Project Final Report

Jinyang Ruan & Yize Hu

CPT_S 591 Elements of Network Science

5/2/2021

Washington State University

I. Abstract

This paper mainly discussed the differences between several methods of computing the similarity between graphs. Instead of using existing graph similarity algorithms directly, we firstly compared different matrices distances algorithms. Manhattan distance, Euclidean distance, Minkowski distance with high power values, and Frobenius distance are mainly discussed in this paper. Based on the distance between matrices, we calculate the similarity scores between graphs. Then, two existing similarity methods which include eigenvector similarity and DeltaCon similarity were discussed. We figured out and concluded those methods are sensitive to what kind of characteristics such as number of nodes and edge probability values. Finally, we tried to apply those methods to real-world graphs and tried to verify the results we got from previous work.

II. Introduction

When it comes to graph similarities, two types of graph similarities which include similarity between nodes in a graph and similarity between graphs are commonly discussed. More specifically, when we talk about similarity between graphs, we usually categorize it into graph similarity with known node correspondence and graph similarity with unknown node correspondence. In this paper, we will mainly discuss the similarities between graphs with known node correspondence. We firstly construct the adjacency matrices and Laplacian matrices for each graph. Instead of computing the similarity between graphs directly, we compute the distance between two adjacency matrices. If the distance is larger, we consider there is less similarity between graphs. However, there are many methods which can be used to calculate the distance between matrices, such as Manhattan distance, Euclidean distance, Minkowski distance, Canberra distance, etc. We will discuss those distances separately in this

paper. During the study, we found that the Canberra distance is very sensitive to the small changes near zero and we can seldom calculate it successfully, so we decided that we would mainly focus on the Minkowski distance family and Frobenius distance since these distance algorithms are most commonly used. Eigenvector similarity and DeltaCon similarity are existing and sophisticated similarity algorithms, we will also apply them to our experimental graph samples. We have mainly analyzed whether those similarity methods are sensitive to the number of nodes or edge probability values. Three real-world networks of co-authorships between scientists posting preprints on the Condensed Matter E-Print Archive in different years are used to verify our study outcomes [1].

III. Data Pre-processing

RStudio is the tool we used for our study and calculation.

First of all, as Figure 1 shows, we generated a graph with 300 nodes and 0.5 edge probability value through Erdős–Rényi method as comparative reference graph. All similarity scores we got were from the calculations between another graph and this graph, called er_1 graph.

```
er_1 = erdos.renyi.game(300, p.or.m = 0.5 , type= c('gnp'))
```

Figure 1. Comparative reference graph generation function

Then we generated 21 graphs with same number of nodes and different edge probability values. As it is shown in Figure 2, the range of the edge probability values is from 0.01 (close to 0) to 1 with the step 0.05.

```
er_2 = erdos.renyi.game(300, p.or.m = 0.01 , type=c('gnp'))
er_3 = erdos.renyi.game(300, p.or.m = 0.05, type=c('gnp'))
er_4 = erdos.renyi.game(300, p.or.m = 0.1, type=c('gnp'))
er_5 = erdos.renyi.game(300, p.or.m = 0.15, type=c('gnp'))
er_6 = erdos.renyi.game(300, p.or.m = 0.2, type=c('gnp'))
er_7 = erdos.renyi.game(300, p.or.m = 0.25, type=c('gnp'))
er_8 = erdos.renyi.game(300, p.or.m = 0.3, type=c('gnp'))
er_9 = erdos.renyi.game(300, p.or.m = 0.35, type=c('gnp'))
er_10 = erdos.renyi.game(300, p.or.m = 0.4,type=c('gnp'))
er_11 = erdos.renyi.game(300, p.or.m = 0.45, type=c('gnp'))
er_12 = erdos.renyi.game(300, p.or.m = 0.5, type=c('gnp'))
er_13 = erdos.renyi.game(300, p.or.m = 0.55, type=c('gnp'))
er_14 = erdos.renyi.game(300, p.or.m =  0.6, type=c('gnp'))
er_15 = erdos.renyi.game(300, p.or.m =  0.65, type=c('gnp'))
er_16 = erdos.renyi.game(300, p.or.m =  0.7, type=c('gnp'))
er_17 = erdos.renyi.game(300, p.or.m =  0.75, type=c('gnp'))
er_18 = erdos.renyi.game(300, p.or.m =  0.8, type=c('gnp'))
er_19 = erdos.renyi.game(300, p.or.m =  0.85, type=c('gnp'))
er_20 = erdos.renyi.game(300, p.or.m =  0.9, type=c('gnp'))
er_21 = erdos.renyi.game(300, p.or.m =  0.95, type=c('gnp'))
er_22 = erdos.renyi.game(300, p.or.m =  1, type=c('gnp'))
```

Figure 2. 21 Random graphs with same number of nodes

Then we generated 25 graphs with the same edge probability value and different number of nodes were generated in the same way. As it is shown in Figure 3, the range of the number of the nodes is from 2 to 600.

```
```{r}
er_30 = erdos.renyi.game(2, p.or.m = 0.5 , type= c('gnp'))
er_31 = erdos.renyi.game(20, p.or.m = 0.5 , type= c('gnp'))
er_32 = erdos.renyi.game(50, p.or.m = 0.5 , type= c('gnp'))
er_33 = erdos.renyi.game(80, p.or.m = 0.5 , type= c('gnp'))
er_34 = erdos.renyi.game(100, p.or.m = 0.5 , type= c('gnp'))
er_35 = erdos.renyi.game(120, p.or.m = 0.5 , type= c('gnp'))
er_36 = erdos.renyi.game(150, p.or.m = 0.5 , type= c('gnp'))
er_37 = erdos.renyi.game(180, p.or.m = 0.5 , type= c('gnp'))
er_38 = erdos.renyi.game(200, p.or.m = 0.5 , type= c('gnp'))
er_39 = erdos.renyi.game(220, p.or.m = 0.5 , type= c('gnp'))
er_40 = erdos.renyi.game(250, p.or.m = 0.5 , type= c('gnp'))
er_41 = erdos.renyi.game(280, p.or.m = 0.5 , type= c('gnp'))
er_42 = erdos.renyi.game(300, p.or.m = 0.5 , type= c('gnp'))
er_43 = erdos.renyi.game(320, p.or.m = 0.5 , type= c('gnp'))
er_44 = erdos.renyi.game(350, p.or.m = 0.5 , type= c('gnp'))
er_45 = erdos.renyi.game(380, p.or.m = 0.5 , type= c('gnp'))
er_46 = erdos.renyi.game(400, p.or.m = 0.5 , type= c('gnp'))
er_47 = erdos.renyi.game(420, p.or.m = 0.5 , type= c('gnp'))
er_48 = erdos.renyi.game(450, p.or.m = 0.5 , type= c('gnp'))
er_49 = erdos.renyi.game(480, p.or.m = 0.5 , type= c('gnp'))
er_50 = erdos.renyi.game(500, p.or.m = 0.5 , type= c('gnp'))
er_51 = erdos.renyi.game(520, p.or.m = 0.5 , type= c('gnp'))
er_52 = erdos.renyi.game(550, p.or.m = 0.5 , type= c('gnp'))
er_53 = erdos.renyi.game(580, p.or.m = 0.5 , type= c('gnp'))
er_54 = erdos.renyi.game(600, p.or.m = 0.5 , type= c('gnp'))
```
```

Figure 3. 25 Random graphs with same edge probability value

We considered the comparisons among graphs with same number of nodes as horizontal comparison and comparisons among graphs with same edge probability values as vertical comparison. After generating $1 + 21 + 25 = 47$ graphs, we got 46 graph pairs which include 21 horizontal comparison pairs and 25 vertical comparison pairs.

So far, all necessary basic graph data for the study is set.

IV. Distance Algorithms Study Analysis

When using Minkowski-family distance methods, the formula looks like this:

$$distance = \left(\sum_{i=1}^{n} |a_i{}^p - b_i{}^p| \right)^{\frac{1}{p}}$$

In the case of p = 1, the method is also known as the Manhattan distance. And in the case of p = 2, the method is also known as the Euclidean distance. We have also included the third and fourth-order for better comparison results. Outside of the Minkowski family methods, we thought about employing the Frobenius and Canberra methods which both can produce decent distance results. But with further experiments, we found that the Canberra distance was too sensitive on the adjacency matrices of weakly connected graph, so we decided to just go with the Frobenius distance. The Frobenius distance formula looks like this:

$$distance = \sqrt{Trace(A^* A^{-1})}$$

After the calculation of the distance, we calculate the corresponding similarity scores. For the similarity metric, we would think that the more similar two graph is the higher the similarity score would be, and in order not to produce error result with Graphs having same identity properties. We normalize the result by adding the normalizing factor.

$$similarity\ score = \frac{1}{1(norm\ factor) + \sqrt{distance}}$$

And to tweak the importance of the distance calculation in the similarity score, we put epsilon factor in front of the distance series.

$$similarity\ score = \frac{1}{1 + \varepsilon * \sqrt{distance}}$$

During the experiment we find that this graph comparison method is incapable of carrying out the vertical comparison and we need to make some modification so that the

similarity score is compatible with the vertical comparison. The modification that we are going with is basically copying the elements in a smaller matrix preserving its identity and into a bigger container matrix filled with -1. Initially we were thinking make the larger matrix filled with 0. After careful trade off evaluation, we decide to make the container matrix filled with -1, since in this way the new matrix preserve the graph matrix identity and corresponding dimension.

We plot the similarity results of all methods on adjacency and Laplacian matrix for vertical and horizontal comparison.

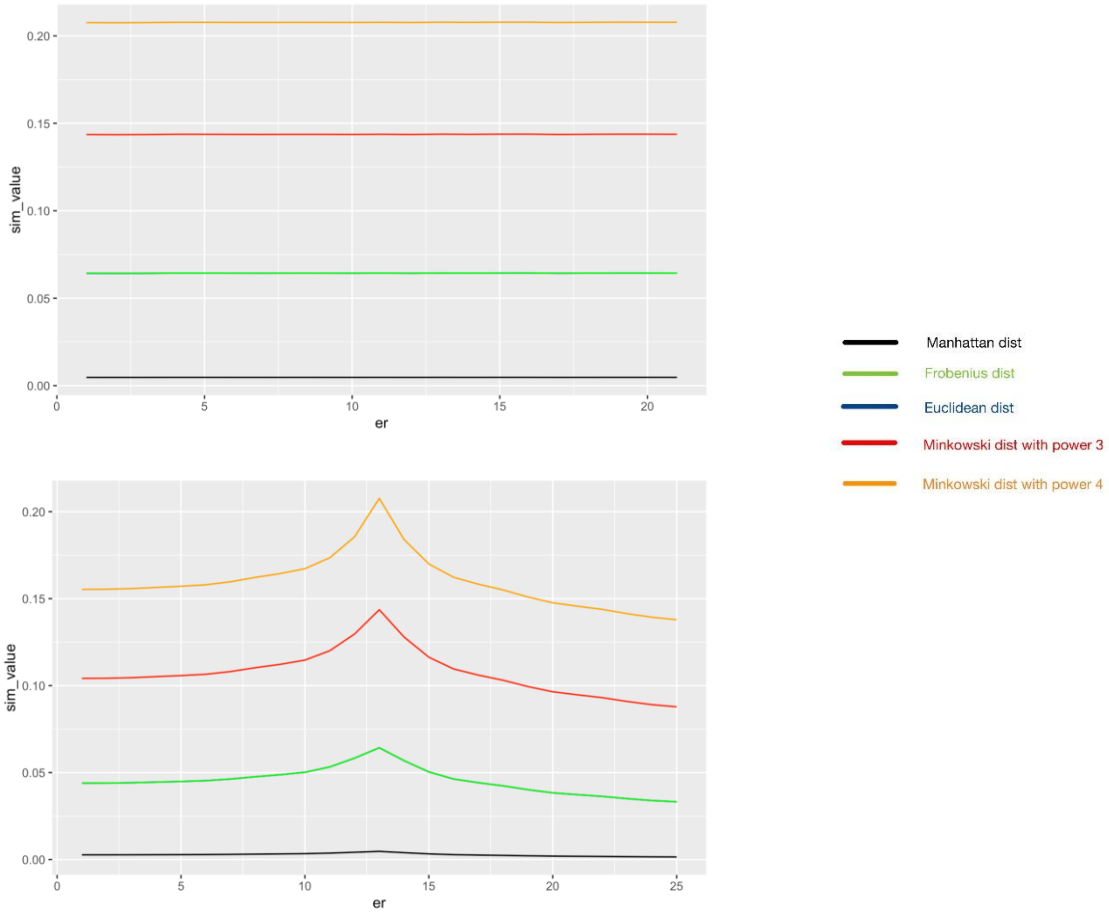

Figure 4 The horizontal and vertical(correspondingly) comparison over the adjacency matrix of a graph

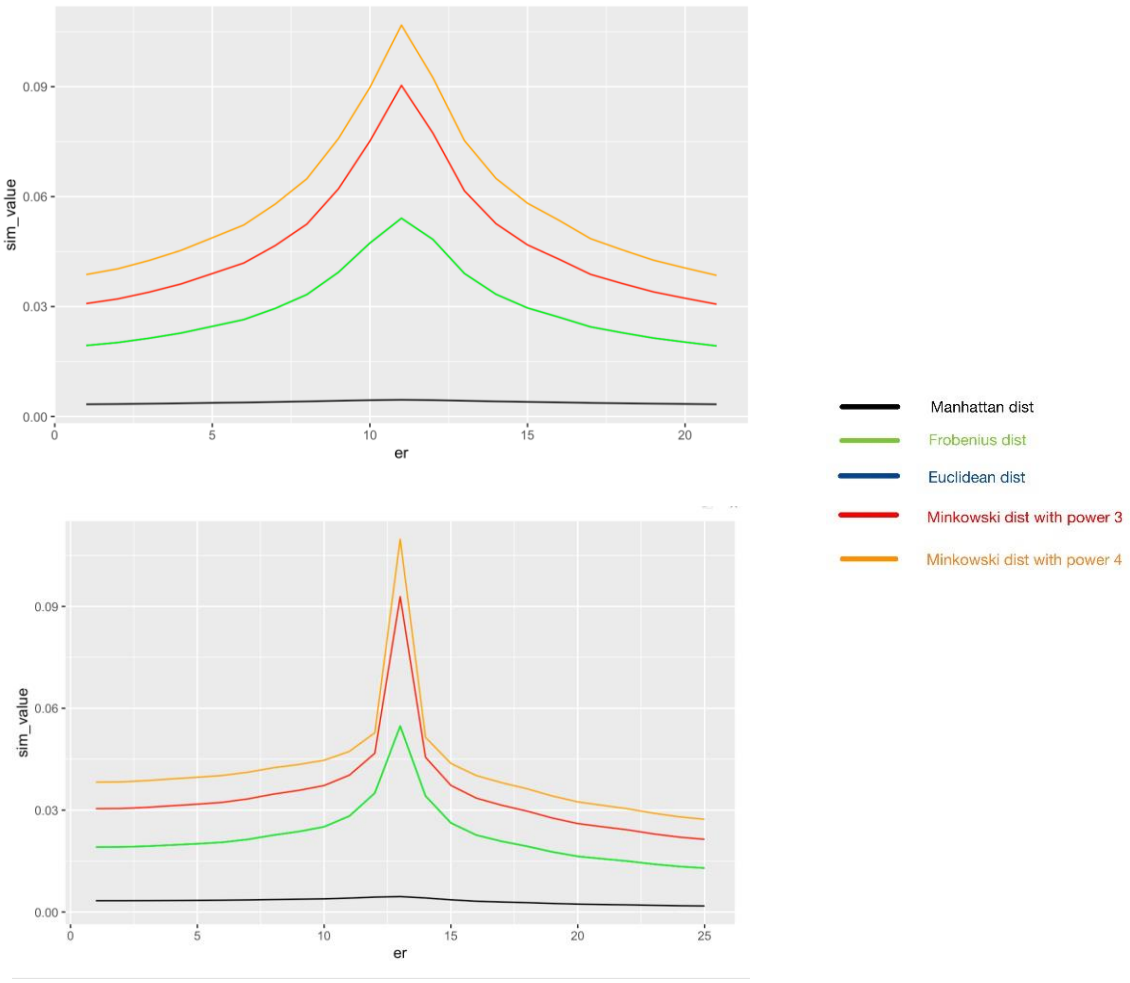

Figure 5 The horizontal and vertical(correspondingly) comparison over the Laplacian matrix of a graph

We want to see the plot fluctuates as we can reach the peak point where we perceive to be the peak similarity point between graphs. The Laplacian matrix appears to perform much better regardless under the condition of vertical or horizontal comparison. And in terms of the integrity of the fluctuation in the graph we don't want the peak point to be getting out of hand. As we can see in the second plot in figure 5. As the order of Minkowski family method gets higher, the peak point is influence in a drastically sensitive way. The Mahattan distance in contrast does not seem to be affected by the peak point, looks roughly about a straight line. In terms of fluctuation of the peak point and stability, we decide to go with Laplacian matrix with Frobenius and Euclidean distance.

V. Eigenvector Similarity and DeltaCon

Eigenvector similarity is one existing method to calculate the similarity between graphs. Basically, we calculate the Laplacian eigenvalues for the adjacency matrices of each of the graphs. For each graph, find the smallest k such that the sum of the k-largest eigenvalues constitutes at least 90% of the sum of all of the eigenvalues. If the values of k are different between the two graphs, then use the smaller one. The similarity metric is then the sum of the squared differences between the largest k eigenvalues between the graphs. This will produce a similarity metric in the range $[0, \infty)$, where values closer to zero are more similar [2]. Based on the definition of the eigenvector, we used the following function (Figure 6.) in R to calculate the eigenvector similarity score:

```
```{r eigensimilarity}
eigensimilarity <- function(graph1, graph2) {
  k1 = select_k(spectrum(graph1)$vectors)
  k2 = select_k(spectrum(graph2)$vectors)
  k = min(k1, k2)

  sim <- sum(as.matrix((spectrum(graph1)$vectors[1:k]) - as.matrix(spectrum(graph2)$vectors[1:k]))**2)
  return(sim)
}
```
```

Figure 6. Eigenvector similarity function

According to the Koutra et al., DeltaCon similarity is a principled massive similarity intuitive, and scalable algorithm that assesses the similarity between two graphs on the same nodes [3], which means DeltaCon similarity should be sensitive to the edge probability value. To apply the DeltaCon method, the logic of the algorithm we use is from a GitHub project [4]. The original code is attached in the appendix shown as “delta_func_block”.

We applied these two methods to our 46 pairs of graphs and check whether they are sensitive to the number of nodes or the edge probability values.

The first step of study is based on the horizontal comparisons, which is aiming to figure out whether those two similarity methods are sensitive to the edge probability value. The result is shown in Figure 5 below.

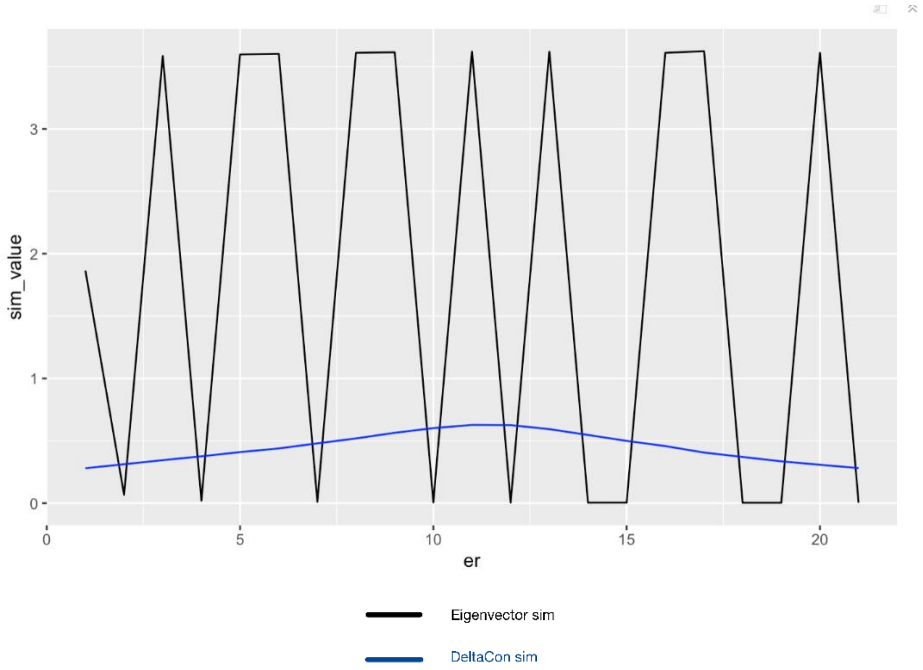

Figure 5. Similarity for each pair of graphs with same # of nodes

In which black line presents the eigenvector similarity and blue line represents the DeltaCon similarity based on the horizontal comparisons. We can see there is not any regulation in black line. When the edge values tend to be the same and different, the eigenvector similarity goes up and down without any regulation. Since eigenvector is not predictable in this case, we consider that eigenvector similarity is not sensitive to the edge probabilities. However, the blue line which is presented the DeltaCon similarity is more regular, we can make the observation that when the edge values tend to be same and different, the DeltaCon similarity score goes up and down very smoothly. Therefore, we can claim that the DeltaCon similarity is sensitive to the edge probabilities.

The second step of the study is based on the vertical comparisons. Different to the first step, now we are aiming to figure out whether those two similarity methods are sensitive to the number of nodes. After calculation, the result is shown in Figure 7.

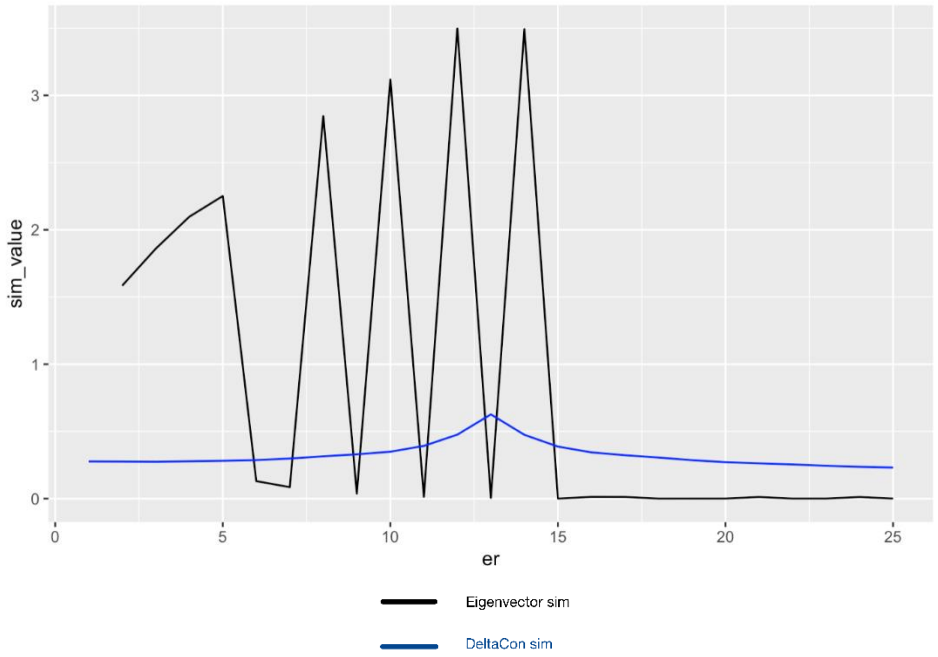

Figure 7. Similarity for each pair of graphs with same edge probability value

From the figure above, we can get the result that eigenvector similarity is also unpredictable in this case since the black line goes up and down without any regulations. We can make the observation that eigenvector similarity is not sensitive to the difference of the number of nodes between graphs. DeltaCon similarity which is presented by blue line is more regular than eigenvector similarity. However, it is not difficult to conclude that the range of increase and decrease of the DeltaCon score in figure 6 is smaller than it in figure 5, and similarity's growth and decline are not smooth. The DeltaCon similarity score is more sensitive when the number of nodes of two graphs is similar.

In conclude, eigenvector similarity is not sensitive to either the number of nodes or the edge probabilities. However, the DeltaCon similarity is much more sensitive to the edge probabilities, which is verified the claim that the DeltaCon similarity is a principled massive similarity intuitive, and scalable algorithm that assesses the similarity between two graphs on the same nodes.

VI. Application In the Real-World Graphs

In this section, we applied Frobenius similarity, eigenvector similarity, and DeltaCon similarity to three real-world graphs in order to check the results about sensitivities of those three methods we got from the above study.

Three networks of co-authorships between scientists posting preprints on the Condensed Matter E-Print Archive in different years are the graphs we used [4], detailed information is shown in the table below:

Name	Number of nodes	Number of edges	Edge probability
Condensed matter collaborations 1999 (1st)	16,726	47,594	3.4027e-4
Condensed matter collaborations 2003 (2nd)	31,163	120,029	2.4721e-4
Condensed matter collaborations 2005 (3rd)	40,421	175,692	2.1507e-4

To simplify, we named those three graphs as 1st graph, 2nd graph, and 3rd graph. After roughly computation through equation $p = 2 * \frac{E}{n(n-1)}$, where p is the edge probability, E is the number of the edges in the graph, and n is the number of the nodes, we can get the 1st graph has the highest edge probability value, and the 3rd graph has the lowest edge probability value.

Firstly, we applied DeltaCon similarity algorithms, the DeltaCon similarity scores is shown below:

Graph pairs	DeltaCon similarity score
1st and 2nd	0.3244495
2nd and 3rd	0.4366916

1st and 3rd	0.3056546

Intuitively, since three graphs show the relationship between one scientist group in different years, compared with the 3rd graph, the 2nd graph should be more similar to the first graph, and the similarity score between the 1st graph and 3rd graph should be the smallest. The computation results perfectively match the intuition. The result we get from the previous study is DeltaCon similarity is more sensitive to the edge probabilities and change of the similarity scores is smooth when the edge probabilities tend to be same. In order to check is claim, we performed the following calculation:

Edge probability difference (abstract)	DeltaCon similarity score
1st and 2nd : 0.9306e-4	0.3244495
2nd and 3rd : 0.3214e-4	0.4366916
1st and 3rd : 1.2520e-4	0.3056546

$$\frac{|0.9306 - 0.3214|e-4}{|0.3244495 - 0.4366916|} = 5.43755348e-4$$

$$\frac{|0.9306 - 1.2520|e-4}{|0.3244495 - 0.3056546|} = 17.1003836e-4$$

$$\frac{|0.3214 - 1.2520|e-4}{|0.4366916 - 0.3056546|} = 7.10181094e-4$$

The change of the slope of DeltaCon similarities along with the change of edge probability difference is larger than except. From this study on the real-world graphs, we can conclude that DeltaCon similarity is sensitive to the edge probabilities, but it is not smooth when the edge probabilities are relatively small.

Then we applied the eigenvector similarity to those three graphs, we got the eigenvector similarity scores as shown below:

Graph pairs	Eigenvector similarity score

1st and 2nd	3.930377e-06
2nd and 3rd	8.829113e-08
1st and 3rd	3.059245e-06

Since the range of the eigenvector similarity metric is from 0 to infinitive, where the values closer to 0 are more similar, 2nd graph is more similar to the 1st graph, compared with the 3rd graph, which is the same result we got from the DeltaCon similarity. However, we can intuitively claim that eigenvector is not suitable in this case because all eigenvector similarity scores we got are extremely small.

Unfortunately, in this study, we are not able to apply the Frobenius similarity algorithms to those three real-world since it requires huge memories of the computer. We can predict that Frobenius similarity scores will accord with Laplacian matrix's fluctuation peak point.

VII. Conclusion and Further Work

In conclusion, the first step of our study compared several commonly used matrices distance algorithms, and we got the result that Frobenius distance is better to be used for the similarity calculation since it is more stable than 3rd and 4th order Minkowski family method and more sensitive compared to the Mahattan method. Eigenvector similarity is not sensitive to either the number of nodes or edge probability values. DeltaCon is more sensitive to the edge probabilities.

For the further work, we think the real-world graphs study is relatively insufficient because we did not consider the computing power of our laptop and the number of the real-world graphs is too small. Therefore, we would like to pick more suitable real-world graphs, including the size of the graphs and the number of graphs. Moreover, there are many other methods to calculate the similarity between graphs, and we only discussed a few in this paper, so we would like to make more comparisons among different similarity algorithms, not only similarity between graphs

with known node correspondence, but also similarity between graphs with unknown node correspondence and similarity between nodes in a graph. Finally, we also would like to try to propose a new similarity algorithm with its own sensitivity.

References

[1] Network data. http://www-personal.umich.edu/~mejn/netdata/

[2] Esultanik, (2014), Retrieved from: https://stackoverflow.com/questions/12122021/python-implementation-of-a-graph-similarity-grading-algorithm

[3] Koutra, Danai, Shah, Neil, Vogelstein, Joshua T, Gallagher, Brian, & Faloutsos, Christos. (2014). DELTACON: A Principled Massive-Graph Similarity Function with Attribution. *ACM Transactions on Knowledge Discovery from Data, 10*(3). https://doi.org/10.1145/2824443

[4] Network data. http://www-personal.umich.edu/~mejn/netdata/

Appendix

The full code is listed in the following link:
https://drive.google.com/file/d/1Jg-C3eRK-3FP2k9IDIACHY4Z4J7zvzJB/view?usp=sharing