A LAYOUT ALGORITHM FOR VISUALIZATION OF GRAPH ALIGNMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ANDAÇ AKARSU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JANUARY 2017

Approval of the thesis:

A LAYOUT ALGORITHM FOR VISUALIZATION OF GRAPH ALIGNMENTS

submitted by **ANDAÇ AKARSU** in partial fulfillment of the requirements for the degree of **Master Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Assoc. Prof. Dr. Tolga Can
Supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. Göktürk Üçoluk
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Tolga Can
Computer Engineering Department, METU _____

Prof. Dr. Faruk Polat
Computer Engineering Department, METU _____

Asst. Prof. Dr. Yusuf Sahillioğlu
Computer Engineering Department, METU _____

Prof. Dr. Uğur Doğrusöz
Computer Engineering Department, BİLKENT _____

Date: **26/01/2017**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ANDAÇ AKARSU

Signature :

ABSTRACT

A LAYOUT ALGORITHM FOR VISUALIZATION OF GRAPH ALIGNMENTS

Akarsu, Andaç

M.S., Department of Computer Engineering

Supervisor : Assoc. Prof. Tolga Can

January 2017, 40 pages

Graph layout algorithms are commonly used when visualizing. Usually these algorithms focus on a single graph. To be able to visualize multiple graphs at once, such as the results of graph alignment algorithms on biological networks, new layout algorithms need to be developed. A layout algorithm for visualizing graph alignments should display the aligned graphs separately, so that both the graphs and their alignment can be viewed individually. In addition, for better interpretation of the alignment results, similar to single graph layout algorithms, edge crossings should be minimized and highly connected subsets of nodes should be grouped together. In this thesis, we propose a graph layout heuristic for visualization of alignments of two graphs. The list of aligned nodes, ordered with respect to node similarity, is taken as input from the graph alignment algorithm. The nodes are re-ordered using an approach adapted from a solution to the minimum linear arrangement problem. The nodes are then placed using their degrees. The proposed layout algorithm is applied on biological networks and the resulting layouts are assessed for quality using the number of edge crosses, a standard measure for evaluating layout algorithms. Our results show that the proposed algorithm performs better for graph alignment visualization compared to standard layout algorithms.

Keywords: Graph Visualization, Graph Layout, Graph Alignment, Minimum Linear Arrangement

ÖZ

ÇİZGE HİZALAMA İÇİN ÇİZGE GÖRSELLEME ALGORİTMASI

Akarsu, Andaç

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi : Doç. Prof. Tolga Can

Ocak 2017, 40 sayfa

Çizge yerleşim algoritmaları çizgeleri ve ağları görselleme için yaygın olarak kullanılır. Genellikle bu algoritmalar tek bir çizge üzerine odaklanır. Birden fazla çizgeyi tek seferde görselleyebilme için, biyolojik ağlar üzerinde uygulanan çizge hizalama algoritmalarının sonuçları gibi, yeni yerleşim algoritmaları geliştirilmelidir. Çizge hizalama görsellemesi için kullanılacak yerleşim algoritması, hizalanan çizgeleri birbirinden ayrı olarak göstermelidir ki, çizgeler ve hizalamaları bireysel olarak görüntülenebilsin. Ayrıca, hizalama sonuçlarının daha iyi yorumu için, tek bir çizge görselleme algoritmalarına benzer olarak, kenar kesişmesi küçültülmeli ve yoğun olarak birbirine bağlı düğüm alt kümeleri bir arada gruplanmalıdır. Bu tezde, hizalanmış iki çizgenin görselleştirilmesi için bir çizge yerleşim sezgisi öneriyoruz. Hizalanmış düğümlerin listesi, düğüm benzerliklerine göre sıralanarak, çizge hizalama algoritmasından girdi olarak alınır. Düğümler minimum doğrusal düzenleme çözümünden adapte edilen bir yaklaşım kullanılarak yeniden sıralanır. Düğümler sonrasında dereceleri kullanılarak yerleştirilir. Önerilen yerleşim algoritması biyolojik ağlar üzerinde uygulanır ve elde edilen yerleşimler, yerleşim algoritması değerlendirmede standard bir kriteri olan kenar kesişimleri sayısı kullanılarak kalite için değerlendirilir. Sonuçlarımız önerilen algoritmanın çizge hizalama görselleştirirken standard yerleşim algoritmalarına kıyasla daha iyi performans sergilediğini gösterir.

Anahtar Kelimeler: Çizge Görselleme, Çizge Yerleşimi, Çizge Hizalama, Minimum Doğrusal Düzenleme

to my dearest family

ACKNOWLEDGMENTS

First of all I would like to thank my thesis advisor Doç. Prof. Tolga Can. He has been a great help through the learning process of this master thesis with his comments, remarks, and guidance. None of this would have been possible without him.

I am also grateful to my family for their continuous support and encouragement throughout my years of study and during the process of writing this thesis.

Finally, this thesis work is supported by the Scientic and Technological Research Council of Turkey (TÜBİTAK) 1001 Program Grant #114E111

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Graphs are the most popular and convenient way to represent relational data in different domains ranging from social networks to biological networks. Nodes can be viewed as objects and edges as the relations between objects. Graph size (number of vertices and edges) varies depending on the data. Small graphs can be drawn manually; however, manual drawing would be very difficult for large graphs. Therefore, a method to place nodes and edges on a display window is required. For this purpose, different graph layout algorithms have been proposed. Some of these algorithms are hierarchical layout, circular layout, and force-directed layout [1, 2]. Force-directed graph drawing algorithms are the most commonly used ones.

Force-directed graph layout algorithms usually take advantage of physical systems to calculate forces and reposition nodes in an iterative way. Charged particles and springs are among the earlier physical systems that have been used. However, force-directed graph layout algorithms are applied on a single network and not on multiple networks. Therefore, the input is considered as a single graph and all the nodes, irrespective of the graph they belong to, are considered during force computation.

In some research problems, the focus is on multiple networks rather than a single one. For example, in pairwise graph alignment, we align two graphs. As the result of graph alignment, we get a one-to-one mapping of the nodes in these two graphs, with some of the nodes possibly left unaligned. In the area of bioinformatics, graph alignments provide us the information of similarity between two biological networks. For instance, we can determine the similarity between two different species or certain parts of two different proteins in the case of alignment of protein structure networks.

Graph alignments help us understand differences between normal and disease states at a systems level.

An alignment of two graphs can be considered as another single graph, in which the set of nodes is the union of the nodes of the two input graphs. There are three types of edges in an alignment graph: 1) the edges of the first graph, 2) the edges of the second graph, and 3) the alignment edges. While placing nodes and edges, standard layout algorithms treat the given alignment graph as a single graph with no distinction between nodes and the different types of edges and do not separate graphs from each other. However, while displaying the alignment result, we need to show two graphs separately with alignment edges between nodes from different graphs. Therefore, in this thesis, we propose a new approach to visualize graph alignment results.

We have first tried to draw the alignment results manually to identify the specific requirements of a layout method for aligned graphs. We have observed that a good layout for the alignment can be obtained by taking one of the graphs as a reference graph and grouping the nodes of this graph such that the edge crossings within the graph are minimized. If the aligned parts of the two graphs are topologically similar, such a grouping of the nodes of the reference graph will also include reduced number of edge crossings in the other graph. This observation constitutes the basis of the proposed algorithm in this thesis. The proposed algorithm starts with taking clusters from the result of graph alignment on two graphs. Then, it separates the graphs into left and right side of the display window, which is a two-dimensional plane. We show the aligned edges in between the two graphs, one by one, as horizontal line segments; therefore; there are no edge crossings between alignment edges. We place the graphs in different sides while placing the nodes of the individual graphs using an approximation algorithm for the minimum linear arrangement problem, which is NP-Complete. Minimum linear arrangement ensures reduced number of edge crossings between original edges of the individual graphs.

Organization of the rest of this thesis is as follows. In Chapter 2, we give some background information. Basically we review graphs, graph layout algorithms, graph alignment algorithms, and the minimum linear arrangement problem. In Chapter 3,

we describe the proposed algorithm in detail. In Chapter 4, we present the experimental results. The datasets used for experiments and the evaluation metrics are given in that chapter. In Chapter 5, the summary and future work for the thesis are given.

CHAPTER 2

BACKGROUND

2.1 About Graphs

Graphs are suitable for modeling relational data. Graphs consist of vertices and edges and a graph is the tuple $G = (V, E)$, where V and E are vertex and edge sets. Graphs are widely used for social networks, biological networks, and database systems. Some of the definitions related to graphs in the context of this thesis are given below:

Undirected Graph: In this type of graphs, edges do not have an indicator for direction. So, edges are drawn simply as line segments.

Unweighted Graph: In this type of graphs, edges contain no weight information.

Subgraph: A **S**ubgraph is simply a part of another graph. From a graph $G = (V, E)$ we can take a subgraph $G` = (V`, E`)$ where $V` \subseteq V$ and $E` \subseteq E$, such that the edges in E' are the edges induced by V'.

2.2 Graph Layout

Graphs have the information for nodes and their relations. However, generally, a graph does not contain information about the visualization of these components. In other words, the placement of the nodes and edges in a 2D display window is usually not given. The purpose of graph layout algorithms is to place nodes in a way that the graph would be easier to interpret. Some of the graph layout algorithms are: hierarchical layout, circular layout, tree layout, and force-directed layout algorithms.

There are some criteria to measure readability and interpretability of a graph. Some of these are the spread of vertex placement, uniform edge length and the number of edge crossings. Vertex placement is critical to readability, since usually, while looking at a graph, we focus on vertices. We need to distribute vertices uniformly. Placing most of the vertices in a corner and the remaining ones in another will not provide an efficient view. Also vertices should not overlap, since we may not focus on the vertex we want to focus. Vertex placement also affects edge crossings. The number of edge crossing should be minimized, since we also focus on the relations while reading a graph. With increasing number of edge crossings, tracing relations become harder. Following these criteria usually provide readable and understandable graphs. To be able to see the distinction, two example layouts are provided below. The first one is an example of random layout of a graph. In random layout, nodes are placed randomly (Figure 2.1). The second one is the force-atlas-2 [5] layout view of the same graph. It provides a better layout since the above mentioned criteria are taken into consideration (Figure 2.2).

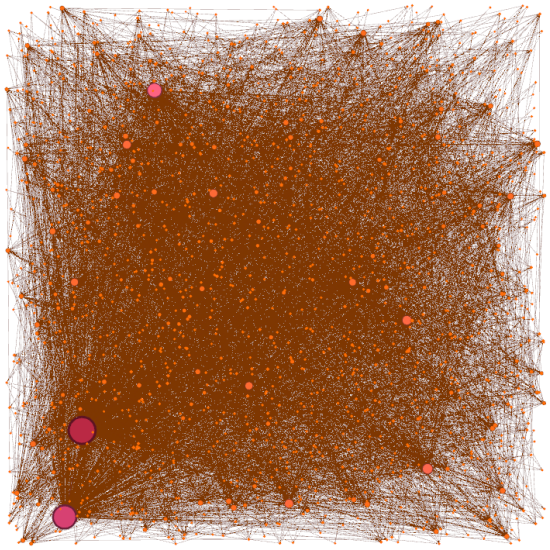

Figure 2.1: Random Layout applied on “java.gexf” dataset from Gephi’s sample datasets. This dataset has 1538 nodes and 8032 edges.

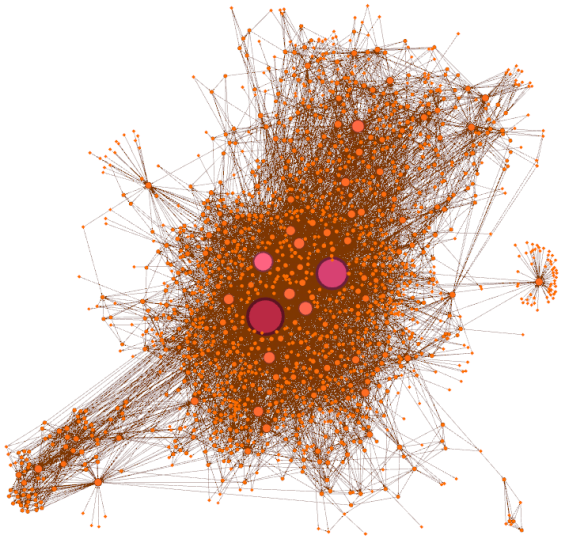

Figure 2.2: Force atlas 2 applied on “java.gexf” dataset. Compared to Figure 2.1, it is more readable. Some of the clusters are more visible.

2.3 Graph Alignment

In molecular biology, there are many kinds of networks such as protein – protein interaction networks, regulatory networks, DNA - protein interaction networks, and metabolic networks. Graph alignment on different networks can provide us information about diseases, protein functions, and evolution.

Graph alignment is a problem of finding similar nodes and edges, using both topological and biological similarity measures. The alignment between two networks can provide us information about species or proteins. We can see whether a certain functionality of a specie is similar to another specie which provides information on evolution. Also we can learn the way the proteins work and this can be helpful in finding a cure for a disease.

In the thesis study, we have used the PROPER (PROtein-protein interaction network alignment based on PERcolation) algorithm as our graph alignment algorithm [3]. Some of the alignment algorithms follow a process where it starts by some information on connected nodes and from the connected nodes iteratively provide a matching for two graphs. This approach in network alignment is also known as

percolation graph matching (PGM). PGM provides the basic idea for the PROPER algorithm. PROPER takes two graphs and BLAST bit-score similarities for the protein pairs from different graphs. Using similarity scores, it generates a seed set which is the information on connected nodes. Then the algorithm applies an iterative procedure to finish alignment of the remaining proteins.

2.4 Related Work

Layout algorithms use different types of approaches to visualize data. Some of them employ a certain approach to provide a certain shaped layout, like circular layout, while some of them take advantage of physical systems, like the force-directed layout. Some layout algorithms also use additional biological data, like Eclerize [4]. Whichever is the case, they usually focus on visualizing a single graph.

While visualizing graph alignment results, we need to display the alignment information in addition to the aligned graphs. This case is different from traditional layout algorithms. Therefore, we need a different kind of graph layout approach.

There are some approaches used in earlier research for visualizing multiple networks, such as C. Erten's work [15]. Although it is focusing on providing a visualization of multiple networks simultaneously, it does not focus on graph alignment. Focus of C. Ertan's algorithm is to provide a visualization of multiple networks, whose nodes are belong to the same set of nodes with differing edges. However, in our case, we are providing the visualization of two graphs, whose nodes and edges are completely different from one another. During studies of graph alignment algorithms, researches used some approaches to provide a basic view of their algorithms. There are two kinds of layout approaches, side by side and all in one. Side by side is an intuitive approach where you need to draw graphs next to each other, and then show the alignment. All in one, as the name implies, tries to put aligned graphs all together. These approaches are described in detail below:

Side by side

Firstly, while placing graphs next to each other, this method tries to place nodes on imaginary levels where nodes from different graphs are aligned. In this way, alignment edges are not required to be drawn (Figure 2.3). However, it is hard to follow conserved edges in the alignment.

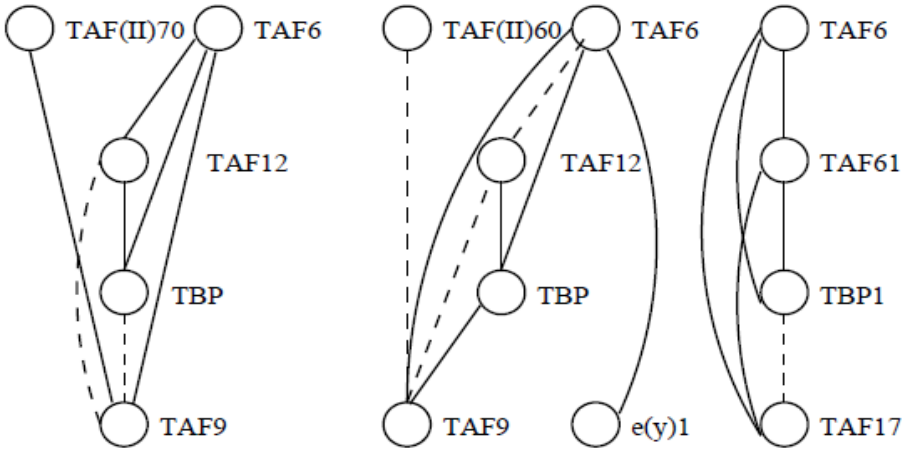

Figure 2.3: Side by side approach with aligned nodes on the same horizontal line [6].

Another one is to keep the graphs the way they are. Imaginary levels are not required. This method uses alignment edges to show the alignment result. In this approach, occurrence of edge crossings is the main drawback. Increasing number of edge crossings make it hard to interpret alignment edges. (Figure 2.4).

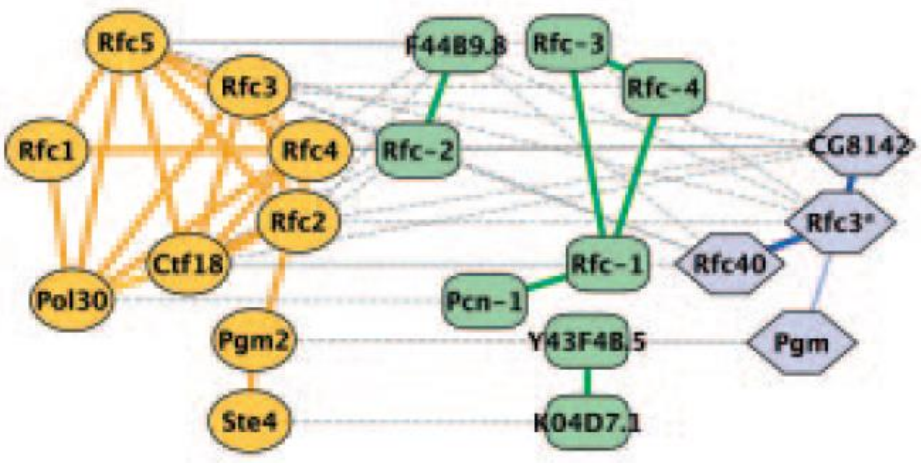

Figure 2.4 : Side by side approach with edges between aligned nodes [7].

The third approach requires graphs to be looking similar [8]. This way, only eye matching is required. No additional effort is needed. Unfortunately, if the graphs are not looking similar, this approach is quite confusing (Figure 2.5).

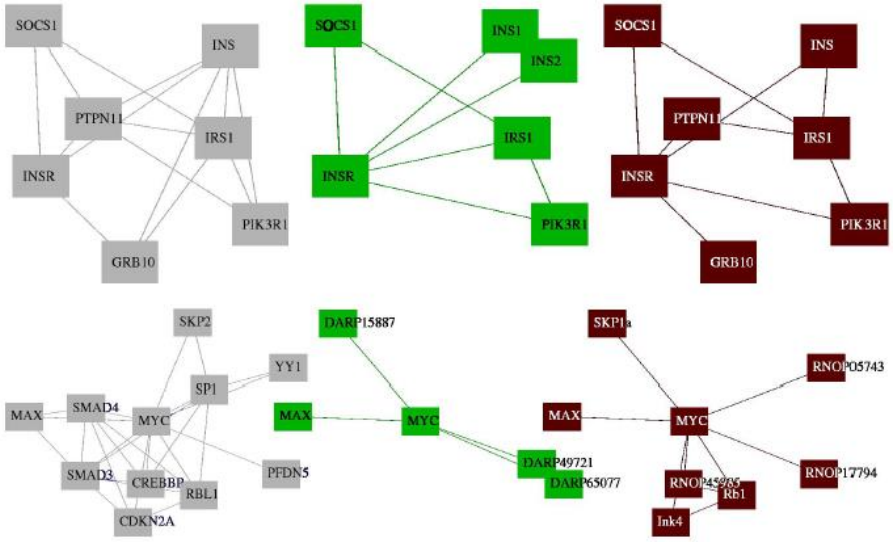

Figure 2.5: Side by side approach with eye matching. As we can see, if the graphs are similar, it is reasonable (top part). However, it is confusing for the bottom one [8].

All in one

To be able to show the graphs as one graph, graph labels and colors are used. Aligned nodes are displayed as single nodes with two labels. Edges are colored differently to imply alignment edges or edges from different graphs. In other words, visualization information is added in order to provide the layout (Figure 2.6).

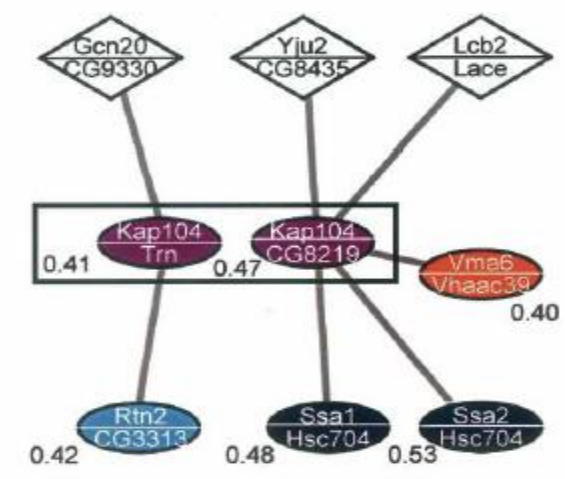

Figure 2.0.6: All in one approach with labels and edge colors [9].

Another all in one approach uses the meta-graph concept. Aligned nodes form meta-nodes. Meta-edges are formed by connecting meta-nodes. The meta-nodes provide a clear visualization of alignment information. However, coloring is used to distinguish meta-edges to show which network the edge belongs to. (Figure 2.7).

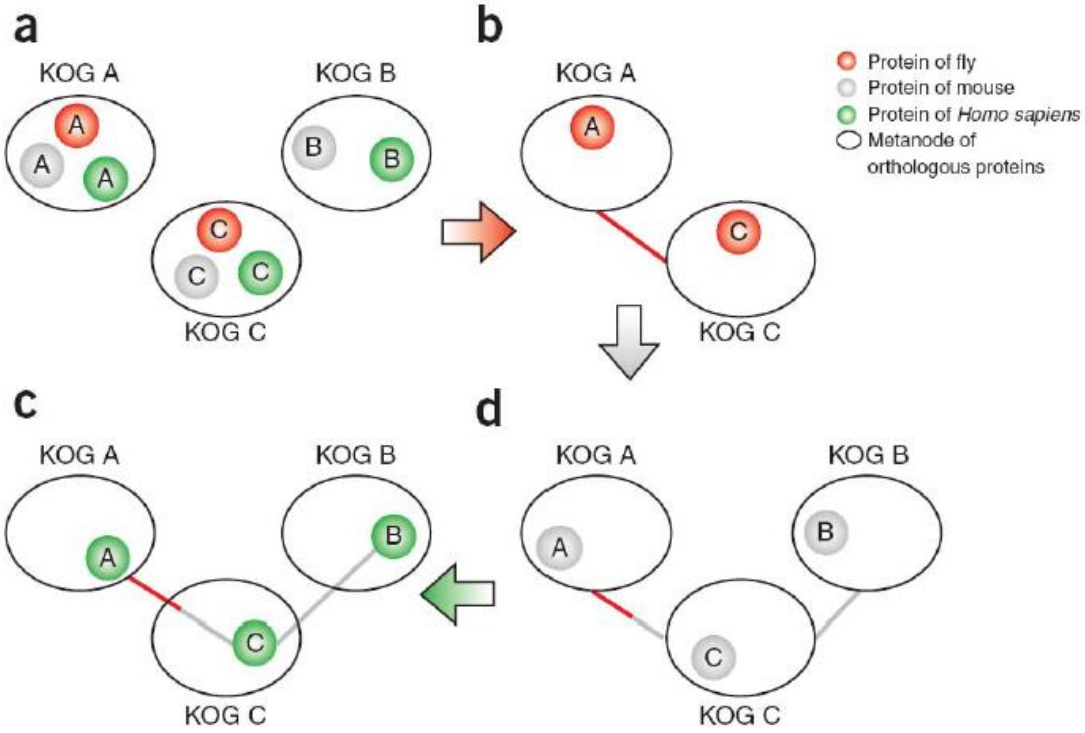

Figure 2.7: All in one approach with the meta-graph concept. a) only meta-nodes, b) –d) projections of meta-nodes on each network [10].

2.5 Minimum Linear Arrangement

Minimum linear arrangement problem (MinLA) is the problem of placing nodes of the given graph ($G = (V, E)$) on an integer line in a way that the total edge length is minimized. In other words, the solution is like a one-to-one function f: $V \rightarrow [1, 2.., |V|]$. MinLA is an NP-Hard problem [11].

In the study of Y. Koren, & D. Harel [11] they are using a multi scale algorithm for MinLA problem. Firstly, the nodes are sorted using spectral sequencing. Then a cost reduction algorithm called Median Iteration is applied. Median iteration is simply iterating over the graph and moving nodes into their neighbors' median. It helps to group nodes together. However, after grouping them their linear arrangement is decided randomly. Finally, after applying Median Iteration, subgraphs of the input graph are selected iteratively and arrangement of the nodes is reassessed.

In this thesis study, we have adapted a heuristic approach from the solution of multi scale algorithm for MinLA problem. Mainly, the adapted part is from Median Iteration. Median Iteration calculates median to move nodes, but, we calculate the mean to move nodes. Also, unlike Median Iteration, while moving nodes we do not group them, however, after each calculation we rearrange placements while moving the node to its new position.

2.6 Markov Cluster Algorithm

Markov Cluster Algorithm (MCL) is a clustering algorithm for graphs [14]. The basic idea behind graph clustering is that in a graph cluster nodes have higher number of edges in between while there are fewer number of edges between clusters. Therefore, starting with a node and randomly traveling to connected nodes, it is more likely to stay in a cluster. Similar to other clustering algorithms MCL is based on the same idea.

In this thesis study, we have used MCL as an initialization method for the proposed algorithm during our initialization experiment.

CHAPTER 3

ALGORITHM

Molecular biologists carry out their research to learn more about diseases, human life, and evolution. Biological networks have an important role to provide the information. For instance, comparing networks of two different species and finding out conserved parts of the networks may help understanding evolution. Similarly comparing two protein-structure networks might help with how the protein works and eventually it can be useful while researching diseases since proteins are also involved in diseases. These types of comparisons are facilitated by graph alignment algorithms. Results of the alignment are used by the biologists. Therefore, clear visualization of graph alignment helps biologists to understand the alignment better. While providing a clear visualization, alignment data and graphs need to be displayed simultaneously. However, current graph layout algorithms do not aim to provide visualization for this purpose. This is why this thesis tries to provide a heuristic graph layout algorithm to visualize graph alignment results.

3.1 Initial Heuristic

In this algorithm, we want to display two graphs and their alignment at the same time, looking similar to a DNA drawing or showing one to one function between two sets (Figure 3.1). We expect displaying a layout in this way will provide a clear visualization of alignment information.

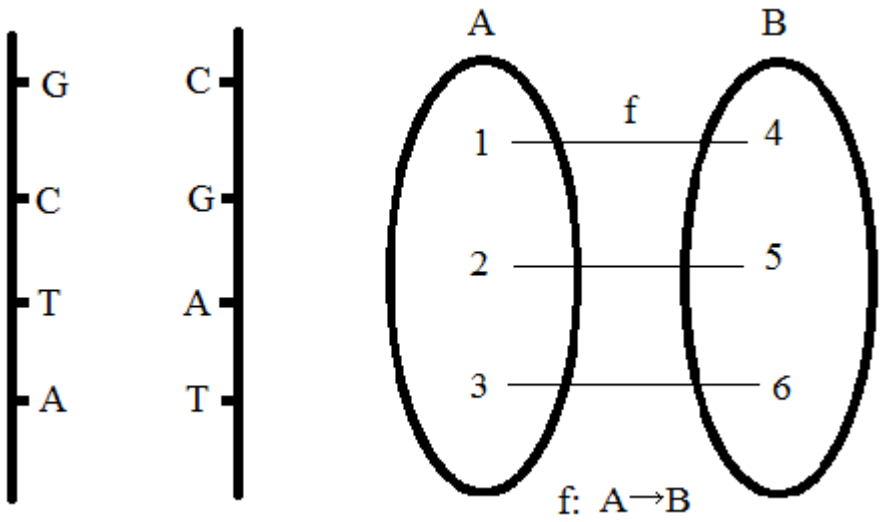

Figure 3.1: DNA drawing (Left) and a one to one function representation (Right).

To be able to display graphs individually we needed to separate graphs into right and left sides. To show the alignment result, we use alignment edges between two nodes which are from two different graphs. In other words, if node *a* from graph A and node *b* from graph B has an alignment edge between them, implies that these nodes are aligned (Let's say for$G_A = (V_A, E_A)$, $G_B = (V_B, E_B)$ there exists an edge $\{v_a , v_b\}$ such that $v_a \subseteq V_A$ and $v_b \subseteq V_B$ then v_a , v_b are aligned).

In graph layout algorithms, minimum edge crossing is desired overall considering all three types of edges. In terms of edges, there are three groups of edges. Edges that belong to graph A, edges that belong to graph B and alignment edges. Separating graphs should be useful while comparing edges from graph A and edges from graph B. With separation they should not cross. Therefore, edges of A and edges of B can be considered separately. Alignment edges are between graphs, so we expect that there will be minimum edge crossing between alignment edges, A and B edges. To minimize edge crossing among alignment edges and to visualize alignment information better, we use a trivial solution. If the aligned nodes stay on the same y-coordinate and no other aligned nodes stays on that y-coordinate, then all alignment

edges will be parallel to each other as horizontal line segments and they will not intersect each other. It can be explained mathematically like this:

Graph A is on the left side of x' and graph B is on the right side of x'. Let there be an alignment edge between nodes a and b. Their positions are given as (x_a, y_a) and (x_b, y_b) where $y_a = y_b$ and$x_a < x' < x_b$. Without changing the quality of the layout, y_a and y_b may be perturbed; therefore, we may actually relax the restriction of y_a and y_b being equal.

At the start, we were going to have a layout similar to the layout in Figure 3.2 in which we just order the aligned nodes from top to bottom on two sides using similarity scores. The most similarly aligned nodes are placed at top whereas the least similar alignment is placed at the bottom. However, although this approach makes the number of crossings of aligned edges zero, the number of edge crossings among the edges of the individual graphs is not minimized and may results in a non-interpretable alignment. At this point, we needed to improve the layout to decrease the number of edge crossing. Firstly, we needed to arrange x-coordinates, so that they will not be on the same x-coordinate, which causes all edges from a single graph to be on the same line. Then we needed to find a way to put nodes with edges in between grouped together, so that edges will have a lower chance to cross other edges.

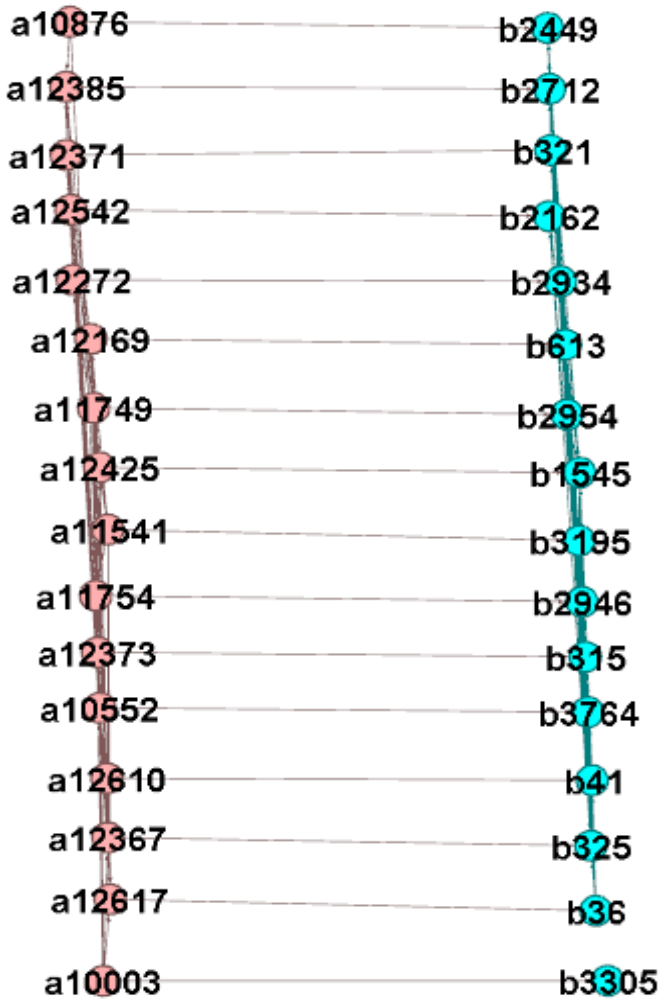

Figure 3.2: Starting layout of our algorithm with alignment edges ordered from top to bottom randomly (Manually drawn).

3.2 Input and Preprocessing

These are the inputs the algorithm takes:

- Two graphs to be aligned,
- Iteration count,
- Two scaling constant for x and y coordinates

First of all, the alignment of the graphs is obtained by running the PROPER algorithm. Alignment result for the input graphs can be calculated by PROPER's

online tool [12]. Then BLAST bit scores are added to the alignment results. Next, the edges that are formed between aligned nodes are separated for each of the given graph. Also, vertices are labeled for the information of which graph they belong to. Lastly, we import the three edge sets, edges of the aligned nodes for the input graphs and alignment result, into the graph visualization tool Gephi.

3.3 The Proposed Algorithm

The proposed algorithm processes the result of alignment data as an undirected and unweighted graph. Using the label information, vertices are separated to put the vertices from one graph to one side and the remaining ones to the other side. While doing so, aligned vertices are put at the same y coordinate, similar to our initial heuristic. Then for the individual graphs, we use an approach, where we calculate positions one after another iteratively. Firstly, we calculate y positions then x positions.

For *y* positions we would like to arrange the nodes in a way that nodes with edges between them are placed closely and nodes without edges are placed farther. Intuitively, we expected that this will reduce the chance of two edges to cross each other, since the edges will be shorter. For instance, if we place two nodes with an edge between them at faraway positions, we expect that edge to cross all the edges in between. We do not want this to happen. This kind of arrangement is a studied problem in some domains and it is called minimum linear arrangement problem. Basically, minimum linear arrangement tries to place *n* nodes into 1 to *n* linear positions. We apply this on one graph and expect a similar result in the other graph since they are aligned beforehand. (If they are aligned then they are topologically similar, so applying minimum linear arrangement moves a node and, doing the same thing on the other graph should yield a similar result.)

In the proposed algorithm, we have not used a linear arrangement solution the way it is, but we have adapted a heuristic approach from Koren and Harel's [11] solution of minimum linear arrangement for calculation of *y* positions. Koren and Harel are using a method called "Median Iteration" to lower the cost of the minimum linear arrangement. We have used an approach similar to Median Iteration. Median

Iteration calculates median of the neighbors of nodes iteratively to group nodes. However, we calculate mean of the neighbors of nodes iteratively and in each iteration we slide placement of the nodes to calculated position rather than let the nodes group together.

For *x* positions we have simply moved nodes with higher degrees to further away from the x'. For the nodes with x-coordinate x < x', we have moved these nodes to left side and for nodes with x-coordinate x > x' we have moved these nodes to right side. The idea behind this is that we expect that a node with higher degree has a greater chance of increasing number of edge crossing if it is placed to the inner side of the graph. Let's say the node with highest degree has d number of edges and it is placed to the innerside, then we expect that it will cross all the other alignment edges around. However, considering the case where it is set up the other way around, the node with the lowest degree will be probably be close to the connected edge. Therefore, it is not expected for that node to cause an edge crossing. In Figure 3.3 the pseudocode of the proposed algorithm is provided.

Algorithm 1: The Proposed Algorithm

input: A Graph $G = (V, E)$ graph of alignment result for $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$ and alignment edges
k = iteration count,
c_x, c_y = scaling constants for x,y
output: Layout of G

```
1 :  Preprocessing()
2 :  for i = 1 to k
3 :     for every v ∈ V
4 :         y = yPlacement(v), y' = Median(N(v))
5 :         moveNodes(y, y')
6 :  end for
7 :  for every v ∈ V
8 :      xPosition(v) = degree(v) * c_x + x'
9:       yPosition(v) = yPlacement(v) * c_y
10: end for
```

Figure 3.3: Algorithm 1. Pseudocode of the proposed algorithm

Some of the variables and functions are defined as follows: “Preprocessing” is the function where we prepare our input to import Gephi. “moveNodes” function simply rearranges placement of nodes by sliding the placement one by one for placements between two given spots. "degree” function calculates the degree of a given node. c_x is a constant that shows the separation of two adjacent nodes on the x-coordinate axis. c_x is also multiplied by (-1) for vertices of Graph A, since we want to push these vertices to left side. c_y is a constant that is used to separate nodes in horizontal direction. x' is the middle point for graph layout.

In general, the proposed algorithm is similar to the second side-by-side approach we have mentioned in Chapter 2. The difference from the one discussed is that graphs have not been placed the way they are. But, vertices are tried to be placed while edge crossing is being considered. With a better vertex placement to lower the chance to edge crossings to occur, which was the drawback for that side-by-side approach, we have provided a better layout. In Figures 3.4 and 3.5 we have provided some screenshots of the generated layout view.

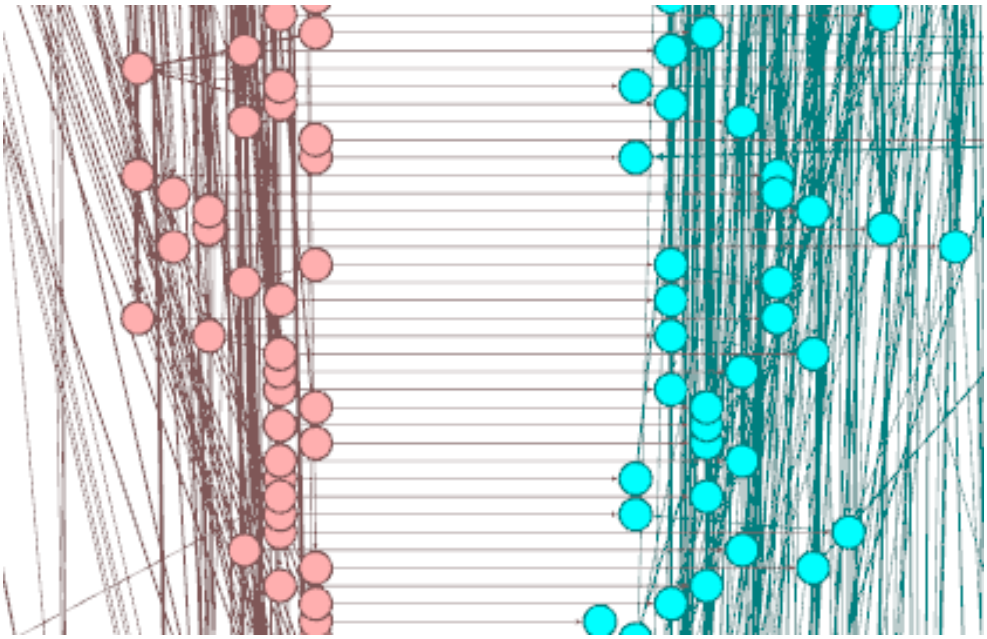

Figure 3.4: A portion of the graph layout result of the proposed algorithm on Ce-Dm (Worm-Fly) dataset.

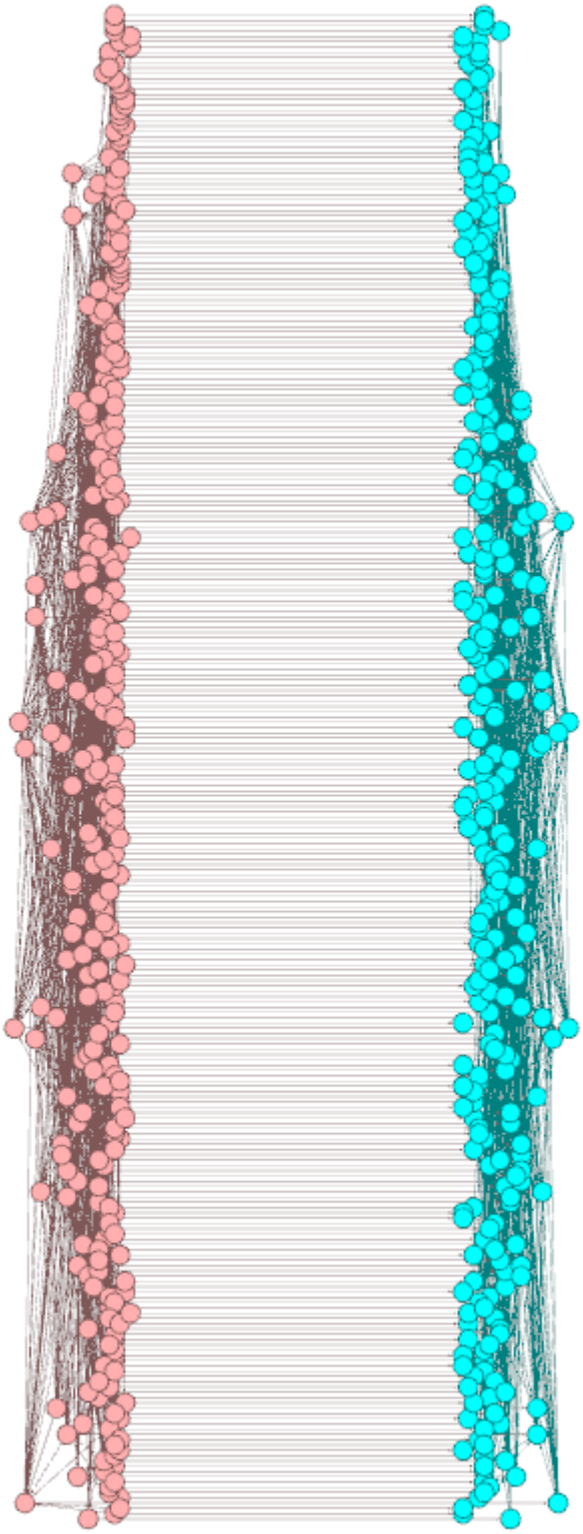

Figure 3.5: Preview of the graph layout result of the proposed algorithm on synthetic test data

CHAPTER 4

EXPERIMENTS AND RESULTS

Development of the proposed algorithm is carried out as a plugin for Gephi (version of 0.8.2). Gephi is an open-source visualization and exploration tool for graphs and networks [13]. Some of the features of Gephi are the application of real-time visualization and layout algorithms on graphs. Also, statistics and metrics framework helps with analysis of networks. A computer with a 64-bit Windows 7 operating system with an Intel Core i5 2.8 GHz processor and 4GB RAM is used in the experiments.

4.1 Datasets

PPI networks used in the experiments are taken from the Isobase database. Isobase is a PPI network alignment method based an ortholog database. The benchmark dataset used to evaluate the Isobase algorithm contains five eukaryotic species. We have used four of them, which are yeast, fly, worm, and human. In addition to the PPI networks, we have obtained BLAST bit scores. We have used these scores for our base layout algorithm. Since we focus on graph alignment results, we needed to apply graph alignment algorithm on these PPI networks. For this purpose, we have used the PROPER alignment algorithm [12]. PROPER is provided as an easy to use web-site where you can apply the algorithm on given PPI networks. Datasets are prepared by firstly downloading graph alignment results for PPI networks from the Isobase benchmark using the PROPER alignment algorithm. BLAST bit scores for these matchings have been identified and added to alignment. Then edges containing aligned nodes are separated. This way, we get three different edge lists for each

alignment. Edge lists for the first and second networks, and alignment edges with associated BLAST bit scores. Then, we have imported all these edges into Gephi as a single graph where alignment and the information on two graphs are included. This way we have prepared six datasets. The details of these datasets are provided in Table 4.1:

Table 4.1: Details of datasets. Number of vertices is the summation of the number of vertices of the two input graphs and the number of edges is the summation of the number of edges of the input graphs and the alignment edges.

Dataset	First network	Second network	# of Vertices	# of Edges
Ce-Dm	Worm	Fly	5512	19867
Ce-Hs	Worm	Human	5618	21340
Ce-Sc	Worm	Yeast	5492	54811
Dm-Hs	Fly	Human	13796	34462
Dm-Sc	Fly	Yeast	10188	135428
Hs-Sc	Human	Yeast	10654	153355

4.2 Evaluation of the Proposed Algorithm

In the experiments we have calculated number of edge pairs that intersect (i.e., number of edge crossings) for different layout algorithms and compared them. There are other metrics like graph area, angles between edges or the uniform distribution of vertices over the displaying plane. However, we have focused on edge crossing since the nature of the proposed algorithm is to organize alignment edges top to bottom in a linear way. Therefore, the aforementioned metrics would not be useful in evaluating the results of the algorithm.

The layout algorithms we have used for comparison are base layout algorithm, which is one of our earlier approaches to visualize graph alignment results, Force-Atlas 2 layout algorithm, and the proposed algorithm.

4.2.1 Base Layout Algorithm

Base layout algorithm is our starting heuristic approach to produce a graph layout for graph alignment results. The algorithm organizes alignment edges top to bottom on the *y*-axis from alignment edges ordered from highest BLAST score to lowest BLAST score. *x* coordinates are arranged randomly (Figure 4.1). Therefore, we do not provide any parameters for this algorithm. However, due to randomness, the algorithm is not deterministic and each time a different layout is produced. Since this is a non-deterministic algorithm, we have applied this algorithm ten times for each dataset then calculated number of edge crossings by taking average of the summation of the total number of edge crossings of ten runs.

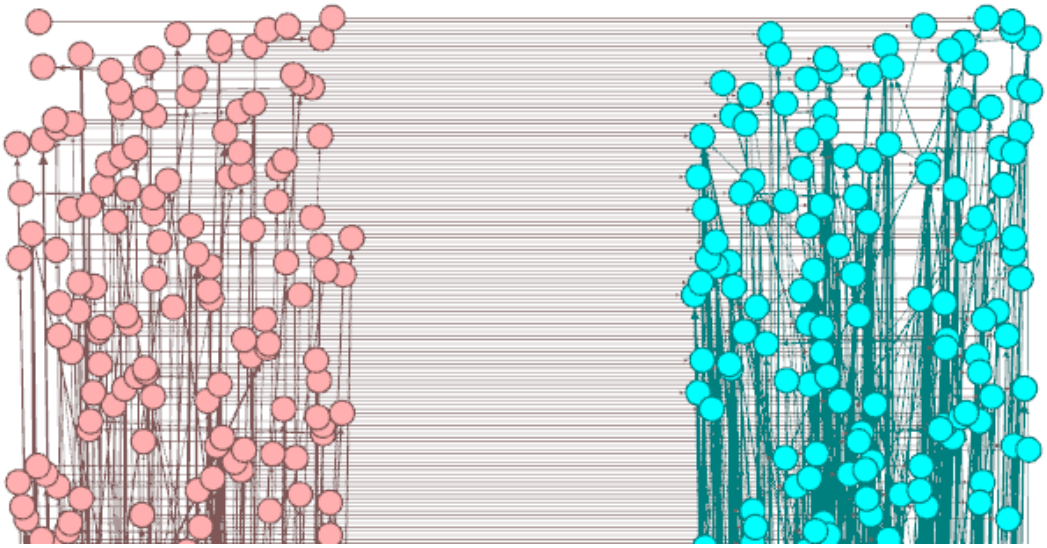

Figure 4.1: A preview of a part of the base layout algorithm applied on Ce-Dm dataset.

4.2.2 Force Atlas 2 Layout Algorithm

Force atlas 2 is a force-directed graph algorithm. It is developed by Gephi software developers. The algorithm calculates positions of vertices iteratively by applying repulsion force between nodes with no edges in between while applying attraction force between nodes connected by an edge (Figure 4.2). The algorithm also takes advantage of Barnes-Hut calculation to improve efficiency. We have used the layout

algorithm with its default options without edge weight influence score; because, edges of our datasets do not have weight information. After applying the algorithm on our datasets, we have calculated edge crossings.

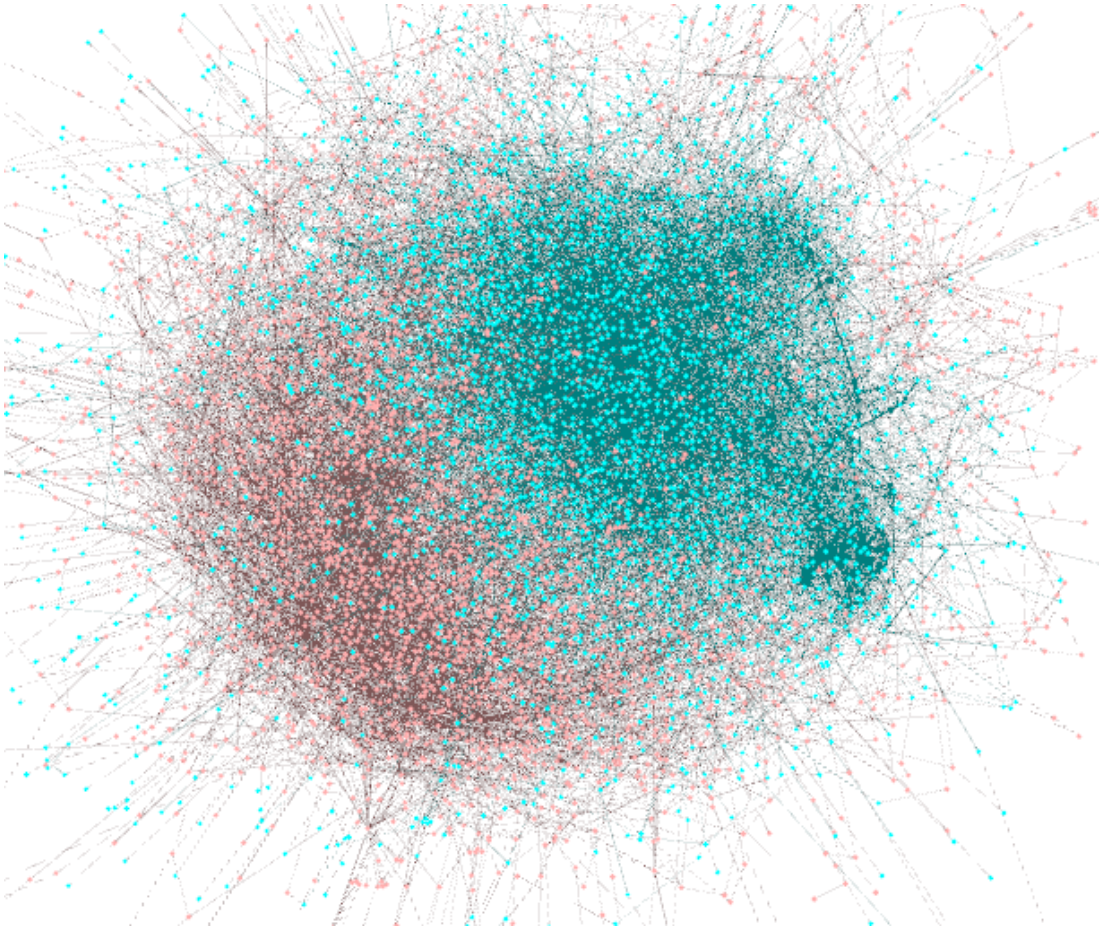

Figure 4.2: A preview of a part of force atlas 2 algorithm applied on Ce-Dm dataset.

4.3.3 The Proposed Algorithm

We provide the iteration count as the only parameter for our algorithm, other than the input alignment graph itself. In Tables 4.2 and 4.3 we have provided number of edge crossings for different iteration counts.

Table 4.2: Number of edge crossings for different iteration counts for the ce-dm dataset.

Iteration Count	Total # of edge crossings	# of edge crossings in the alignment
0	19.142.103	1.795.514
10	16.927.121	1.687.329
20	16.916.184	1.681.248
30	16.914.579	1.679.788
40	16.900.920	1.682.249
50	16.897.717	1.684.565

Table 4.3: Number of edge crossings for different iteration counts for the ce-hs dataset.

Iteration Count	Total # of edge crossings	# of edge crossings in the alignment
0	22.361.548	1.684.081
10	20.023.441	1.573.823
20	20.046.339	1.573.305
30	20.062.192	1.574.572
40	20.064.117	1.575.051
50	20.057.288	1.576.600

As we can see from Tables 4.2 and 4.3 running the algorithm with iteration count of 10 lowers number of edge crossings slightly. Increasing iteration count more than that does not affect the result much. Therefore while comparing our algorithm to the other algorithms we chose 10 as the iteration count.

4.3 Results

We have used the algorithms in Section 4.2 and applied them on our datasets. Then, we calculated the total number of edge crossings, number of edge crossings including alignment edges and the ratio of the alignment edge crossings to the total number of edge crossings for each algorithm. This way we evaluate our algorithm with different graph alignment results while comparing with other layout algorithms. In Table 4.2, Table 4.3, and Table 4.4 the number of edge crossings of each layout algorithm for each dataset is presented.

Table 4.4: Number of edge crossings after applying the Base Layout algorithm on all datasets.

	Ce-Dm	Ce-Hs	Ce-Sc	Dm-Hs	Dm-Sc	Hs-Sc
Total # of edge crossing	23 M	28 M	266M	67M	1318M	1531M
# of alignment edges that cross a graph edge	7M	7.7 M	22M	23M	104M	118M
Percentage (%)	30	28	8	34	8	7.7

Table 4.5: Number of edge crossings after applying the Force Atlas 2 algorithm on all datasets.

	Ce-Dm	Ce-Hs	Ce-Sc	Dm-Hs	Dm-Sc	Hs-Sc
Total # of edge crossings	5.9 M	7 M	85M	17M	388M	432M
# of alignment edges that cross a graph edge	1.7M	1.7 M	8M	4.8M	42M	47M
Percentage (%)	29	24	9	28	11	11

Table 4.6: Number of edge crossings after applying the proposed algorithm on all datasets.

	Ce-Dm	Ce-Hs	Ce-Sc	Dm-Hs	Dm-Sc	Hs-Sc
Total # of edge crossings	17 M	20 M	240M	40M	1237M	1333M
# of alignment edges that cross a graph edge	1.7M	1.6 M	7M	4M	31M	34M
Percentage (%)	10	8	3	10	3	3

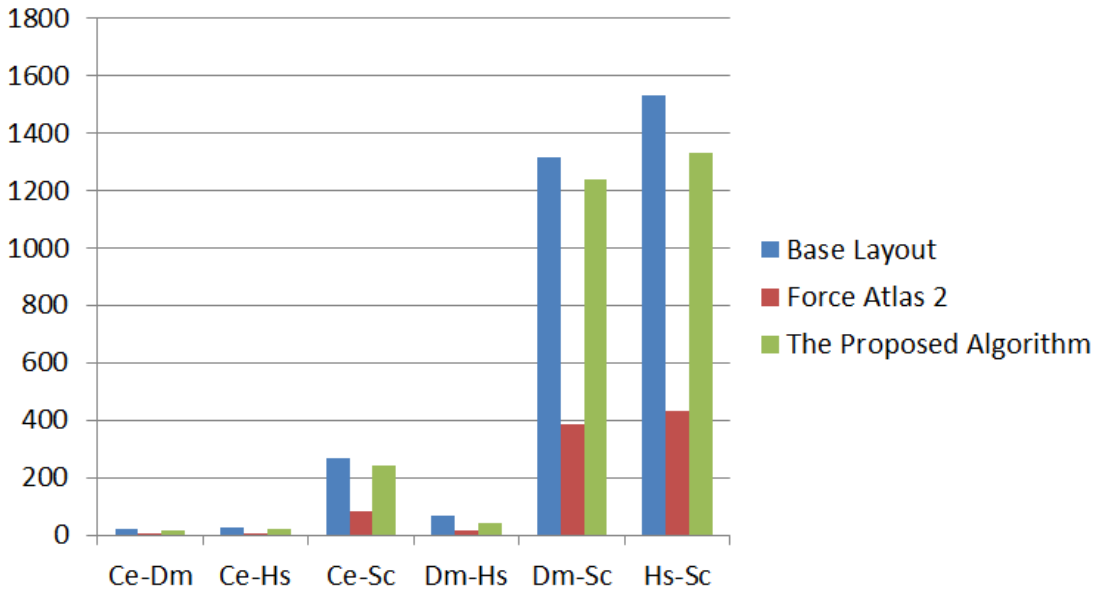

Figure 4.3: Bar graph of total number of edge crossings for applying three algorithms on six different datasets.

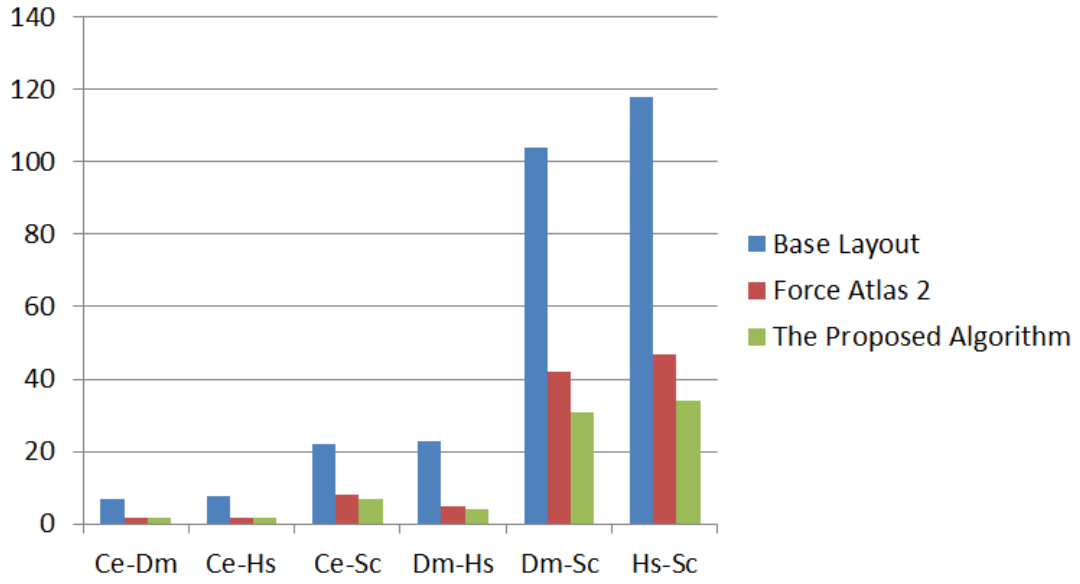

Figure 4.4: Bar graph of total number of edge crossings occurred by alignment edges for applying three algorithms on six different datasets.

As we can see from Figure 4.3, and Figure 4.4 for all test cases the highest number of edge crossings occurs in the Base Layout Algorithm. The proposed algorithm has the second highest number of total edge crossings, while Force Atlas 2 has the least amount of total edge crossings.

As we have expected our algorithm produced a better result compared to our base layout algorithm. Organizing edges with respect to pairwise sequence similarity (i.e., BLAST bit scores) without considering the underlying graph topology does not lower the number of edge crossing. As we can see, applying mean iteration improves the results by grouping connected nodes together.

Comparing Force Atlas 2 results with the proposed algorithm, we can see that Force Atlas 2 has provided a better result considering the number of edge crossings. This is expected since the nature of force directed graph layout algorithms pulls adjacent nodes closer. Since the count of alignment edges are quite low compared to the number of edges in the graphs themselves, edges from graphs improves attraction forces, which leads to separation of aligned graphs. However, as we can see, alignment edges cross a higher number of edges. Our algorithm may result in a higher number of total edge crossings, but it focuses on lowering the number of edge

crossings on alignment edges. From Tables 4.2, 4.3 and 4.4 it is easy to notice that the ratio of the number of edge crossings on alignment edges to total number of edge crossings is minimum for our proposed algorithm. Since our main focus was providing a layout algorithm for graph alignment we tried to display the graph alignment result in a better way, in this case with lower number of edge crossings. From the experiments for alignment visualization, we may conclude that the proposed algorithm achieves the specified objectives.

4.4 Applying Initialization

After experimenting with six different datasets and three different layout algorithms, we have carried out an extra experiment to see if we can improve the results. In our proposed algorithm, nodes are initially placed one after another without any specific order. Therefore, in the preprocessing, we apply an initialization to see how it affects the result of our algorithm.

As initialization we have used three approaches, which are using no specific order, same as our earlier algorithm, ordering with respect to degrees, and ordering with respect to clusters which are calculated using Markov Cluster Algorithm. We have also tried to increase iteration count after initialization, however, similar to the earlier experiments, after ten iterations, number of edge crossings do not change much. We have given the experiment results in Table 4.7.

Table 4.7: Number of edge crossings after applying initialization and the proposed algorithm on Ce-Dm dataset.

	No Iteration (Only Initialization)			10 Iteration		
Initialization	Cluster	Degree	None	Cluster	Degree	None
Total # of edge crossings	16.8M	22M	19M	16M	17.7M	17
# of alignment edges that cross a graph edge	1.5M	1.6M	1.8M	1.5M	1.7M	1.7
Percentage (%)	9	7	9	9	9	10

As we can see from Table 4.7, initialization by degree causes total number of edge crossings to increase in both before and after applying the algorithm itself. Initialization with no specific order gives fewer number of edge crossings compared to ordering with degrees. However, best results are produced from initialization with clusters. Using clusters slightly improves the results of the proposed algorithm. Also, we can notice that even only using initialization on the dataset produces fewer number of edge crossings compared to our earlier experiments.

4.5 Experimenting Small Graphs

The datasets we have used have thousands of nodes and edges. Therefore, we wanted to see the results using smaller graphs. For this purpose, we have generated two synthetic graph alignment data. One of them is alignment of two planar graphs, while the other one is alignment of non-planar graphs.

The dataset of non-planar graphs have 40 nodes and 104 edges. We have applied three algorithms we have used in experimenting. Resulting views of applying algorithms are provided in Figure 4.5, 4.6 and 4.7. Also, in Table 4.8 numbers of edge crossings are given.

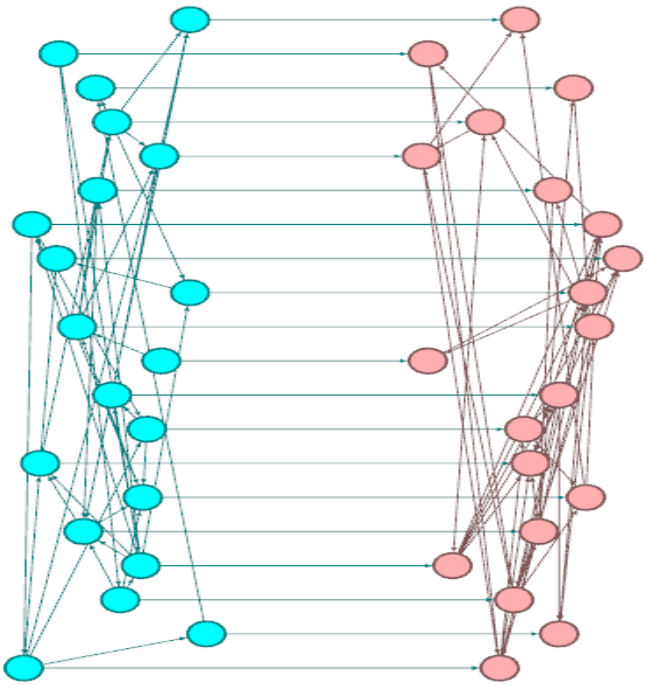

Figure 4.5: View of base layout algorithm applied on a small dataset.

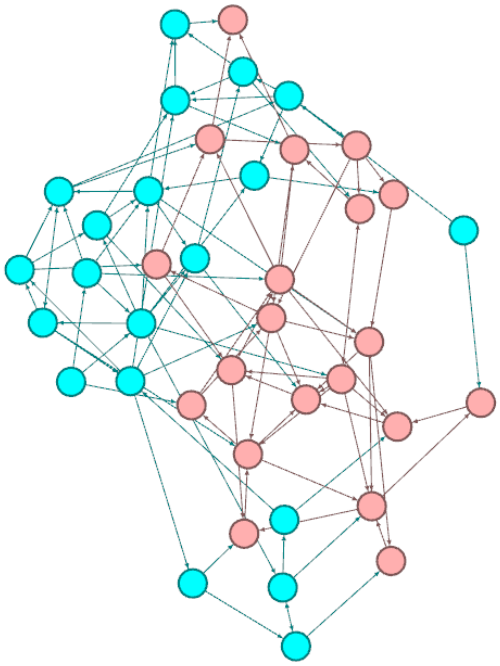

Figure 4.6: View of force atlas 2 algorithm applied on a small dataset.

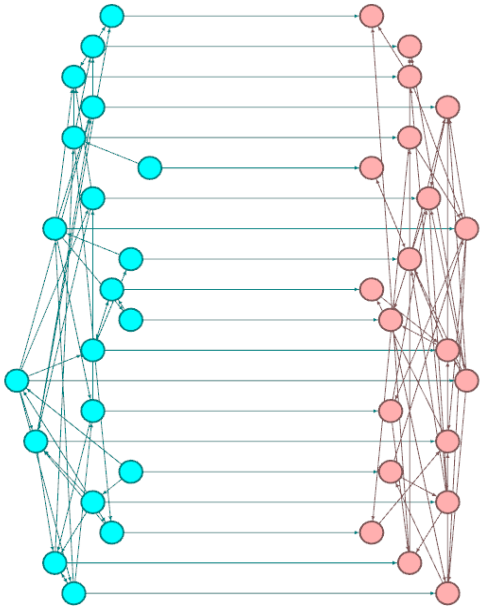

Figure 4.7: View of the proposed algorithm applied on a small dataset.

Table 4.8: Number of edge crossings after applying layout algorithms on a small dataset.

	Base Layout	Force Atlas 2	The Proposed Algorithm
Total # of edge crossings	1081	707	889
# of alignment edges that cross a graph edge	418	282	307
Percentage (%)	39	40	35

Node placement of the base layout algorithm and the proposed algorithm looks similar, since both come from the same starting point. It can be hard to tell the difference between them at the first sight. However, as we can see from the edge crossing numbers, the proposed algorithm provides slightly a better result. Also, while edge crossing numbers claim that force atlas 2 provides the best result among

the three layout algorithm, when we look at the Figure 4.6 alignment information is not as obvious as the other algorithms.

The dataset of planar graphs have 42 nodes and 97 edges. We have applied three algorithms we have used in experimenting. Resulting views of applying algorithms are provided in Figure 4.8, 4.9 and 4.10. Also, in Table 4.9 numbers of edge crossings are given.

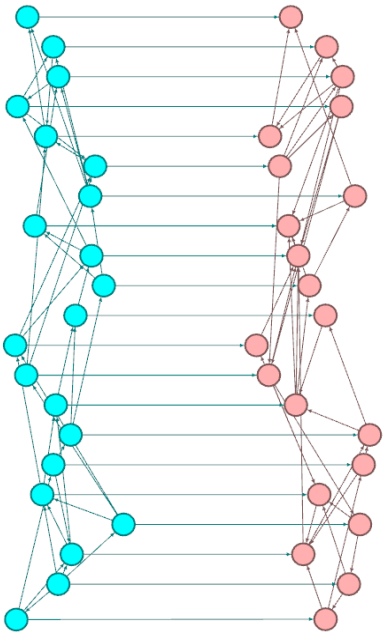

Figure 4.8: View of base layout algorithm applied on alignment of planar graphs.

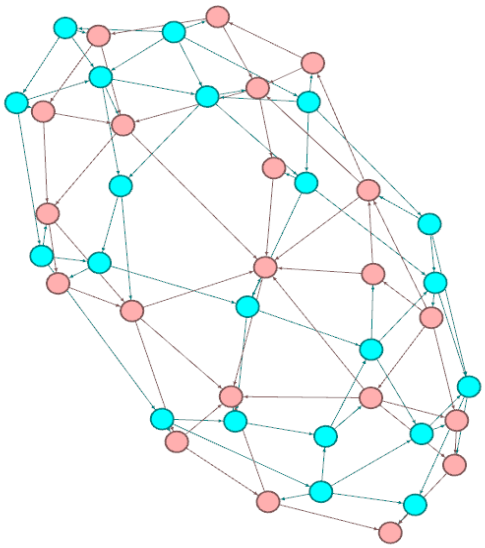

Figure 4.9: View of force atlas 2 algorithm applied on alignment of planar graphs.

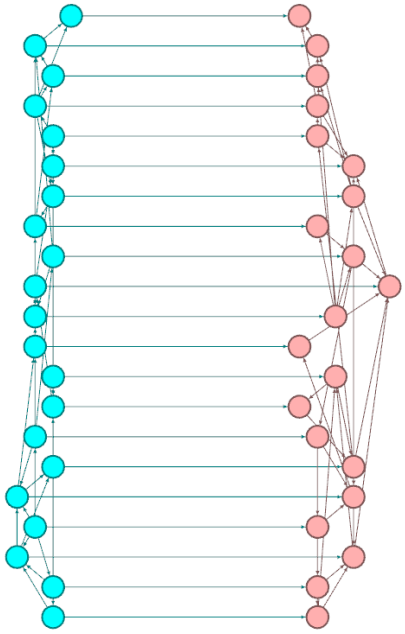

Figure 4.10: View of proposed algorithm applied on alignment of planar graphs.

Table 4.9: Number of edge crossings after applying layout algorithms on alignment of planar graphs.

	Base Layout	Force Atlas 2	The Proposed Algorithm
Total # of edge crossings	650	431	567
# of alignment edges that cross a graph edge	320	156	220
Percentage (%)	49	36	39

In terms of node placement, base layout and the proposed algorithm provide similar results whether graphs are planar or non-planar graphs. However, force atlas 2 provides a different view. In Figure 4.9, aligned nodes are almost next to each other. Since, connectivity of the nodes decide node positioning, this view was expected for the planar graph generated with small number of node degrees. Also, Table 4.9 shows that force atlas 2 has better results comparing to the other layout algorithms.

CHAPTER 5

CONCLUSION

5.1 Summary

Graphs are commonly used in bioinformatics to represent biological networks. Visualization of the networks plays a significant role in understanding them. Layout algorithms are developed for this purpose, visualization of a single graph. However, some studies on the biological networks require these networks to be aligned. This is why a layout algorithm for visualizing alignment of networks is required.

There are existing layout algorithms to visualize networks. However they usually run on a single network. Some of them considers the graph like a physical system, such as the force-directed layout algorithm. Some of the layout algorithms use biological data as edge weights. These approaches calculate positioning of vertices as a single graph. Therefore, a new approach is needed to handle graphs separately while considering alignment information or edges. In this study, we have proposed a new layout approach where we focus on providing a better visualization of alignment edges.

Our algorithm firstly separates aligned graphs into left and right side of the plane, from top to bottom. Then applying mean iteration iteratively, the algorithm calculates new positions of aligned nodes and moves nodes with respect to the new position. Then x-coordinates are arranged by the degree of the vertices. Finally, experiments are carried out on different datasets with three different layout approaches. The numbers of edge crossings are analyzed for each type of layout algorithm. Overall the number of edge crossings is lowest in the force-atlas 2 layout algorithm;

however, alignment edges cross each other the least in the proposed algorithm, hence resulting in a better presentation of the graph alignment.

5.2 Future Work

In the proposed algorithm, since the two graphs are aligned, input graphs are expected to be similar. In other words, we expected to find a similar layout displayed as a result of applying mean iteration on either of the input graphs. This may not be always the case. Therefore, an approach without the assumption of similar behaviors of aligned nodes can be experimented to see whether it provides a better result or not.

Applying initialization, depending on the way of ordering, may lower the number of edge crossings. From the experiment, MCL clustering seems to be the better approach in ordering. However, clusters are calculated using a single graph and ordering is carried out using the cluster information from one of the input graphs. Individual clustering results are not similar to each other; therefore, clustering can be calculated using both input graphs. Also, another graph clustering algorithm can be applied to improve ordering.

The proposed algorithm can also be applied in a two-level approach. After initial clustering, at first level, clusters can be used as single nodes and the proposed algorithm can be applied on these cluster-nodes. In the second level, the proposed algorithm can be applied for each of the clusters. This way, the clusters will still be grouped together while inner arrangements are adjusted.

Graph visualization can be harder to read with increasing number of vertex count. Therefore, using metanodes , which consist of a group of vertices, can be helpful while studying graph alignment results. These metanodes can be decided using different approaches. For instance, we can use biological data to group certain vertices. Also, we can use clustering algorithms. We may come up with a certain clustering algorithm where we try to decide metanodes in a way that edge numbers between metanodes are under a certain threshold. Once we figure out the metanodes we can apply the proposed algorithm.

In the proposed algorithm we have organized nodes top to bottom vertically. However, this approach does not allow us to use the display window efficiently. Therefore to be able to use the display window more efficiently, another node organization can be used. However, if we do not use our vertical view, then we will need to find another alignment edge placement since we cannot simply use horizontal edges. For instance, we can try to provide two networks as circles one is inside the other one. Alignment edges can be between two circles. For the inner network, edges that belong to the network are can be drawn in the inner area of the circle while for the other network we can use the outer side of the circle to draw edges that belong to the network.

REFERENCES

[1] T. M. Fruchterman and E. M. Reingold, “Graph drawing by force-directed placement,” Softw., Pract. Exper., vol. 21, no. 11, pp. 1129–1164, 1991.

[2] Y. Hu, "Efficient, high-quality force-directed graph drawing." *Mathematica Journal* 10.1: 37-71, 2005.

[3] E. Kazemi, H. Hassani, M. Grossglauser, & H. P. Modarres. PROPER: global protein interaction network alignment through percolation matching. *BMC bioinformatics*, *17*(1), 527, 2016.

[4] H. F. Danacı, A Customized Force-Directed Layout Algorithm For Biological Graphs Whose Vertices Have Enzyme Commission Attributes. MSc thesis, Middle East Technical University, Ankara, Turkey, 2015.

[5] M. Jacomy, S. Heymann, T. Venturini, & M. Bastian, Forceatlas2, a continuous graph layout algorithm for handy network visualization. *Medialab center of research*, *560*, 2011.

[6] M. Koyutürk, Y. Kim, S. Subramaniam, W. Szpankowski, and A. Grama, “Detecting conserved interaction patterns in biological networks.” *J Comput Biol*, vol. 13, no. 7, pp. 1299–1322, September 2006.

[7] R. Sharan, S. Suthram, R. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. Karp, and T. Ideker, “Conserved patterns of protein interaction in multiple species.” *Proc Natl Acad Sci U S A*, vol. 102, no. 6, pp. 1974–1979, 2005.

[8] S. Brasch, L. Linsen, & G. Fuellen. Visualization of aligned biological networks: a survey. In *Cyberworlds, 2007. CW'07. International Conference on* (pp. 49-53). IEEE, October 2007.

[9] S. Bandyopadhyay, R. Sharan, and T. Ideker, “Systematic identification of functional orthologs based on protein network comparison,” *Genome Res.*, vol. 16, no. 3, pp. 428–435, March 2006.

[10] Z. Hu, J. Mellor, J. Wu, M. Kanehisa, J. M. Stuart, and C. Delisi, “Towards zoomable multidimensional maps of the cell,” *Nature Biotechnology*, vol. 25, no. 5, pp. 547–554, May 2007.

[11] Y. Koren, & D. Harel “A multi-scale algorithm for the linear arrangement problem.”, http://www.wisdom.weizmann.ac.il/~/dharel/papers/min_la.pdf , Accessed: 2015-12.

[12] “The PROPER Algorithm” http://proper.epfl.ch/ , Accessed: 2017-01-05.

[13] M. Bastian, S. Heymann, & M. Jacomy. Gephi: an open source software for exploring and manipulating networks. *ICWSM*, *8*, 361-362, 2009.

[14] S.v. Dongen, Graph Clustering by Flot Simulation. PhD thesis, University Utrecht, May 2000.

[15] C.Erten, S.G.Kobourov, V.Le, & A.Navabi “Simultaneous Graph Drawing: Layout Algorithms and Visualization Schemes”, Journal of Graph Algorithms and Applications, vol.9, no.1, pp. 165-182 2005.