CSE 549

Computational Biology

Alignment Free Network Comparison

Prepared By

Harshkumar Patel

Shubhada Patil

Anusha Ghanta

Deepika Peringanji

(Git Repo: https://github.com/shupatil/AlignmentFreeNetwork.git)

Table of Contents

1. Introduction

Graphs can be used to model any type of relationship that exists in real world. Use of graphs and its analysis is increasing day after day. Graph theory has its own importance in many fields including fields of Mathematics, Physics, Biology, Social Network etc. As the importance increases we need a way in which we can work with them as efficiently as possible. When we talk about the graphs, we often come across the problem in which we need to compare two or more graphs. Comparison of graphs is an intractable problem when we are dealing with graphs related to the real world problems. There are many approaches to compare graphs. One obvious thing is to compare graphs node by node which will become intractable as soon as we start talking about graphs with hundreds of nodes. In this document, we tried to highlight few methods that we have used for efficient graphs comparison. We have used alignment free network comparison in which we are looking for structural similarities between graphs in order to compare them.

2. Description

Many methods exists for graph comparison based on their structure. Most the successful methods are based on graph or subgraph isomorphic relations of the underlying graphs. Here, we have described methods based on feature vectors of the graph and their comparison. We developed a signature for each graph here and after that we are comparing these signatures with each other. We are making a distance matrix which will show the difference between signatures of different graphs. After that we are using precision recall curve to verify the result we get. Here, we are comparing graphs of different families (produced via distinct network growth models) which include ER (Erdos-Reyni), DDM, PAM and GEO. We have shown result in terms of precision recall curve along with its area under the curve. We have implemented two different signatures for this project:

1) Degree sequence- Idea from this paper - http://brainmaps.org/pdf/similarity2.pdf
2) Top-k Eigenvalues (EIGS)-Idea from this paper - https://www.cs.cmu.edu/~jingx/docs/DBreport.pdf

2.1 Degree Sequence

This method compare graph based on their feature vectors. Feature vectors are generated in following way. For every graph we are making feature vector based on degree sequence of its nodes. For an undirected graph, degree of any node is the number of nodes associated with through a single edge. So for each graph we will have a feature vector of length equals to the number of the nodes of the graph. I.e. for a graph G with 4 nodes having degree equals to 2, 3, 2, 3 we'll represent its feature vector as (2, 3, 2, 3). Now in order to compare two graphs we are making their degree sequences equal length. So for a degree sequence with smaller number of elements we're appending zeros to it. Once we have equal length vectors we can compare them using following formula.

$$d(G_1, G_2) := \frac{1}{n \cdot \Delta} \sum_{i}^{n} \|s(G_1)(i) - s(G_2)(i)\|.$$

Where d(G1,G2) is the measure of how closely they are related. If this value is close to 1 then graphs are more similar and if this value is close to 0 then they are not similar to each other. Here 'n' is the number of elements in the feature vector. Delta is the highest degree value from any of the vector. S(G) is the degree sequence of the graph G. We've implemented this thing using python's network X library.

The basic idea behind this approach is that if two graphs are very dissimilar then their degree sequence will differ by a large margin and we are calculating this margin in order to compare them. One problem with degree sequence is that two different graphs can have the same degree sequence. And this can spoil the approach used here. In order to address this issue we've used the concept of **line graphs**. Line graph for any given graph is the graph in which we have edges of original graph as nodes in the new graph. And if two edges are neighbor of each other in original graph (i.e. have common vertex between them) then we'll have edge between their corresponding nodes in line graph. Following example will illustrate this concept.

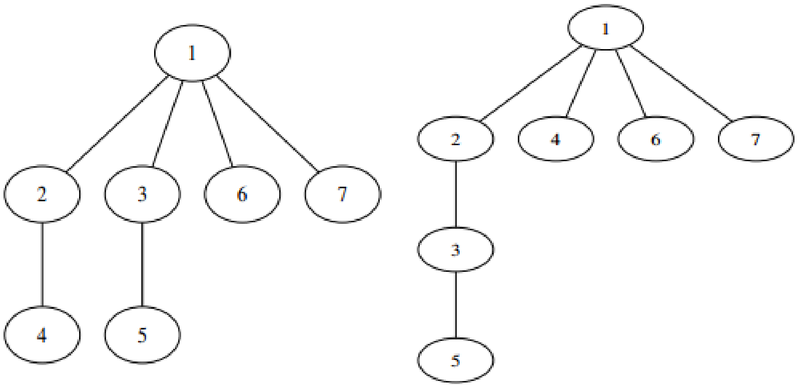


Figure 1 Two different graphs with similar degree sequence

Above two graphs are not same though they have similar degree sequence that is (4, 2, 2, 1, 1, 1, 1). Now we will look at their line graphs which are as below.

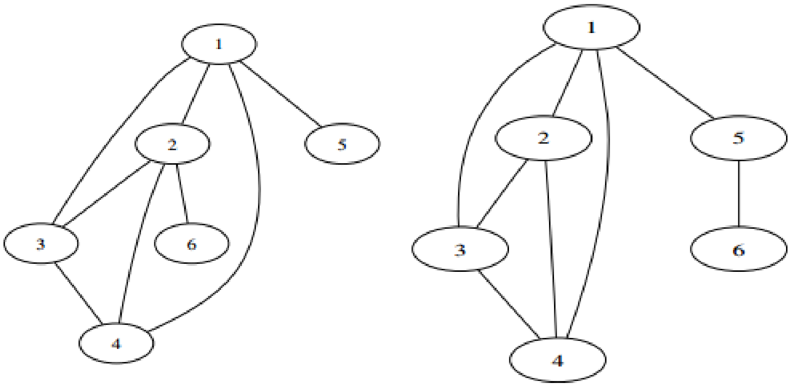


Figure 2 Line graphs of above graphs

Now degree sequence of above graph will be (4, 4, 3, 3, 1, 1) for left graph and (4, 3, 3, 3, 2, 1) for right graph. Thus, by making line graph for each graph we can distinguish between different graphs.

2.1.1 Analysis of Signature

We have implemented this thing in python by making use of network X library of it. We have tested this method with different specification. Following is the result we are getting for different number of graphs and nodes.

Table 1 Result for degree sequence

Graphs	Area Under Curve (AUC)
40 Graphs (100 nodes each)	0.989143
80 Graphs (100 nodes each)	0.989814
20 Graphs (1000 nodes each)	0.998566
80 Graphs (1000 nodes each)	0.999326
280 Graphs (100 nodes each)	0.989900

Below are the precision recall curves for the above results:

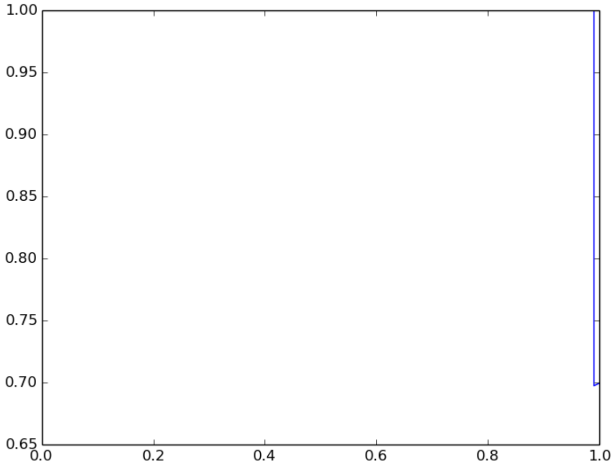


Figure 3 20 Graphs (1000 nodes each)

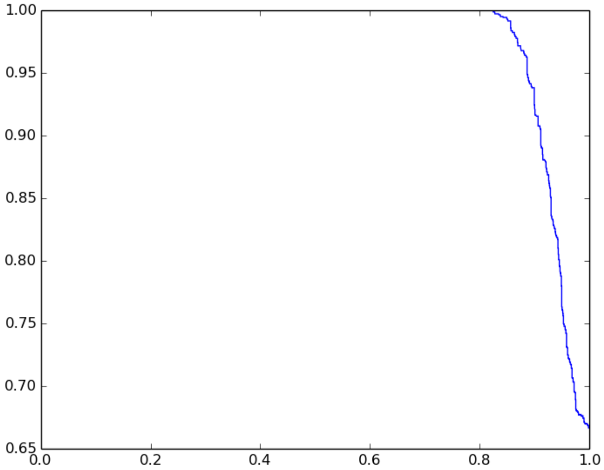


Figure 4 80 Graphs (100 nodes each)

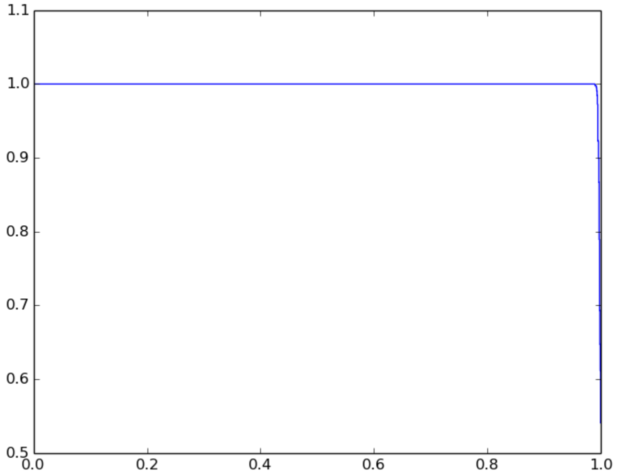


Figure 5 80 Graphs (1000 nodes each)

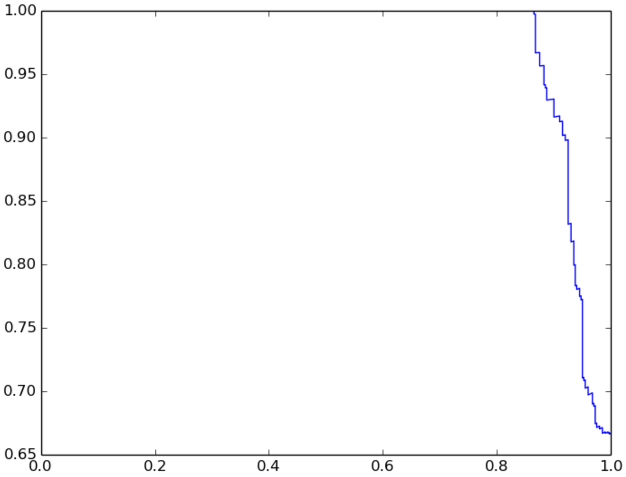


Figure 6 40 Graphs (100 nodes each)

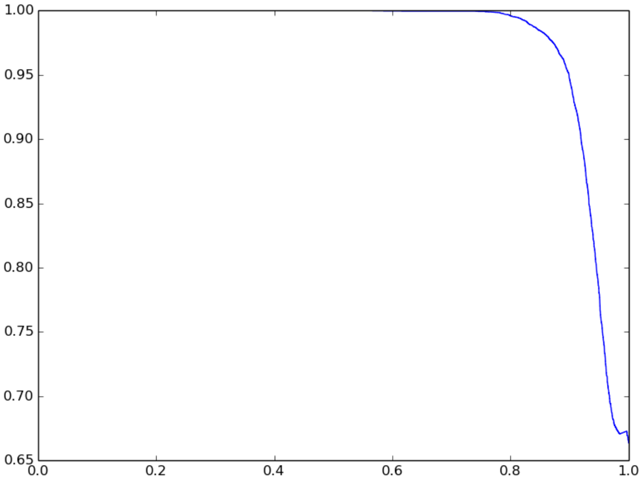


Figure 7 280 Graphs (100 nodes each)

2.2 Top K - Eigen Values

It is known that the eigenvalues of the adjacency matrix are the most important graph invariants. Therefore it is worth considering the power of simply using the dominant eigenvalues. Note that we cannot take all eigenvalues because the total number of eigenvalues varies with the graph size. Instead, we take top-k eigenvalues of the corresponding adjacency matrices and compute the normalized inner product between them. We show the results for k = 5 (EIGS-5) and k = 10 (EIGS-10)

An adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph. In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix with zeros on its diagonal. If the graph is undirected, the adjacency matrix is symmetric. The relationship between a graph and the eigenvalues and eigenvectors of its adjacency matrix is studied in spectral graph theory. In linear algebra, an eigenvector or characteristic vector of a square matrix is a vector that does not change its direction under the associated linear transformation. In other words—if v is a vector that is not zero, then it is an eigenvector of a square matrix A if Av is a scalar multiple of v. This condition could be written as the equation:

$Av = \lambda v$,

where λ is a scalar known as the eigenvalue or characteristic value associated with the eigenvector v.

Geometrically, an eigenvector corresponding to a real, nonzero eigenvalue points in a direction that is stretched by the transformation and the eigenvalue is the factor by which it is stretched. If the eigenvalue is negative, the direction is reversed.

Generation of eigen values: Obtained adjacency matrix of a graph by reading edges of a graph from a file. Given the adjacency matrix eigen values can be generated. These eigen values are used to generate signatures

Generation of signatures: Sort the eigen values and take the top K eigen values as a signature for each graph. Taking the top k eigen values (K eigen values which have the highest energy) helps in reducing the dimension of the input graph while still capturing the intrinsic graph properties.

The time complexity of this procedure is $O(n^2)$, where n is number of nodes in the graph. Can be applied to graphs with different number of nodes.

2.2.1 Analysis of Signature

Number of nodes/Graph	Number of graphs	Top – k	Threshold	Area under ROC curve
1000	40	5	5	0.9278
1000	40	10	5	0.9222
1000	40	10	10	0.9350
100	80	5	10	0.9342
100	80	10	10	0.9906
1000	80	5	10	0.9206
1000	80	10	10	0.9125

Observed running times: For graphs with 100 nodes, the running time was 1-2 minutes. For graphs with 1000 nodes the running time was ~5 minutes.

As K value increases, the information about a graph increases (more eigen values are considered), so there is better performance. We have tried different thresholds to not allow too many false positives or too many false negatives.

Sequence of steps:
first run data_generator.py
then run python eigen.py

ROC curves:

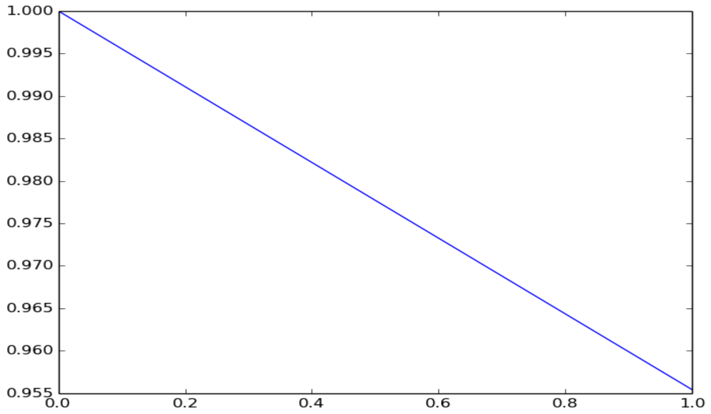


Figure 8 40 Graphs (100 nodes each, k=5, threshold=10))

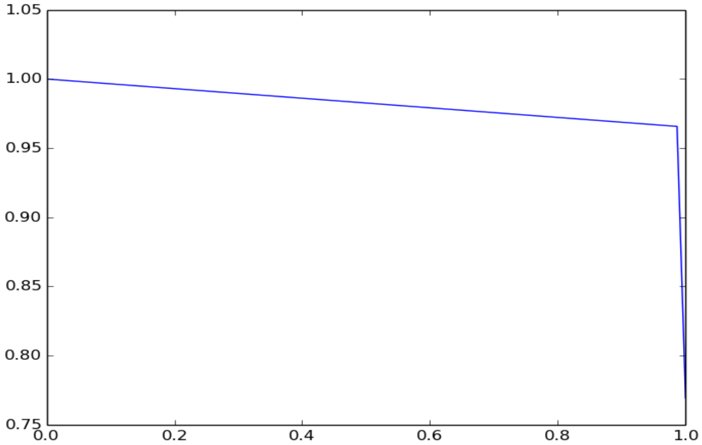


Figure 9 40 Graphs (100 nodes, k=threshold=10)

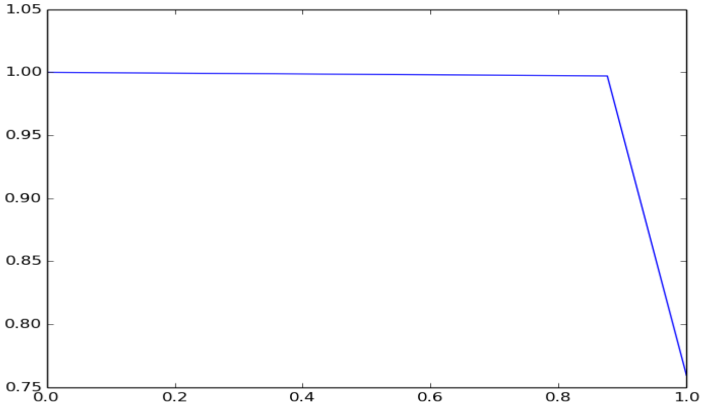


Figure 10 80 Graphs (100 nodes, k=5, threshold=10)

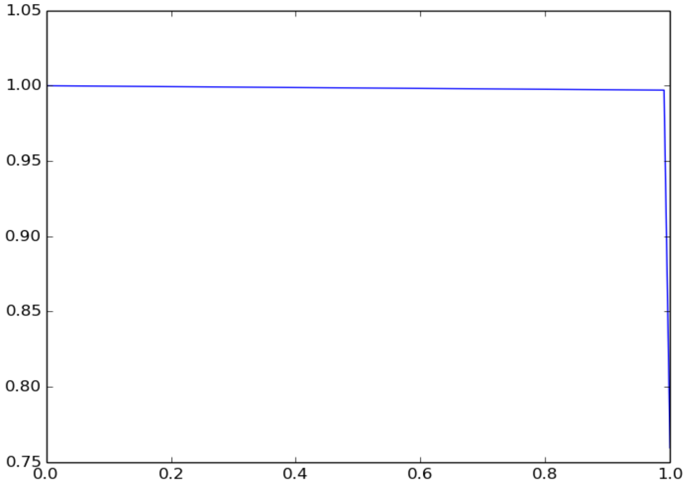


Figure 11 80 Graphs (100 nodes, k=threshold=10)

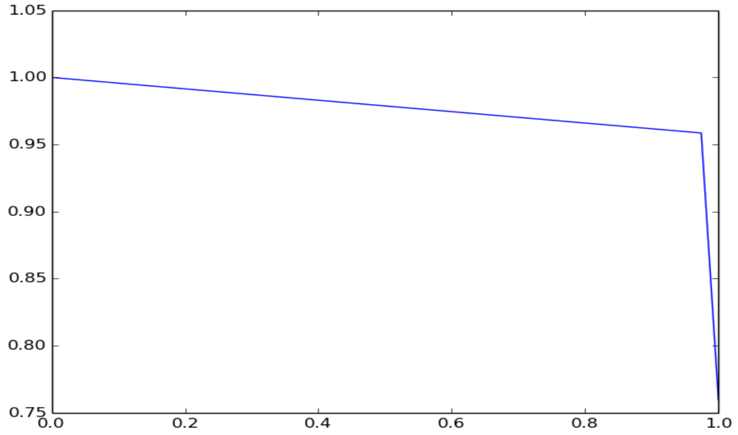


Figure 12 80 Graphs (1000 nodes, k=5, threshold=10)

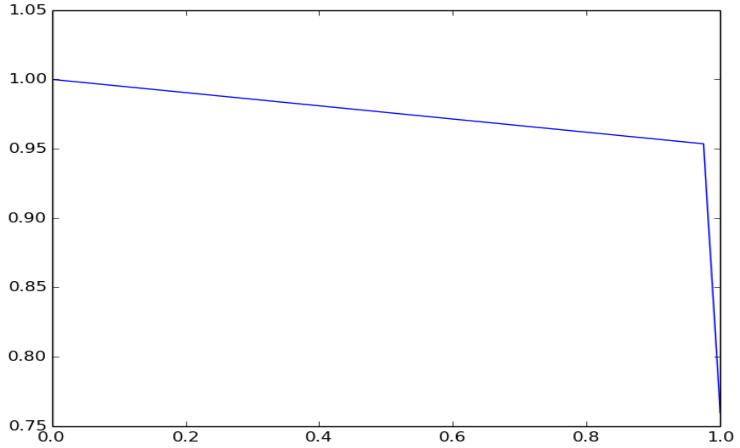


Figure 13 80 Graphs (1000 nodes, k=threshold = 10)

3. Other Methods

Prior to above implementation, we tried below approaches to compare graphs without actual alignment:

3.1 Diameter

The diameter d of a graph is the maximum eccentricity of any vertex in the graph. That is, d it is the greatest distance between any pair of vertices or, alternatively,

$$d = \max_{v \in V} \epsilon(v)$$

To find the diameter of a graph, we first find the shortest path between each pair of vertices. The greatest length of any of these paths is the diameter of the graph. We found that comparing graphs based on the diameter is not a good approach for graph comparison as the algorithm is not able to predict the extent of dissimilarity between two graphs accurately.

3.2 Centricity index

Centricity index of a graph is average of centricity of each vertex. Centricity of a vertex is a number generated by multiplying degree of vertex and its eccentricity. Centricity of vertex is given by following formula:

$C_v = (d_v)(e_v)$

Where d_v is degree of vertex v and e_v is eccentricity of vertex v.

Thus, centricity index of graph G is given by following formula:

$C = \sum_v dv.ev$

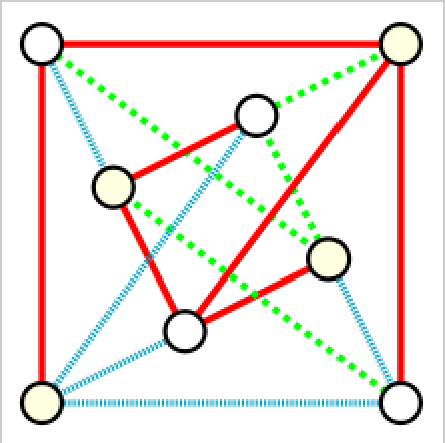
Centricity index is not a good measure to compare graphs as it is highly dependent on the number of edges present in the graph. Centricity index value decreases when we increase edge count in the graph. Thus this method might predict two graphs as dissimilar but same graphs can become similar if we increase edges in graphs which has higher centricity index compared to other one. Thus, the results of this method are not accurate.

3.3 Arboricity

The arboricity of an undirected graph is the minimum number of forests into which its edges can be partitioned. Equivalently it is the minimum number of spanning forests needed to cover all the edges of the graph. The arboricity of a graph is a measure of how dense the graph is: graphs with many edges have high arboricity, and graphs with high arboricity must have a dense subgraph.

In more detail, as any n-vertex forest has at most n-1 edges, the arboricity of a graph with n vertices and m edges is at least $\lceil m/(n-1) \rceil$. Additionally, the subgraphs of any graph cannot have arboricity larger than the graph itself, or equivalently the arboricity of a graph must be at least the maximum arboricity of any of its subgraphs. Nash-Williams proved that these two facts can be combined to characterize arboricity: if we let n_S and m_S denote the number of vertices and edges, respectively, of any subgraph S of the given graph, then the arboricity of the graph equals

$$\max\{\lceil m_S/(n_S-1) \rceil\}.$$



A partition of the complete bipartite graph $K_{4,4}$ into three forests, showing that it has arboricity three.

Arboricity is a good measure to predict similarity between graphs accurately. The only disadvantage of using arboricity as a basis of graph similarity/dissimilarity is its time complexity. Arboricity is a graphlet based technique and has an exponential time complexity. Thus, it takes a lot of time to compare dense graphs.

4. Conclusion

Comparison of graph should include structural similarity between them. In this document we have highlighted few methods along with its result analysis. It is clear from the result that along with methods that include graphlets kind of substructure, we can also develop feature vectors that can be used to compare different graphs. These methods are also performing well compare to those conventional methods. Advantage of such approaches is that they can be run in polynomial time compare to methods that involves graphlets.

5. References

1. Network X : https://networkx.github.io/
2. http://brainmaps.org/pdf/similarity2.pdf
3. http://arxiv.org/pdf/1404.4644.pdf
4. http://people.csail.mit.edu/whit/macindoe_richards_soccom10.pdf
5. https://en.wikipedia.org/wiki/Graph_property
6. http://scikit learn.org/stable/modules/generated/sklearn.metrics.precision_recall_curve.html
7. http://arxiv.org/pdf/1404.4644.pdf- top 5
8. https://www.cs.cmu.edu/~jingx/docs/DBreport.pdf
9. http://www.cs.utexas.edu/~ssi/mseigs_final.pdf