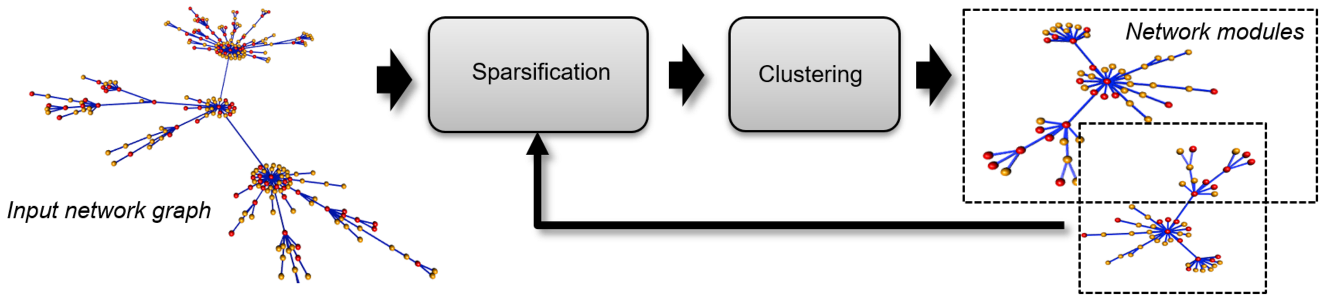

Figure 1. Overview of the Sparseclust algorithm. The input network is repeatedly sparsified and subsequently clustered, until no more sparsification can be performed.

Methods

Our approach recursively iterates graph sparsification and clustering until convergence, i.e. until no more graph sparsification is possible or no more modules of a predefined size exist. To identify intra cluster edges, Satuluri *et al.* (Satuluri *et al.*, 2011) proposed to rank edges using a Jaccard similarity coefficient, based on the assumption that an edge is likely to lie within a module if its adjacent nodes have a high overlap in mutual edges. Here, we use the inverse log-weighted similarity of the edge's adjacent two nodes. It denotes the number of common neighbors of both nodes, weighted by the inverse logarithm of their degrees. We prefer this measure over the Jaccard similarity, as it is based on the assumption that two vertices should be considered more similar if they share a low-degree common neighbor, given that high-degree common neighbors are more likely to have occurred by chance (Adamic, 2003).

The algorithm, as described in pseudo code below, first performs an initial clustering on the full graph G. In addition, a similarity matrix S is formed, computing the inverse log-weighted similarity, over all nodes V. The clustering results are analyzed, retaining all predicted modules within a given size, i.e. a maximum of 100 genes for the DREAM challenge on Disease Module Identification (Choobdar et al., 2018). All larger modules are subsequently sparsified and clustered until convergence. Per large module, the similarity matrix is evaluated, removing all edges in the module specific subgraph, with a similarity coefficient below a given threshold. This threshold is given as a percentage based cutoff with respect to the module specific similarity cumulative density function. This makes our sparsification approach local and adaptive to each individual module. A local similarity threshold is preferred over a global one, given that various clusters may have different densities. As a consequence, a global threshold might therefore retain far many more edges in the denser clusters, disconnecting the less dense clusters (Satuluri *et al.*, 2011). For subsequent module detection, we choose the Infomap algorithm because of its accuracy, scalability and computational complexity (Yang, 2016), although any other clustering algorithm or combinations could be used as well.

```
# Definition of the sparsification and clustering function
sparsify_and_cluster(V, modules, c_module_gene_assignment, G, S, th.cdf = 80, l.old = 0){

 lst_modules <- list() # keep track of found modules

 for(i from 1 to modules.size){

  # set of graph vertices assigned to cluster
  V_i <- V[membership(c_module_gene_assignment) == modules[i]]
  G_i <- G[V_i] # subgraph by module vertices
  S_i <- S[V_i] # jaccard similarity submatrix

  # define local cutoff with respect to local cumulative density function over jaccard similarities
  th_i <- percentile(S_i[S_i > 0], th.cdf)
  S_i[S_i < th_i] <- 0
  S_i[S_i >= th_i] <- 1
  G_i <- G_i * S_i # graph sparsification

  # clustering of G_i
  (modules_i, module_gene_assignment_i) <- infomap_clustering(G_i)

  # count number of genes per cluster
  c_module_gene_assignment_i <- count(module_gene_assignment_i)

  # which modules fullfil competition criteria
  c_module_gene_assignment_to_keep_i <- which(c_module_gene_assignment_i >= 3 & c_module_gene_assignment_i <= 100)

  # keep track of found clusters per iteration
  recursive_clustering <- numeric(V.size)

  for(j from 1 to c_module_gene_assignment_to_keep_i.size){
      V_j <- V[membership(c_module_gene_assignment_to_keep_i) == modules.i[j]]
      recursive_clustering[V_j] <- j
  }

  lst_modules <- append(lst_modules, recursive_clustering)
  module_gene_assignment_to_sparsify_i <- which(c.module_gene_assignment_i >= 100)

  if(module_gene_assignment_to_sparsify.size == 0 | module_gene_assignment_to_sparsify.size == s_old){
   # finished
  }else{
   s_old <-module_gene_assignment_to_sparsify_i.size
   lst_modules_i <- sparsify_and_cluster(V_i , modules_i, module_gene_assignment_to_sparsify_i, G, S, th.cdf = 80, s_old)
   return lst_modules <- append(lst_modules, lst_modules_i)
  }
 }
 return lst_modules
}


# Initial setup and clustering
(modules, module_gene_assignment) <- infomap_clustering(G) # initial clustering of G
V <- V(G) # vertices of Graph G
S <- jaccard_edge_similarity(G) # jaccard similarity matrix

# count number of genes per cluster
c_module_gene_assignment <- count(module_gene_assignment)

# keep modules that meet challenge criteria
c_module_gene_assignment_to_keep <- which(c_module_gene_assignment >= 3 & c_module_gene_assignment <= 100)

final_clustering <- numeric(V.size) # keep track of found clusters per iteration

for(k from 1 to c_module_gene_assignment_to_keep.size){
     Vk <- V[membership(c_module_gene_assignment_to_keep) == modules[k]]
     final_clustering[Vk] <- k
}

c_module_gene_assignment_to_sparsify <- which(c_module_gene_assignment >= 100)

lst_modules <- sparsify_and_cluster(V, modules, c_module_gene_assignment_to_sparsify, G, S, th.cdf = 80, l.old = 0)

# combine function to integrate the modules into one final cluster set
final_clustering <- combine(lst_modules, final_clustering)
```

Discussion

Given the scarcity of graph sparsification approaches in R, in contrast to a plethora of state of the art community detection algorithms, our goal is to build our approach as a one stop solution in R that can do network pre-processing and clustering in one function call. For the DREAM challenge on Disease Module Identification (Choobdar et al., 2018), given several biological networks, we used two different sparsification thresholds, i.e. 80 % for the protein interaction, signal and cancer networks, as well as 95 % for the coexpression and homology networks. These selection were based on visual inspection of the clusters, with respect to the initial network densities and the resulting number and sizes of clusters. Having entered the DREAM competition on Disease Module Identification (Choobdar *et al.*, 2018) only within its last week, we see our approach as a conceptual prototype that focuses on speed as much as accuracy.

References

Choobdar *et al.*, 2018. Choobdar, S., *et al*. (2018) Disease Module Identification in Genomic Networks as an Open Community Challenge, Cell, in submission.

Yang *et al.*, 2016. Yang, Z., Algesheimer, R., and Tessone, C. J. (2016) A Comparative Analysis of Community Detection Algorithms on Artificial Networks. Scientific Reports 6

Karger, 1994. Karger, D. (1994) Random sampling in cut, flow, and network design problems. STOC. 648–657

Glattfelder and Battiston, 2009. Glattfelder J. and Battiston S. (2009) Backbone of complex networks of corporations: The flow of control. Phys. Rev. E, 80(3):36104

Tumminello *et al.*, 2005. Tumminello, M., Aste, T., Di Matteo, T., and Mantegna, R. (2005) A tool for filtering information in complex systems. PNAS, 102(30):10421

Hamann *et al.*, 2016. Hamann, M., Lindner, G., Meyerhenke H., Staudt, C., Wagner, D. (2016) Structure-Preserving Sparsification of Social Networks. Social Network Analysis and Mining. 6:22

Satuluri *et al.*, 2011. Satuluri, V., Parthasarathy, S., and Ruan, Y. (2011) Local graph sparsification for scalable clustering. Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. 721-732

Newman and Girvan, 2004. Newman, M., and Girvan, M. (2004) Finding and evaluating community structure in networks. Phys. Rev. E, 69(2):026113

Lada *et al.*, 2003. Lada A. Adamic and Eytan (2003) Adar: Friends and neighbors on the Web. Social Networks, 25(3):211-230