

Supplementary Information - MONET

MONET: a toolbox integrating top-performing methods for network modularisation

Mattia Tomasoni^{1,2,*}, Sergio Gómez³, Jake Crawford^{4,5}, Weijia Zhang⁶, Sarvenaz Choobdar^{1,2}, Daniel Marbach^{1,2,7} and Sven Bergmann^{1,2,8}

1 Department of Computational Biology, University of Lausanne, Lausanne, Switzerland

2 Swiss Institute of Bioinformatics, Lausanne, Switzerland

3 Dep. d'Enginyeria Informàtica i Matemàtiques, Universitat Rovira i Virgili, Tarragona, Catalonia, Spain

4 Department of Computer Science, Tufts University, MA, USA

5 *Present address*: Graduate Group in Genomics and Computational Biology, Perelman School of Medicine, University of Pennsylvania, Philadelphia, PA, USA

6 School of IT and Mathematical Sciences, University of South Australia, Adelaide, Australia

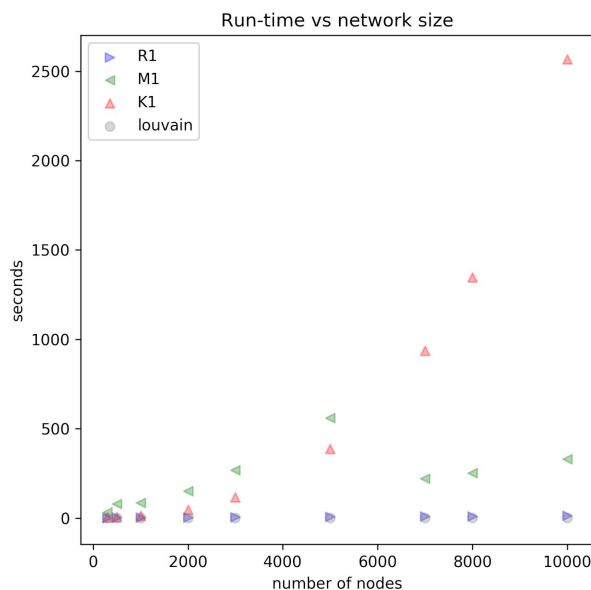
7 *Present address*: Roche Pharma Research and Early Development, Pharmaceutical Sciences, Roche Innovation Center Basel, F. Hoffmann-La Roche Ltd, 4070 Basel, Switzerland

8 Department of Integrative Biomedical Sciences, University of Cape Town, Cape Town, South Africa

* To whom correspondence should be addressed: mattia.tomasoni@unil.ch

Execution time

We enrich the performance evaluation proposed in our paper with a plot of the execution time, in addition to the memory requirements provided in the main text. This plot was generated on a piece of modest commodity hardware desktop running CentOS 6.5 with 8 GB of RAM, a 5GB SWAP partition and 4 2.6 GHz cores. This was done as a proof of concepts. Performance can be expected to increase on faster hardware.



Similarly to Fig. 1, here we propose a comparison of the MONET methods (K1, M1 and R1) against a baseline (Louvain) on simulated graphs with planted community structure. Here the methods are compared based on execution time (y-axis, expressed in seconds) as a function of network size (x-axis, expressed in number of nodes). Each point represents an average of the results obtained performing a grid search over the following parameter space (at least two repetitions for each combination of parameters): number of nodes: 5k, 7k, 8k, 10k; average node degree: 15, 20, 25; exponent of the distribution of community sizes: 1, 2; exponent of the distribution of node degrees: 2, 3.

The reader might have noticed the bump in M1 performance. This needs explaining: the search of the optimal resolution at which to look for modules is quite computational intensive for large networks, thus M1 implements a simple heuristic formula for networks with more than 5000 nodes, which reproduces the resolutions used in the Disease Module Identification DREAM Challenge without the overload of making the search for the optimal resolution. Once the network has been divided in first-level communities at that resolution, M1 applies a new search for resolution to all communities of size ≥ 100 nodes: since these communities are much smaller than 5000 nodes, this is quite efficient. On the contrary, if the initial network is between 4000-5000 nodes, the first finding of the resolution is costly, thus explaining the bump in execution time.

Notes about M1's choice of resolution

The selection of the right resolution for M1 is a difficult task, which requires a full scan of the mesoscale at all scales, from the macroscale (all nodes in one community) to the microscale (every node isolated in its own community), looking for the most stable partitions [Arenas et al. 2008]. A more effective alternative consists in finding a resolution which produces a small set of communities, breaking the network according to these modules, and applying recursively the same method to each of them [Granell et al. 2012], until all communities have sizes below the maximum required, i.e., 100 nodes in the *Disease Module Identification DREAM Challenge* (<https://www.synapse.org/#!/Synapse:syn6156761/wiki/400645>). In MONET, following the strategy for the DREAM challenge, we have adopted an intermediate approach to try to reduce the computing time:


- When the network is very large (50 times the maximum desired size), it selects the resolution using a simple linear formula, which fits the selected higher level resolutions used in the challenge: $\text{resolution} = (\text{num_nodes} - 50 * \text{max_comm_size}) / 100$
- For smaller networks, M1 looks for resolutions that have a largest community containing at most between 20% and 50% of the nodes.
- When a community has a size larger than the desired limit, they are recursively divided using the previous criteria.

Apart from being much faster, this methodology also solves the merge and split problem described in [Lancichinetti & Fortunato, 2011].

The optimization of modularity is performed using a combination of complementary methods, namely: extremal optimization [Duch and Arenas, 2005], spectral optimization [Newman, 2006], fast algorithm [Newman, 2004], louvain algorithm [Blondel et al., 2008], and fine-tuning by iterative reposition of individual nodes. Each one starts with its standard initialization, and then fast, tabu and reposition are also applied to those partitions to try to improve their values of modularity. The overall partition with highest modularity is finally chosen. The community detection tool in Radatools (<http://deim.urv.cat/~sergio.gomez/radatools.php>) has been selected to run all these methods.

References:

- Arenas et al. 2008. Analysis of the structure of complex networks at different resolution levels. *New J. Phys.* 10, 053039
- Granell et al. 2012. Granell, C., Gómez, S., and Arenas, A. (2012). Hierarchical multiresolution method to overcome the resolution limit in complex networks, *Int. J. Bifurc. Chaos* 22, 1250171.
- Lancichinetti & Fortunato, 2011. Limits of modularity maximization in community detection. *Physical Review E*, vol. 84, Issue 6, id. 066122
- Duch and Arenas, 2005. Community detection in complex networks using extremal optimization. *Phys. Rev. E* 72, 027104.
- Newman, 2006. Modularity and community structure in networks. *Proc. Nat. Acad. Sci. USA* 103, 8577.
- Newman, 2004. Fast algorithm for detecting community structure in networks. *Phys. Rev. E* 69, 066133.
- Blondel et al., 2008. Fast unfolding of communities in large networks. *J. Stat. Mech.* (2008) P10008.

 M1: Top method using modularity optimization

Synapse ID: syn7352969

Storage Location: Synapse Storage

M1: Top method using modularity optimization - adjusting resolution in Community Detection Algorithms

Sergio Gómez^{1,2}, Manlio De Domenico¹, Alex Arenas¹¹ Departament d'Enginyeria Informàtica i Matemàtiques, Universitat Rovira i Virgili, Tarragona, Spain.² Contact: S.G. (sergio.gomez@urv.cat)

Abstract

Community detection methods for complex networks usually ignore the finding of the correct scale of resolution. This is crucial in this Disease Module Identification challenge, since communities must contain no more than 100 genes. Therefore, we have used the Multiresolution technique in (Arenas, Fernández and Gómez, 2008) to choose an appropriate scale for two different methods: modularity optimization in subchallenge 1, and a multiscale Multimap approach (the extension of Infomap to multilayer networks, De Domenico et al., 2015) in subchallenge 2.

Contents

- Introduction
- Methods
 - 1. Preprocess
 - 2. Multiresolution community detection
 - 3. Analysis of large communities
 - 4. Postprocess
- Software
- Implementation details
- Conclusions
- References

Introduction

In the absence of information on the cluster sizes of the graph, a method should be able to explore all possible topological scales at which clusters may satisfy the definition of module, to make sure that it will eventually identify the right communities (Arenas, Fernández and Gómez, 2008). *Multiresolution* methods are based on this principle, proposing the screening of the topology at different resolution levels. Moreover, many real graphs display hierarchical cluster structures, with clusters inside other clusters. In these cases, there are more levels of organization of vertices in clusters, and more relevant scales (Granel, Gómez and Arenas, 2012). In principle, algorithms devoted to community detection should be able to identify them. Multiresolution methods can perform this task, as they scan continuously the range of possible cluster scales. The determination of the correct scale of description is reinterpreted as the scale that is more persistent at changes in the resolution parameter. While single-scale methods find a unique partition satisfying the criteria imposed by a quality function or other information maximization techniques, multiresolution based algorithms are able to scrutinize the space of partitions in a deeper way. Eventually, the heterogeneity of the partitions that better classify the data in groups is captured by tuning the topological resolution.

The multiresolution method in (Arenas, Fernández and Gómez, 2008) works by just adding a parameter r , known as *resistance*, to the community detection algorithms. This resistance controls the aversion of nodes to form communities; the larger the resistance, the smaller the size of the modules. For community detection algorithms based on the optimization of the well-known *modularity* function (Newman and Girvan, 2004), this resistance takes the form of a self-loop (with a weight equal to r) which is added to all nodes of the network. In this way, all nodes contribute to the internal strength of their modules with a constant

amount r , which is independent of the rest of the connectivity of the network. In mathematical form, if W is the matrix formed by the weights w_{ij} from node i to node j , the new weights matrix W_r after the addition of the resistance r is $W_r = W + rI$, where I is the identity matrix. Of course, when the resistance is zero, the standard (and implicit) scale of resolution is recovered ($W_0 = W$).

Although this multiresolution methodology was conceived for community detection algorithms based on the optimization of modularity, it can also be applied to other completely different approaches, for example to those based on flow dynamics or random walks. In these cases, the resistance offers the flow (or random walkers) the possibility of remaining in the node, thus enabling the access to new scales of resolution. One of the algorithms which can take advantage of our multiresolution approach is *Infomap* (Rossvall and Bergstrom, 2008), which finds communities by looking for minimum description lengths of the paths of random walkers through the network.

Another important aspect for this challenge is the alignment of the networks in subchallenge 2, which can be modeled as layers of a multilayer system (De Domenico et al., 2013). Here we have used the multiscale *Multimap* approach (the extension of Infomap to multilayer networks, De Domenico et al., 2015) to unravel the mesoscale structure of the overall network. The method encodes the dynamics of random walkers exploring the multilayer network (De Domenico et al., 2014; De Domenico et al., 2016) while minimizing its description length. This algorithm has also been adapted to take advantage of our multiresolution methodology, with the addition of a resistance to all nodes of the multilayer network.

The novelty of the approach is twofold: 1) it relies on the introduction of self-loops that rescale all the topological dimensions involved in the process of capturing the best modular structure of networked data; 2) it makes use of multilayer community detection based on how information flows through the network.

Methods

The methodology applied involves the following main steps:

1. Preprocess and filter the input data, and build the networks
2. Find communities at different resolutions, and choose the appropriate one
3. Extract large communities as new networks, and find subcommunities at an appropriate resolution
4. Process and check the final partitions

1. Preprocess

The goal here is to build weighted networks for the posterior community detection phase, removing the non-significant edges, and normalizing them when required. Network "3_signal_anonym_directed_v3" in subchallenge 1 and the multilayer network in subchallenge 2 are considered and analyzed as directed networks.

Subchallenge 1

The analysis of the input data showed that most of the networks were very dense (average degrees above 100), so a first step was to discard the less significant edges. Thus, we only retained between 10% and 20% of the edges (those with largest weights), with the exception of network "3_signal_anonym_directed_v3" (average degree 4.15), for which all edges were maintained.

Subchallenge 2

Since the ranges of variation of the weights in each layer were different, we first ensured each layer was between 0 and 1 (normalizing those which did not fulfill this requirement), and then the overall 15% of the largest weights were used to form the input multilayer network.

2. Multiresolution community detection

When you directly apply community detection algorithms to our current networks, without caring about the resolution scale, in most of the cases you obtain between 20 and 40 communities of sizes larger than 100, which include several ones with more than 300 nodes, and even some above 500 nodes. This means that almost half of the nodes belong to modules which would be discarded by the challenge size requirements: "Only modules that contain at least 3 genes and at most 100 genes will be used for the scoring (modules outside this range will simply be ignored)". As explained above, we have applied our Multiresolution method (Arenas, Fernández and Gómez, 2008) to look for partitions at more appropriate scales of resolution, capable of delivering most communities below the 100 nodes threshold. However, it is not convenient to use a too large resistance to attain this goal. The problem lays in the other extreme of the modules' sizes: the larger the resistance, the larger the number of nodes in very small communities. As a general rule, small modules are non-important ones, but we want to avoid their proliferation. Therefore, the best solution is to maintain a trade-off between large and small communities, and this can be achieved by maximizing the proportion of nodes inside communities of the desired sizes, i.e. between 3 and 100. Only a few values of the resistance parameter have been checked for each network (between 5 and 10 different values) due to the time cost of each community detection step, but that has been enough to find much better resolutions than the default one ($r = 0$).

Subchallenge 1

We have found the partitions of communities by optimizing modularity (Newman and Girvan, 2004). In fact, specific versions of modularity for weighted (Newman, 2004a) and directed (Arenas et al., 2007) networks are needed to avoid losing this valuable information. Our optimization of modularity has involved the use of a "cocktail" of community detection algorithms. The idea is that a combination of several algorithms has less chances to get stacked in a bad partition, and the quality of the final partition is much improved. Specifically, they are:

- Extremal optimization (Duch and Arenas, 2005)
- Spectral optimization (Newman, 2006)
- Fast algorithm (Newman, 2004b)
- Fine-tuning by iterative reposition of individual nodes

Subchallenge 2

The selected community detection algorithm is Multimap (De Domenico et al., 2015), the extension of Infomap (Rossvall and Bergstrom, 2008) to multilayer networks.

3. Analysis of large communities

The partitions obtained with the previous multiresolution analysis may contain several communities with more than 100 nodes. To avoid them being ignored by the scoring process of the challenge, we extracted those communities from the full graph, creating new reduced networks, and applied them a second level of multiresolution analysis. This procedure is similar to the fully hierarchical method in (Granell, Gómez and Arenas, 2012).

Subchallenge 1

In this case, we replaced the Extremal and Spectral methods of the "cocktail" of modularity optimization algorithms with the Tabu search (Arenas, Fernández and Gómez, 2008). The advantage of Tabu search is its capacity to obtain partitions with a much better modularity than the Extremal and Spectral ones. Unfortunately, Tabu is much slower than Extremal and Spectral, thus it was not possible to apply it to the full networks in the previous phase.

Subchallenge 2

No further second level analysis was performed with the multilayer network.

4. Postprocess

Despite the previous phase, some communities with more than 100 nodes still remained, due to either their appearance in the second level of optimization, or because their size was only a small amount above 100. In these cases, a random split in modules of 100 nodes (plus a remainder module) was performed to ensure their consideration in the scoring process. It must be emphasized that the restriction of 100 nodes is rather arbitrary, thus there appear communities which cannot fulfill this requirement in any natural way, tending to survive during the whole analysis. Finally, the first and second level communities were combined to form the definitive partitions, checking their consistency and being formatted in the required way.

Software

The software we have used is the following:

- Radalib (<http://deim.urv.cat/~sergio.gomez/radalib.php>): This is an "Ada library and tools for the analysis of Complex Networks and more". It contains, among many others, our programs to perform the Multiresolution community detection by the optimization of modularity. The software is freely available both in this main site and in GitHub (<https://github.com/sergio-gomez/Radalib>). Executables of the tools for the main platforms are also available in Radatools (<http://deim.urv.cat/~sergio.gomez/radatools.php>). The tools we have used are the following:
 - *Communities_Detection*: implements the modularity optimization algorithms (Tabu, Extremal, Spectral, Fast and Reposition), with resistance parameter, and weighted and directed modularity.
 - *List_To_Net*: converts a list of edges into a network in Pajek format.
 - *Network_Properties*: calculates many properties of the networks.
 - *Extract_Subgraphs*: extracts communities from a partition as individual networks.
 - *Reformat_Partitions*: to write partitions substituting indices of nodes by names.
- Infomap at MapEquation.org (<http://www.mapequation.org/>): implements Multimap, the extension of Infomap to multilayer networks, as an option within the Infomap program. It is freely available both in this main site and in GitHub (<https://github.com/mapequation/infomap>).
- A simple python script for the postprocess phase described above.

Implementation details

Here we describe some of the details needed to reproduce our results as submitted to this challenge. Of course, the stochastic nature of the optimization algorithms means that new executions could lead to slightly different results, but they serve as a guide.

Subchallenge 1

In the next table we summarize the main information corresponding to the analysis of the networks. During the preprocess phase, all edges with a weight below the pruning threshold are discarded. The multiresolution community detection makes use of an `ersrf` optimization (e: extremal; s: spectral; r: reposition; f: fast), with the finally selected resistances shown in the Table. Then, the indicated number of largest communities are extracted as new networks for the 2nd level multiresolution community detection, a `trfr` optimization scheme is applied (t: tabu), and the corresponding selected resistances are shown in the last column (sorted by decreasing number of nodes). Note that some of the resistance values are negative, in order to increase the size of the new modules with respect to the ones at default ($r = 0$) resolution.

Network	Pruning threshold	1st level resistance	Number of 2nd level networks	2nd level resistances
1_ppi_anonym_v2	0.60	200.0	11	70, 25, 40, 10, 0, 10, 5, 5, -5, -5, -5
2_ppi_anonym_v2	0.60	400.0	3	0, -10, 0
3_signal_anonym_directed_v3	0.00	100.0	2	-10, -10
4_coexpr_anonym_v2	0.30	80.0	8	0, 0, -2, 0, 0, 0, 0, 0
5_cancer_anonym_v2	0.45	100.0	3	50, 10, 10
6_homology_anonym_v2	15.00	10000.0	3	500, 750, -5

Subchallenge 2

In this case, after the preprocessing step explained above which prunes 85% of the edges, the multilayer Infomap is run with the parameters in the next Table (the others being the default ones). Several values of the resistance parameter (self-loops) are tried until an appropriate scale is obtained, at $r = 2.0$.

Parameter	Value
Employ multiplex random walkers	<code>-i multiplex</code>
The network is directed	<code>--directed</code>
Number of independent tries	<code>-N 20</code>
Output the cluster file	<code>--clu</code>
Use specific value of the relax rate	<code>--multiplex-relax-rate 0.15</code>
Optimization method	<code>-2 --two-level</code>
Obtain communities of physical nodes (not of state nodes)	<code>--hard-partitions</code>

Conclusions

We have developed a strategy for disease module identification which, using different algorithms for community detection, shows the importance of adjusting the resolution of the study. Multiresolution analysis should always be a central part whenever you are interested in the mesoscale of a system, since many methods define (usually in an implicit form) a certain scale, which could be inappropriate for your needs. We have also shown how this multiresolution can be incorporated in two general classes of community detection algorithms: modularity optimization for single layer networks, and random walk flows in multilayer networks. Other multiresolution approaches exist in the literature, but the important message here is that the users must be aware of their central role, and use this knowledge to improve the outcome in all their practical applications.

Finally, it must be stated that we did not participated in the previous rounds of the challenge (we were invited to participate too late), which would have been an important source of information to improve our set-up and to automate the different phases of the whole process.

References

- Arenas, A., Duch, J., Fernández, A., and Gómez, S. (2007). Size reduction of complex networks preserving modularity. *New J. Phys.* 9, 176.
- Arenas, A., Fernández, A., and Gómez, S. (2008). Analysis of the structure of complex networks at different resolution levels. *New J. Phys.* 10, 053039.
- De Domenico, M. et al. (2013). Mathematical Formulation of Multi-Layer Networks. *Phys. Rev. X* 3, 041022.
- De Domenico, M., Solé-Ribalta, A., Gómez, S., and Arenas, A. (2014). Navigability of interconnected networks under random failures. *Proc. Nat. Acad. Sci. USA* 11, 8351.
- De Domenico, M., Lancichinetti, A., Arenas, A., and Rosvall, M. (2015). Identifying modular flows on multilayer networks reveals highly overlapping organization in social systems. *Phys. Rev. X* 5, 011027.
- De Domenico, M., Granell, C., Porter, M.A., and Arenas, A. (2016). The physics of spreading processes in multilayer networks. *Nature Phys.* 12, 901.
- Duch, J., and Arenas, A. (2005). Community detection in complex networks using extremal optimization. *Phys. Rev. E* 72, 027104.


- Granell, C., Gómez, S., and Arenas, A. (2012). Hierarchical multiresolution method to overcome the resolution limit in complex networks, *Int. J. Bifurc. Chaos* 22, 1250171.
- Newman, M.E.J., and Girvan, M. (2004). Finding and evaluating community structure in networks. *Phys. Rev. E* 69, 026113.
- Newman, M.E.J. (2004a). Analysis of weighted networks. *Phys. Rev. E* 70, 056131.
- Newman, M.E.J. (2004b). Fast algorithm for detecting community structure in networks. *Phys. Rev. E* 69, 066133.
- Newman, M.E.J. (2006). Modularity and community structure in networks. *Proc. Nat. Acad. Sci. USA* 103, 8577.
- Rosvall, M., and Bergstrom, C.T. (2008). Maps of random walks on complex networks reveal community structure. *Proc. Nat. Acad. Sci. USA* 105, 1118.

Author Contributions

S.G., M.D.D. and A.A. conceived the study. S.G. performed the study of subchallenge 1. M.D.D. performed the study of subchallenge 2. S.G. refined and prepared the final submission. S.G., M.D.D. and A.A. wrote the manuscript.

Acknowledgements

A.A. and S.G. acknowledge MULTIPLEX (grant number 317532) of the European Commission, and the Spanish Ministerio de Economía y Competitividad (grant number FIS2015-71582-C2-1). A.A. also acknowledges ICREA Academia and the James S. McDonnell Foundation. M.D.D. acknowledges financial support from the Spanish program Juan de la Cierva (IJCI-2014-20225).

 K1: Top method using kernel clustering

Synapse ID: syn7349492

Storage Location: Synapse Storage

K1: Top method using kernel clustering - a double spectral approach

Jake Crawford^{1,3}, Junyuan Lin², Xiaozhe Hu², Benjamin Hescott¹, Donna Slonim¹, Lenore Cowen^{1,3}¹ Department of Computer Science, Tufts University, Medford, MA 02155² Department of Mathematics, Tufts University, Medford, MA 02155³ Corresponding authors: john.crawford@tufts.edu (mailto:john.crawford@tufts.edu) and cowen@cs.tufts.edu (mailto:cowen@cs.tufts.edu)

Abstract

We use the "Diffusion State Distance" spectral graph metric of Cowen and Hescott et al. to define a matrix of pairwise dissimilarities between all nodes of the network. We then run an off-the-shelf spectral clustering algorithm on the matrix of these distances, using the training rounds to set the target number of clusters the spectral clustering should return. In half the networks for subchallenge 1, we also found a few additional clusters by looking for dense bipartite subgraph structure.

Introduction

The heart of our method is the "Diffusion State Distance" metric defined in Cao et al. in 2013¹ and generalized to deal with confidence weights on the edges in 2014².

Our general strategy was as follows: 1) Compute the DSD matrix 2) Use this as input to a generic clustering method (in the case of our final submission, spectral clustering³).

For networks 2, 3, and 5, we also constructed several additional clusters using an entirely different method: this was a search for dense bipartite subgraphs. Dense bipartite subgraphs were found to be signatures of possible instances of the "Between Pathway Model"³ type of functional modules in networks whose edges indicated *negative genetic interactions*⁴. We thought that might be the case for network 2. We searched for these dense bipartite subgraphs on networks 3 and 5 as well, without real theoretical justification, but it appeared to help results in the training rounds on these networks as well. The method we used to search for these is essentially the algorithm in⁴: the implementation we used was the Genecentric software package⁵ with the network passed to the algorithm with all edge weights set to -1. We explain how we merged these clusters with the spectral clusters below.

Some Intuition about DSD

PPI networks are known to be "small world" networks in the sense that they are small-diameter, and most nodes are close to all other nodes. Thus any method that infers similarity based on proximity will find that a large fraction of the network is proximate to any typical node. In fact, this issue has already been termed the "ties in proximity" problem in the computational biology literature⁶.

Cowen and Hescott et al.^{1,2} argued that the typical shortest-path measure of proximity missed a lot of informative structure that was encoded in the network: not all short paths gave equal indications of similarity; paths through low-degree nodes were stronger indications of functional similarity than paths that went through high-degree nodes, or hubs. For intuition, we could take an analogy with social networks. If two people who do not know each other, have several Facebook mutual "friends" each with a low to moderate number of total number of friends themselves, the claim is that this is a stronger vote that they might have something in common than if they are both Facebook "friends" with a "famous" node with millions of friends such as Oprah Winfrey. Because everyone is Facebook friends with Oprah, a length-two path through Oprah is a much weaker indication of similarity than a length-two path through a low-degree node would be. Going back to biological networks, we find a similar phenomenon, where hub nodes tend to be involved in the general machinery of the cell, interacting with many proteins that are not functionally related. Proteins that are both connected to such a high-degree hub are less likely to have similar function, than proteins that both interact with a protein of much lower degree.

In order to come up with a more fine-grained measure of similarity that downweights hubs, in ¹ we defined a new spectral graph metric called Diffusion State Distance, or DSD. Consider the undirected graph $G(V, E)$ on the vertex set $V = \{v_1, v_2, v_3, \dots, v_n\}$ and $|V| = n$. Define $He^{[k]}(A, B)$ as the expected number of times that a simple symmetric random walk starting at node A , and proceeding for k steps, will visit node B . In what follows, assume k is fixed, and when there is no ambiguity in the value of k , we will denote $He^{[k]}(A, B)$ by $He(A, B)$. Notice that $He(A, B)$ begins to get at the notion of the relative importance of paths through low-degree rather than high-degree nodes; if A and B are connected through a low-degree neighbor, the random walk starting at A is likely to reach B ; if however, a neighbor that connects them is of very high degree, the walk through that neighbor is likely to diffuse away and reaches B with much lower probability. However, we are not done, because $He(A, B)$ is not a metric; in particular, it is not even symmetric (for example, if A is at the center of a star graph with many petals, and B is at one of the petals, a walk started at B will always reach A , whereas a walk started at A is unlikely to reach B).

Thus we further define a n -dimensional vector $He(v_i), \forall v_i \in V$, where

$$He(v_i) = (He(v_i, v_1), He(v_i, v_2), \dots, He(v_i, v_n)).$$

This vector can be thought of as the global view of the $He(A, B)$ measure from each vertex to all the other vertices of the network.

Then, the Diffusion State Distance (DSD) between two vertices u and v , $\forall u, v \in V$ is defined as:

$$DSD(u, v) = \|He(u) - He(v)\|_1,$$

where $\|He(u) - He(v)\|_1$ denotes the L_1 norm of the He vectors of u and v .

¹ showed for any fixed k , that DSD is a metric, namely that it is symmetric, positive definite, and non-zero whenever $u \neq v$, and it obeys the triangle inequality. Thus, one can use DSD to reason about distances in a network in a sound manner. Further, when the network is ergodic, DSD converges as the k in $He^{[k]}(A, B)$ goes to infinity, allowing us to define DSD independent from the value k , and to compute the converged DSD matrix tractably, with an eigenvalue computation.

Some Intuition about Genecentric

A *maximal bipartite subgraph* of $G(L)$ is a partition of the vertex set of $G(L)$ into two disjoint subsets of vertices A and B as follows. Let $|C|$ denote the number of edges with one endpoint in A and one endpoint in B . Now, for any vertex $v \in A$, define A' to be $A - v$ and B' to be $B \cup v$. Then (A, B) is maximal in $G(L)$ implies that the number of edges of $G(L)$ that have one endpoint in A' and one endpoint in B' is at most $|C|$. And similarly for $v \in B$. In other words, moving a single vertex from A to B or vice versa cannot increase the number of edges that go across from A to B .

The randomized algorithm of Brady et al ⁴ searches for *stable* bipartite subgraphs. These are bipartite subgraphs that appear in a large proportion of the *maximal bipartite subgraphs* of G . They tend to be small, dense bipartite subgraphs. When the Genecentric software package ⁵ is passed a network where edges are assigned constant negative edge weight, it implements precisely the algorithm of Brady et al ⁴.

Dense bipartite substructure is likely to have meaning in *genetic* interaction networks, because it is likely to represent redundant functional pathways ^{3,4}. We also find such structure in networks 3 and 5, where it might also have functional meaning.

Methods

Computing the DSD Matrix

The DSD matrix can be computed for any weighted network using the "cDSD" method from software available at: <http://dsd.cs.tufts.edu/capdsd> (<http://dsd.cs.tufts.edu/capdsd>).

Converged cDSD can be computed in time proportional to inverting a matrix; it is completely feasible to compute it directly on networks of the size of this DREAM challenge, even on ordinary desktop computers. Furthermore, it only has to be done once, for each of the six networks. However, because we were running on ordinary desktop computers, we decided to instead produce very close approximations for the cDSD matrix instead of computing it exactly in one of two ways. 1) Either running cDSD with a 7-step random walk only (i.e. setting $k = 7$ instead of $k = \infty$; note that when k is set greater than the diameter of the network, cDSD converges very quickly, so $k = 7$ presents a good approximation on the DREAM challenge networks) or 2) a new method to apply algebraic multigrid (AMG) methods to speed up the computation of the discrete Green's function. We describe 2) in more detail in the next section.

For Subchallenge 1, we used the AMG methods to produce the approximate cDSD distance matrices for networks 1, 2, 4, 5, and 6. For network 3 (the directed signaling network), we wanted to keep the edge directions for subchallenge 1. However, the edge directions could have caused the network to be (weakly) disconnected, which would result in some nodes having infinite cDSD distance. So, we kept the directions of the original edges and added low-weight backedges, having $\frac{1}{100}$ of the minimum edge weight in the network, to prevent the network from being disconnected. We then ran the original version of cDSD, with a random walk of length 7, on the resulting largest connected component.

On subchallenge 2, we could have used the faster AMG methods on our combined networks (see below) as well, but we ended up using the original version of cDSD with a random walk of length 7 instead, because it was easier than coordinating between different members of the team.

AMG Methods: Overview

Consider a connected graph $G = (V, E)$. The Diffusion State Distance (DSD) between two nodes u and v can also be written as:

$$\text{DSD}(u, v) = \|(\mathbf{b}_u^T - \mathbf{b}_v^T)(\mathbf{I} - \mathbf{P} + \mathbf{W})^{-1}\|_1$$

where \mathbf{I} is the identity matrix, \mathbf{P} is the transition matrix, and \mathbf{W} is a matrix whose each row is equal to the transpose of the steady state distribution π , i.e., $\mathbf{W} = \mathbf{e}\pi^T$, where \mathbf{e} is the constant vector of all ones. Since we need to compute the distances between all pairs of nodes, the most time expensive part is computing the matrix inverse in the equation above, which is the discrete Green's function of the graph G , i.e.,

$$\mathbb{G} = (\mathbf{I} - \mathbf{P} + \mathbf{W})^{-1}.$$

Our efficient algorithm for computing the discrete Green's function is based on the nearly-optimal solver for the graph Laplacian. To use this, we need to rewrite the discrete Green's function a little bit. Let \mathbf{A} be the adjacency matrix and \mathbf{D} be the diagonal degree matrix. The graph Laplacian is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ and the normalized graph Laplacian can be obtained as $\mathbf{N} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$. It is well-known that the eigenvector corresponding to the zero eigenvalue of \mathbf{N} is given by $\mathbf{d} := \frac{1}{\sqrt{d_{total}}}\mathbf{D}^{-\frac{1}{2}}\mathbf{e}$. Using those definition and some algebraic calculations, we arrive at

undefined

where \mathbf{N}^\dagger is the pseudo-inverse of \mathbf{N} which can be computed by fast linear solvers. In our case, we use the algebraic multigrid (AMG) method to compute it. The following describes our algorithm:

Algorithm 1 Compute $\mathbb{G} = (\mathbf{I} - \mathbf{P}^T + \pi\mathbf{e}^T)^{-1}$

- 1: **for** $i = 1 : N$ **do**
 - 2: Compute $\mathbf{b} = \mathbf{D}^{-\frac{1}{2}}\mathbf{e}_i$ where \mathbf{e}_i is the i -th column of the identity matrix
 - 3: $\mathbf{b}_2 \leftarrow (\mathbf{d}^T\mathbf{b})\mathbf{d}$ and $\mathbf{b}_1 \leftarrow \mathbf{b} - \mathbf{b}_2$
 - 4: Compute \mathbf{x}_1 by solving $\mathbf{N}\mathbf{x}_1 = \mathbf{b}_1$ by the AMG method
 - 5: $\mathbb{G}(:, i) \leftarrow \mathbf{D}^{\frac{1}{2}}(\mathbf{x}_1 + \mathbf{b}_2)$
 - 6: **end for**
-

Since we use the AMG method in our algorithm, which has computational complexity $\mathcal{O}(N \log N)$, $N = |V|$, the overall computational complexity of our algorithm is $\mathcal{O}(N^2 \log N)$. Note that, if the standard direct method is used, the overall computational cost is $\mathcal{O}(N^4)$ and the best computational cost we can expect for computing the inverse of a matrix is $\mathcal{O}(N^2)$. Therefore, our approach is nearly optimal.

CPU time for computing the distance of one pair of nodes, in seconds (Sub-challenge 1 Networks):

Network	Edges	Nodes	Old method (C)	New method (Matlab)	Speedup
Network 1	2,232,405	17,397	17.959	1.596	11.25
Network 2	397,309	12,420	7.558	0.277	33.30
Network 3	21,826	5,254	0.953	0.048	19.85
Network 4	1,000,000	12,588	7.713	0.524	14.20
Network 5	1,000,000	14,679	11.060	0.687	16.10
Network 6	4,223,606	10,405	5.233	3.068	1.71

The table above presents the amortized cost for computing the distance between one pair of nodes (to estimate an upper bound on the resulting total time to compute the entire matrix, multiply by the square of the number of nodes in the network). This is actual CPU time for each of the networks from subchallenge 1, where the CPU we used for the tests was an Intel Xeon CPU E5-2637 v3 @ 3.50GHz. One can see that our new method using the AMG algorithm can achieve about 16X speed up on average comparing with old method using direct solvers, even though it is implemented in the slower Matlab language. We can expect a larger performance improvement when we compute the whole distance matrix of the network.

Edge Weights

For subchallenge 1 and our final submission to subchallenge 2, we passed all of the edge weights unchanged to our cDSD algorithm. In each case, if the network was disconnected we ran cDSD on each connected component individually, and used the largest for clustering.

For subchallenge 2, we tried different ways of rescaling the edge weights on networks 3 and 6 in order to combine them with the other networks, since the edge weights in networks 3 and 6 appeared to be on a different scale. However, we ended up not including any information from networks 3 and 6 in our final submission to subchallenge 2.

Clustering Method

Originally, we tried applying two different clustering methods to the resulting cDSD matrix: edge removal (similar to the Girvan-Newman algorithm⁷, using DSD distance rather than edge betweenness to choose edges to remove), and spectral clustering⁸.

In the training rounds, we observed that spectral clustering performed much better across all of the input networks. We used the spectral clustering package provided in scikit-learn⁹, documented at <http://scikit-learn.org/stable/modules/clustering.html> (<http://scikit-learn.org/stable/modules/clustering.html>).

Note that the spectral clustering algorithm we used operates on a similarity matrix (i.e. entries that are most alike have higher values in the matrix, and entries that are more disparate have lower values in the matrix). However, the cDSD matrix is a distance matrix; that is, similar entries have low cDSD values, and vice-versa. To convert the cDSD matrix to a similarity matrix, we applied the RBF kernel to each entry in the cDSD matrix, which maps low distances to high similarity scores and vice-versa. We then clustered the similarity matrix.

Since the spectral clustering algorithm provided by scikit-learn uses k -means as the underlying clustering mechanism, it takes a parameter k specifying the number of cluster centers. In the training rounds, we varied k to change the number of clusters that were output.

We tried several different values of k for each network ($k = 100, 200, 500, 800, 1000, \text{ and } 1200$ for each network). Large clusters ($k = 100$) appeared to perform the best on networks 3, 4, and 6, and smaller clusters ($k = 1000$ and $k = 1200$) performed the best on network 1. Networks 2 and 5 performed the best when large clusters were combined with smaller Genecentric clusters, as described below.

Also note that spectral clustering will produce clusters of size less than 3, and clusters of size more than 100. Whenever we produced a cluster of size less than 3, we ignored those vertices and did not include them in a cluster. Whenever we produced a cluster of size more than 100, however, we recursively called spectral clustering again, with a cluster size of 2, and continued the recursion until all clusters were of size < 100 . We compared this method with submissions in which large clusters were thrown out. With only one exception (network 3 with cluster size 200), splitting clusters resulted in performance that was unchanged or improved, usually improved.

Merging the Genecentric and Spectral Clustering Sets

On networks 2, 3, and 5, we also ran Genecentric with all the edge weights in the network set to -1. In the interest of running Genecentric reasonably quickly, we split the graph into large spectral clusters (using $k = 10$ and $k = 20$ for networks 2 and 5) based on cDSD as before, and ran Genecentric on the resulting clusters. Network 3 was small enough to run Genecentric on directly, without splitting it into spectral clusters first.

We then ran Genecentric using a minimum partition size of 3 and a maximum partition size of 100, to generate clusters that fit within the size limits for the challenge, and edge squaring, as is recommended in the Genecentric documentation, to speed up the computation slightly. Otherwise, we used all of the default parameters.

Genecentric outputs clusters (which it calls modules) split into precisely two individual subclusters (which it calls pathways). We considered each individual pathway to be a separate Genecentric cluster, unless either of the partitions was smaller than 5 nodes. In that case, we combined the partitions into one large cluster. Note that Genecentric only puts a small percentage of the network into clusters; most of the network remains unclustered by Genecentric. Furthermore, Genecentric may return an overlapping set of clusters, so we order the Genecentric clusters by size and greedily choose a collection of non-overlapping clusters (with preference for the larger clusters). These now define our Genecentric cluster set.

We then had two sets of clusters: spectral clustering clusters and Genecentric clusters, and we had to merge them into a single set of non-overlapping clusters. To merge Genecentric clusters with spectral clusters, we tried 2 approaches. In the first, we preferentially selected Genecentric clusters when we merged them in a greedy approach; i.e. we simply removed all of the spectral clusters that overlapped at all with any Genecentric cluster. This meant that some (potentially informative) clustering information was thrown out, but in practice adding the Genecentric clusters and removing the corresponding spectral clusters seemed to help our scores overall on networks 2, 3, and 5.

We also tried a second approach for merging Genecentric clusters based on their overlap with spectral clusters. Consider a modified Genecentric clustering that places all nodes that are not placed in an original Genecentric cluster into a new, single additional large cluster. Now return clusters of nodes that are placed in the same cluster by *both* the spectral and Genecentric clustering. In practice, this overlap method seemed to result in more informative final clusters on networks 2 and 5 during the training rounds, as long as the starting clusters are sufficiently large to allow for a high degree of overlap.

We tried this overlap strategy on network 3 as well, but it did not show appreciable improvement over the greedy merge strategy described above, at any cluster size we tried. In our final submission, we used the greedy merge approach for network 3, and the overlap approach for networks 2 and 5.

Subchallenge 2

We tried a variety of ways to merge our clusters on networks 1-6 so that the collection was non-overlapping: nothing we tried gave us a combined clustering with more than 10 enriched clusters.

Therefore, we instead created a *combined* network where we took the union of all the edges from a subcollection of networks 1-6, (with network 3 treated as

undirected and networks 3 and 6 reweighted as follows: all edges on network 3 were given a constant weight of 0.95, and low-weight edges on network 6 were thrown out, with the remaining edge weights normalized between 0 and 1).

We then performed exactly the same spectral clustering method as we did for subchallenge 1 on the combined network. We did not add any Genecentric clusters. We computed cDSD on the combined network, and then performed spectral clustering, recursively splitting all clusters of size > 100 as before.

In the training rounds, we tried including different subsets of networks 1-6. We tried including all 6 networks, we tried including networks 1-4, we tried networks 1-3 and 1-2, and in the end, decided to include edges from network 1, 2, and 4 only. At a k -value of 500, this gave the best performance across all FDR values during the training rounds.

Discussion

We showed that our double spectral approach was able to produce seemingly meaningful clusters for both subchallenges in the training rounds, in a fairly unsupervised fashion; training mostly on the final number of clusters the spectral clustering method was asked to output. One additional aspect we would have definitely liked to train on, but really didn't have sufficient submissions to figure out, was whether we should be changing the edge weights. We made one submission for network 1 where we replaced the existing edge weights with a bimodal distribution as follows: edges below a confidence of 0.2 were removed, edges below 0.6 kept their original weight, and edges above 0.6 got a weight of 1. This approach performed better than the original edge weights at $k = 500$, but we did not have sufficient training rounds to optimize both the edge weighting scheme and the value of k simultaneously, so we kept the original weights in our final submission.

We feel that more extensive training data would allow us to adjust the confidence weights on the various networks to make them more informative, or to adjust the method by which we combined the networks for Subchallenge 2 to get the most information out of the union of the edges in the networks.

Source Code

Source code has been submitted through the Synapse platform and can be found at <https://www.synapse.org/#!/Synapse:syn7349492/files/> (<https://www.synapse.org/#!/Synapse:syn7349492/files/>). This includes both the Python code to compute using the cDSD method, and Matlab code for the AMG method. If it is easier, pre-computed DSD matrices can also be obtained from the authors by request.

Author Contributions


Overall strategy and design: JC BH DS LC. Computational DSD speedup: XH JL. Performed the Experiments and Wrote the Code: JC. Analyzed the data: JC LC. Wrote the paper: JC XH LC.

References

- ¹ Cao, M., Zhang, H., Park, J., Daniels, N. M., Crovella, M. E., Cowen, L. J. and Hescott, B. (2013). Going the distance for protein function prediction. PLOS One 8, e76339.
- ² Cao, M., Pietras, C. M., Feng, X., Doroschak, K. J., Schaffner, T., Park, J., Zhang, H., Cowen, L. J. and Hescott, B. (2014). New directions for diffusion-based prediction of protein function: incorporating pathways with confidence. Bioinformatics 30, i219–i227.
- ³ Kelley, R. and Ideker, T. (2005). Systematic interpretation of genetic interactions using protein networks. Nature Biotechnology 23, 561–566.
- ⁴ Brady, A., Maxwell, K., Daniels, N. and Cowen, L. J. (2009). Fault tolerance in protein interaction networks: stable bipartite subgraphs and redundant pathways. PloS one 4, e5364.
- ⁵ Gallant, A., Leiserson, M. D., Kachalov, M., Cowen, L. J. and Hescott, B. J. (2013). Genecentric: a package to uncover graph-theoretic structure in high-throughput epistasis data. BMC bioinformatics 14, 1.
- ⁶ Arnau, V., Mars, S. and Marin, I. (2005). Iterative cluster analysis of protein interaction data. Bioinformatics 31, 364–378.
- ⁷ Girvan, M. and Newman, M. E. (2002). Community structure in social and biological networks. Proceedings of the national academy of sciences 99, 7821–7826.
- ⁸ Ng, A. Y., Jordan, M. I., Weiss, Y. et al. (2002). On spectral clustering: Analysis and an algorithm. Advances in neural information processing systems 2, 849–856.
- ⁹ Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.

Wiki created on 10/07/2016 5:51 PM and last modified on 10/08/2016 4:09 PM

 Wiki Revision History

 R1: Top method using random walk

Synapse ID: syn7286597

Storage Location: Synapse Storage

R1: Top method using random walk - balanced multi-layer regularized Markov clustering algorithm

Weijia Zhang^{1,2}, Thuc Le^{1,2}, Lin Liu¹, Jiuyong Li¹¹ University of South Australia, Adelaide, Australia.² Contact: W.Z. (weijia.zhang@mymail.unisa.edu.au (mailto:weijia.zhang@mymail.unisa.edu.au)), T.L. (thuc.le@unisa.edu.au (mailto:thuc.le@unisa.edu.au))

Abstract

In this challenge we (team CAUSALITY) utilize balanced Multi-layer Regularized Markov Cluster Algorithm (bMLRMCL) method to identify disease modules from a variety of networks.

Background/Introduction

Markov Cluster Algorithm (MCL) is one of the most popular ways for community discovery in biological data. However, the method has the following drawbacks. (1) it does not scale well to large graphs. (2) due to the existence of hub node, the method tend to output highly fragmented clusters (3) the size of output clusters tends to be imbalanced.

We choose an existing method, MLRMCL, to address all above problems [Satuluri et al. 2010].

Methods: Sub-challenge 1

Data pre-processing

There are mainly two steps in the pre-processing procedure. The first one is a filtering operation that discards edges with low weight, and the second one is a scaling operation that transform the edge weight into integers.

(1) We filter out the edge weight that falls in below-threshold quantile. For example, for 4_coexpr we discard all edges with weight smaller than 25% quantile threshold. For 1_ppi we first discard all edges with weight smaller than 75% quantile threshold, and then again filter out all edges with weight smaller than the already filtered 75% quantile threshold.

(2) After filtering, we scale all edge weights by $(x = \frac{(V-1) \cdot (x - \min)}{\max - \min} + V)$, where V is the number of nodes in the graph, max and min are the range of the original weight. This is not performed on 3_signal since it already has integer weight.

All networks are treated as undirected.

Clustering

We first briefly introduce the bMLRMCL algorithm, for more details please refer to [Satuluri et al. 2010].

- Let the adjacency matrix of graph G be M , MCL perform the so-called EXPAND and INFLATE operations on M recursively until convergence. EXPAND is simply defined as $M \times M$ and INFLATE raise each entry by the inflation parameter i and re-normalizing so that each column sums to 1.
- Regularized MCL (RMCL) change the expansion step by replacing $M \times M$ with $M \times M_G$ where M_G is the canonical flow matrix of G . This modification alleviate the problem that MCL produces too many singleton clusters.

- Multi-layer regularized MCL addresses the running time by first coarsening the graph into multiple layer of smaller graphs to run RMCL and project the flow back to the large graphs, the size of the coarsened graph is set to c .
- Last but not least the methods enables adjustable balancing with balance parameter b , which replaces M_G with $M_R = M_G \times P^{-b}$. The balance parameter is used to address the third drawback of MCL.

Therefore bMLRMCL involves three parameters: inflation parameter i , coarsening size c and balance parameter b . The parameters are set individually for each network to optimize performance.

Post processing

The clustering result from Step 2 may contain large clusters (>100) which is not considered in the competition. Therefore we utilizes two different strategy to deal with this.

The first one is "recluster", which construct a subgraph that contains all the nodes in those large clusters and all their associated edges, and then apply the same clustering process in Step 2. This is done recursively until there is no clusters with more than 100 nodes.

The second one is called "discard", which simply discards all clusters with more than 100 nodes.

Putting everything together

To sum up, our methods requires the following parameters.

- (1) the quantile threshold for edge filtering
- (2) the inflation parameter i , coarsening size c and balance parameter b for the clustering.
- (3) the post-processing strategy. And if use "recluster", also the parameters for the re-clustering process.

It is clear that with a total of 20 submission chances it's not possible to try all settings. For tuning parameters for each individual network, we first pick a list of values to try, then if one trials performs reasonably good we do some fine tuning around that value.

Because this is essentially an unsupervised method, the risk of over-fitting is much lower than supervised approaches. We choose the settings that are robust among multiple p-values for our final submission.

Methods: Sub-challenge 2

We aggregate the weight of 1_ppi, 2_ppi and 4_coexpr into one graph by first normalizing the weight in each graph and then sum them together. And then we apply bMLRMCL algorithm to generate the output clusters.

Conclusion/Discussions

bMLRMCL shows very good performance on most of the networks in the Leaderboard phase. However, with the limited submission chances it is hard to say what's the best parameter for each type of network.

The method is also reasonably robust to different p-values and validation set, which makes it a very promising candidate for future investigation (the p-value result for all our Leaderboard submission can be found in the file section).

Source code

The source code is written in R and C, which can be found at the file tab in this project. It should run on any Linux machine, if not please compile the C source with your own machine. Please feel free to contact me if you have any questions.

To replicate the result for sub-challenge 1, please follow the script "Final.R".

To replicate the result for sub-challenge 2, please follow the script "Sub2V27.R".

References

Disease Module Identification DREAM Challenge (syn6156761).

Aggarwal, C. and Reddy, C. Data Clustering: Algorithms and Applications. Chapman & Hall, 2013.

van Dongen, S., Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht, May 2000.

Satuluri, V., and Parthasarathy, S., Scalable Graph Clustering Using Stochastic Flows: Applications to Community Discovery, ACM SIGKDD 2009.

Satuluri, V., Parthasarathy, S., and Ucar, D., Markov Clustering of Protein Interaction Networks with Improved Balance and Scalability, ACM BCB 2010.

Author Contributions

WZ conceived and implemented the method used in this challenge. WZ and TL performed the experiments. WZ TL LL JL analyzed the data and results. WZ wrote the summary. TX and JZ provided background knowledge on disease module network.