International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Grid Based Straight-Line Layout Algorithm for Clustered Biological Networks

Noopur Landge[1], Manikrao Dhore[2], Pushkar Jogalekar[3], Vrushali Inamdar[4]

[1]M.Tech. student, Vishwakarma Institute of Technology, Pune, India

[2]Professor, Dept. of Computer Science, Vishwakarma Institute of Technology, Pune, India

[3]Asst. Professor, Dept. of Computer Science, Vishwakarma Institute of Technology, Pune, India

[4]Team Lead, Persistent Systems Limited, Pune, India

Abstract: ***In this paper we give a new layout algorithm for Biological Networks. Clusters in the Biological Networks impose additional constraint on the structure of underlying graph and traditional graph layout algorithms are not directly useful for laying out such networks. We propose a grid based straight line drawing algorithm for laying out clustered graphs. Another novel feature of our algorithm is: it allows nodes of suitable sizes and shapes as per the requirement which makes the algorithm more suitable for laying out Biological Networks.***

Keywords: Graph Layout, Grid Based Straight-Line Drawing, Clustered Graph, Biological Networks, Grid Based Layout.

1. Introduction

Graph layout is an area of mathematics combining methods from geometric graph theory and information visualization. It is used to determine 2-dimensional representations emerging from applications such as bioinformatics, social network analysis, and cartography. A graph layout (graph drawing) is a pictorial representation of the vertices and edges of a graph. However, the term graph layout is different from the term graph, since a graph can be represented with different layouts. The arrangement of these vertices and edges in a layout affects its understandability, usability, and aesthetics.

In first section, we discuss various factors that affect the graph aesthetics and preliminary terms related to the graph layouts. In the next section, we discuss the biological networks and role of the graph drawings in representation of biological networks. We also state the existing graph layout techniques. Also we state the challenges posed by clustered biological networks while laying out. In third section, we discuss the new algorithm which tries to cover the limitations of existing ones. Further sections analyze the new algorithm and discuss the results.

1.1 Graph Aesthetics

Many different quality measures have been defined for graph drawings, in an attempt to find objective means of evaluating their aesthetics and usability. In addition to guiding the choice between different layout methods for the same graph, some layout methods attempt to directly optimize these measures.

The *crossing number* of a drawing is the number of pairs of edges that cross each other. Sometimes, removing all crossings is not possible. In such cases minimizing the number of crossings is desired.

The *area of a drawing* is the size of its smallest bounding box, relative to the closest distance between any two vertices. The *aspect ratio* of the bounding box may also be important.

Symmetry display is the problem of finding symmetry groups within a given graph, and finding a drawing that displays as much of the symmetry as possible.

It is important that *shapes of edges* are as simple as possible, to make it easier for the eye to follow them. The shape may be straight line segments or curved lines.

Length of edges is also one of the factors. It is generally desirable to minimize the total length of the edges as well as the maximum length of any edge. Additionally, it may be preferable for the lengths of edges to be uniform rather than highly varied.

1.2 Basic Terminology

Planar graph: A planar graph is a graph that can be embedded in the plane in such a way that its edges intersect only at their endpoints.

Non-planar graph: It is a graph which cannot be drawn on a plane without the edges being intersected at points other than its endpoints.

Maximal planar graph: A simple graph is called maximal planar if it is planar but adding any edge on the given vertex set would destroy that property. All its faces including the outer one are then bounded by three edges.

Dual of a graph: The dual graph of a plane graph G is a graph that has a vertex corresponding to each face of G, and an edge joining two neighboring faces for each edge in G.

2. Biological Networks and Existing Work

2.1 Biological Networks

A biological network is any network that applies to biological systems. A network is any system with sub-units that are linked into a whole, such as species units linked into a whole food web. Biological networks provide a mathematical analysis of connections found in ecological, evolutionary, and physiological studies, such as neural networks.

Biological processes are widely used in the form of networks such as metabolic pathways, protein-protein interaction networks. The study of biological networks, their modeling, analysis, and visualization are important tasks in life science today [1]. An understanding of these networks by the scientists and researchers is essential to make biological understanding of such complex data, and large amount of this data is being generated now. But most biological networks are usually difficult to interpret due to the complexity of the relationships and the peculiarities of the data. Network visualization is a fundamental method that helps scientists in understanding biological networks and in uncovering important properties of the underlying biochemical processes.

2.1.1 Visualization requirements of Biological Networks

Pathways: The direction of the processes should be clearly visible using directed edges to express the order of the events.

Compartments/Clusters: The elements of biological networks (nodes, other compartments and edges) are grouped in visual and functional compartments and this information should be visually represented. There can be a nesting of compartments and the entities part of a compartment should be belong to same compartment even after applying layout.

Species: Species are nodes and they can belong to a single compartment at a single point.

Parts of reactions: The substances, i.e. reactants, products, and enzymes, of a reaction should be visible and identifiable. Usually for main substances their name, structural formula or both should be shown.

Reactions: The reaction arrow(s) should be shown from the reactants to the products with enzymes placed on one side of the reaction arrow and co-substances on the opposite side.

2.2 Grid Based Straight-Line Drawings

We consider the problem of embedding the vertices of a planar graph into a small grid in the plane in such a way that the edges are straight, non-intersecting line segments. By applying grid based straight-line drawing on biological network, we can have even distribution of vertices over the area with fewer edge-crossings. This will increase the readability of the complex graph.

A criterion to improve the aesthetic quality is convexity: every interior face must be drawn convexly. Tutte [2] showed that every tri-connected planar graph can be drawn with convex interior faces. Thomassen [3] characterized the class of planar graphs which admit a convex drawing, and Chiba et. al. [4] presented a linear-time drawing algorithm for this class. However, the coordinates of the vertices can be real numbers with fractions and a huge number of vertices can be clustered in a small area.

The existence of such straight-line embedding for planar graphs was independently discovered by Fary [5], and Stein [6]. The first algorithms for constructing straight-line-embeddings required high-precision arithmetic, and the resulting drawings were not very aesthetic, since they tend to produce uneven distributions of vertices over the drawing area.

Rosenstiehl and Tarjan [7] noticed that it would be convenient to be able to map vertices of planar graph into a small (polynomial sized) grid, because then high-precision operations would be unnecessary. The question whether such embeddings are possible or not was left open.

This problem was solved by de Fraysseix, Pach and Pollack [8], [9] who proved that, for n>=3, each n-vertex planar graph can be drawn on the (2n-4)x(n-2) grid. They also presented an O(nlogn) time algorithm for constructing such embedding, but left open the problem of whether it is possible to find such an embedding in linear time.

Chroback and Payne [10] answer this question in the affirmative by presenting a linear-time implementation of the technique from Fraysseix, Pach and Pollack. Although they base their algorithm on the general method from previous proof, the algorithm itself differs significantly from the previous one. Also, unlike the previous algorithm, their method does not require any sophisticated data structures. Instead, it distributes and carefully manages the information needed in the algorithm. This algorithm is easier to implement and the resulting embedding tend to be more aesthetic.

Kant [11] introduces independently an ordering on the vertices and faces of a tri-connected planar graph, called the canonical ordering. He also proves the existence of canonical ordering for every planar graph. The canonical ordering can be computed in linear time. Refining the canonical ordering to a leftmost canonical ordering leads to a general framework for drawing tri-connected planar graphs on a grid, and implies several drawing results.

2.2.1 The algorithm by Chroback-Payne

The algorithm embeds G, one vertex at a time in canonical order, at each stage adjusting the current partial embedding.

Initially put first three vertices in the canonical order at points (0,0), (1,0), and (1,1). Let C=(W1,…,Wm) be the contour of current embedding. Vk be the next vertex in the canonical embedding. We try to add vertex Vk to this embedding. In this contour, let Vk is adjacent to vertices Wp,…,Wq, in the original graph. Now we shift the vertices

in current embedding such that,

1. Shift Wq,…,Wm to right by 2 locations
2. Shift Wp+1,…,Wq-1 to right by 1 location

Then find the point of intersection of lines through Wp and Wq with slopes +1 and -1 respectively. This point gives the position of Vk in the embedding. The process is repeated for all vertices in canonical order.

Wolf [12] in his thesis presents a polynomial time algorithm for converting a non-planar graph to a planar graph. He does it by converting the planarization problem to vertex insertion problem. The solution suggested is based on heuristic approach. Further, Wolf suggests using permutation heuristics, shortest first heuristics, remove and reinsert heuristics for improving the quality of results while minimizing edge crossings.

2.3 Challenges Posed by Biological Networks

As discussed in previous section, the previous research has been done on drawing layout for maximal planar graphs. Also researchers have proposed that any simple non-clustered graph can be converted to a planar graph in polynomial time and a planar graph can be converted to maximal planar graph in linear time. Thus a non-planar graph can be plotted on a grid in overall polynomial time.

But in real world, the problem is to draw graphical representation of complex structures of biological or molecular structures. These structures are most of the time clustered in nature. Thus a layout method for these graphs is needed.

The researchers have given some ways to convert simple non-planar graphs to maximal planar graphs. But they do not talk about converting them back to the original form after being plotted on a grid. They do not talk about the effect of removal of dummy edges and dummy vertices in the resulting graph. Clearly, it will meet the criteria for grid based layout but it will be interesting to see if it can be further compacted to fit in smaller area without violating the criteria.

Thus we observe that there is a scope for developing layout algorithms that can work with biological network. These algorithms can prove to be useful for the researchers in the field of systems biology.

3. Proposed Algorithm for Clustered Graphs

3.1 Modified Chroback-Payne Algorithm with Node-Size Consideration

The algorithm given by Chroback-Payne considers vertices in a graph as points without any dimensions. It places these vertices on a grid with minimum separating distance as 1 unit; whereas biological networks can have nodes with some finite dimensions. These dimensions need to be retained in the laid out network without nodes overlapping each other. To achieve this, we modify the algorithm in following way:

Consider we have a partial layout G' of a graph G=(V,E). G'=(V',E'), V'⊂V, E'⊂E. First k-1 vertices ($V_1 \ldots V_{k-1}$) from canonical order O are added to G'. Now we try to add V_k to G'.

Let ($W_1 \ldots W_p \ldots W_q \ldots W_m$) be the current contour in G' such that V_k is neighbor to $W_p \ldots W_q$ in G. We insert V_k according to Chroback-Payne. At this stage, V_k can overlap other vertex in G' horizontally if it is at same level as V_k. Thus V_k can overlap horizontally only with $W_1 \ldots W_{p-1}$ and $W_{q+1} \ldots W_m$.

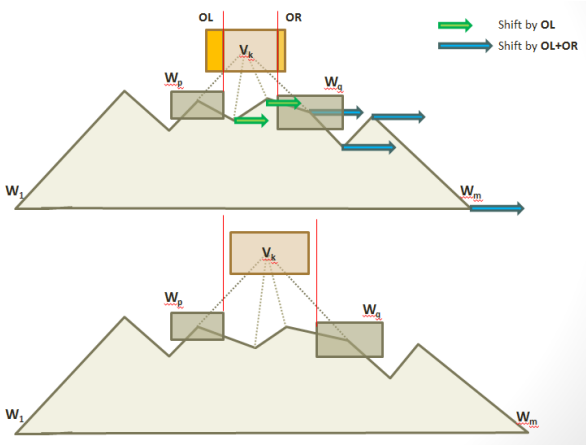


Figure 1: Horizontal overlap removal

This follows from the fact that V_k is always placed above its neighbors. Hence, if we shift the vertices so as to place V_k between W_p and W_q, we ensure that there are no other overlaps horizontally (Fig 1).

Now, since V_k is placed between C_p and C_q, it can overlap only $W_p \ldots W_q$. hence by avoiding overlaps among V_k and $W_p \ldots W_q$, we ensure all vertical overlaps are avoided (Fig 2).

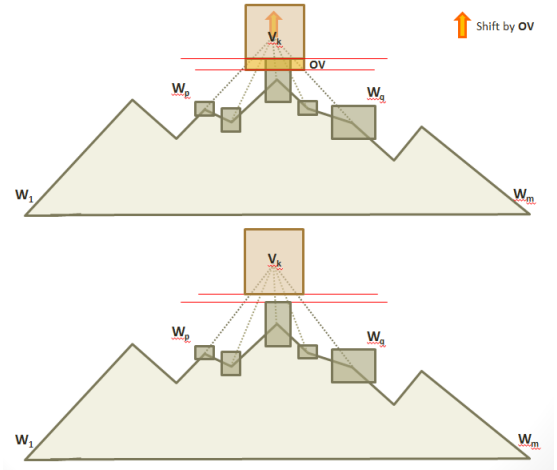


Figure 2: Vertical overlap removal

Algorithm 1: Straight-Line Drawing with Node-size Consideration

function removeHorizontalOverlap(C, p, overlapR, overlapL)

begin

foreach (v ∈ associated[C_{p+1}])

shift v to right by overlapL;

```
    end
    foreach (v_i ∈ associated[C_{p+2}])
        foreach (v_j ∈ associated[v_i])
            shift v_j to right by overlapL+overlapR;
        end
    end
end
function removeVerticalOverlap(V_k, V_t)
begin
    overlapV←amount of vertical overlap between V_k and V_t;
    Shift V_k up by overlapV;
end
function gridBasedLayout(G, O):
begin
initialize position[V_1]←(0,0), position[V_2]←(0,0);
initialize contour C←(V_1, V_2) where O = (V_1,…,V_n);
initialize associated[V_i]←V_i
    overlapL← horizontal overlap between C_1 and C_2;
removeHorizontalOverlap(C, 1, overlapL, 0);
    foreach (V_k ∈ (O[3]…O[n])
        p ←index of first neighbor of V_k in C;
        q ←index of last neighbor of V_k in C;
        V_t ←topmost vertex in (W_{p+1}…W_{q-1});
        foreach (v ∈ (W_{p+1}…W_{q-1}))
            position[v] ← position[v]+1;
        end
        foreach (v ∈ (W_q…W_m))
            position[v] ← position[v]+2;
        end
        position[V_k] ←μ(position[W_p], position[W_q]);
        associated[V_k] ←{V_k} ∪ associated[W_{p+1}] ∪…
                                …∪ associated[W_{q-1}];
        C← (W_1…W_p,V_k,W_q,…W_m);
        overlapL← amount of horizontal overlap between V_k
                  and its first neighbor in C;
        overlapR← amount of horizontal overlap between V_k
                  and its last neighbor in C;
        removeHorizontalOverlap(C, p, overlapL, overlapR);
        position[V_k] ← μ(position[W_p], position[W_{p+2}]);
        removeVerticalOverlap(V_k, V_t);
    end
end
```

Here, $\mu(P_1,P_2) = \frac{1}{2}(x_1-y_1+x_2+y_2\ ,\ -x_1+y_1+x_2+y_2)$ gives grid-point on intersection of lines with slopes +1 and -1 passing through points with positions P_1 and P_2.

3.2 Obtaining Maximal-Planar Graph from Planar Graph

Algorithm 2 is based on the algorithm given by Chroback-Payne. It finds straight-line drawings of a maximal-planar graph. To be able to layout any planar graph, we obtain a maximal-planar graph from it. To do this, we simply go on adding all possible edges not present already in the graph. We remember them so that they can be removed after layout.

Algorithm 2: 3.2 Obtaining Maximal-Planar Graph from Planar Graph

```
function toMaximalPlanar(G, dummyEdges)
begin
    G'← copy of G with no edges
    foreach (e in G)
        add e to G';
        if (G' is planar)
            continue;
        else
            dummyEdges←remove e from G';
        endif
    end
end
```

3.3 Obtaining Planar Graph from Non-Planar Graph

For this conversion step, we use the algorithm given by Wolf. This problem is NP-Hard. We use basic heuristics given by Wolf to solve the problem in polynomial time. For planarization step, we create a copy of the non-planar graph without any edge. Then go on adding each edge one by one unless and until the graph is planar. This gives a planar subgraph of given graph. We treat the problem as Vertex Insertion Problem with Fixed Embedding (VIP-FIX). After solving this problem, using the algorithm by Wolf, we get a planar graph which contains some extra vertices which replace the edge crossings.

3.4 Layout of Non-Clustered Graph

This algorithm treats input graph as a non-planar graph without any clusters in it. To layout any non-planar graph on grid using straight line drawing, we use algorithms discussed in previous sections. We first obtain a planar graph corresponding to given graph by adding dummy vertices to the graph. A maximal-planar graph is obtained from the resultant planar graph by adding dummy edges. The canonical order of vertices in this maximal-planar graph is obtained as described by Kant. We then apply modified straight-line drawing algorithm (see algorithm 1) to the obtained maximal-planar graph. After the layout, we remove added dummy edges and the dummy vertices to get the original graph back.

Algorithm 3: Layout of Non-Clustered Graph

```
function layoutNonClusteredGraph(G)
begin
    dummyVerteces←palnarizeByWolf(G);
    dummyEdges←toMaximalPlanar(G);
    O ← caononicalOrderByKant(G);
    gridBasedLayout(G, O);
    foreach (e in dummyEdges)
        remove e from G;
    end
    foreach (v in dummyVertices)
        remove v from G;
    end
end
```

3.5 Layout of Clustered Graph

The basic idea is to apply algorithm 3 recursively to the clusters in a clustered graph. In our implementation, we follow bottom-up approach. Starting with laying out innermost cluster, we go on laying out outer clusters in the hierarchy.

At any level in this clustered hierarchy, we create a non-clustered representation of that graph. To do this, we replace its children clusters by nodes of same dimensions. The edges are also adjusted so as to end at this new set of nodes. Now the graph is a non-clustered graph and algorithm 3 can be applied to it. After this new graph is laid out, we replace the new nodes by their corresponding clusters (Fig 3). These clusters are already laid out by calling algorithm 4 recursively.

Algorithm 4: Layout of Clustered Graph

```
function layoutClusteredGraph(G)
initialize G' as empty graph;
begin
    foreach (cluster c in G)
        layoutClusteredGraph(c);
        add node to G' with same size as c;
    end
    foreach (outer node v_o in G)
        add node to G' with same size as v_o;
    end
    foreach (e in G)
        if (e comes out or goes in a cluster c)
            add edge to G' incident on node corresponding to
                c;
        elseif (e lies completely outside all clusters)
            add edge to G' incident on nodes corresponding to
                its endpoints;
        elseif (e lies completely inside a cluster)
            ignore e;
        endif
    end
    layoutNonClusteredGraph(G');
    foreach (v in G')
        if (v represents a cluster c)
            offset ← top-left-coordinates[v];
            foreach (v_c in c)
                position[v_c] ← position[v_c] + offset;
            end
        elseif (v represents outer vertex v_o)
            position[v_o] ← position[v];
        endif
    end
end
```

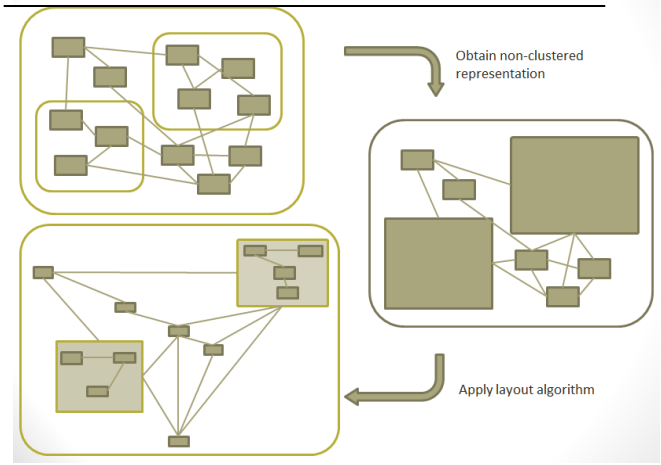


Figure 3: Layout of Clustered Graph

4. Analysis and Observations

4.1 Computational Complexity

The algorithm consists of four major steps, namely, conversion to planar graph, conversion to maximal planar graph, finding canonical order of vertices and applying the modified layout algorithm (see algorithm 3). Out of these four steps, the conversion of graph to planar graph tries to solve an NP-hard problem using heuristic approach. Also, this step becomes bottleneck step of the algorithm.

We start this step by constructing a planar subgraph. Its Boost implementation takes time $O(E^2+EV)$. According to Wolf, when the algorithm considers fixed embedding, vertex insertion takes $O(V^2.log\ V)$ for each edge that makes it non-planar. Thus the conversion to planar graph takes $O(E^2+EV^2.log\ V)$.

Algorithm 2 checks the planarity of the graph after adding each edge. Thus the time required is $O(E(E+V))$.

The Boost Graph Library provides canonical order of vertices in time $O(E+V)$.

Also the algorithm by Chroback-Payne draws the layout in linear time $O(V)$. We modify it to consider vertex sizes. These steps also require $O(deg(V_k))$. Thus the complexity of algorithm 1 becomes $O(V)$.

The overall complexity of algorithm 3 becomes,

$$O(E^2+EV^2 log\ V) \quad (1)$$

4.2 Bounding Area of Non-Clustered Layout

We give bounds on the height and width of output graph in Algorithm1. Let maximum width of node in graph G be W, and maximum height of node in graph G be H. Consider G consists of n number of nodes.

From Algorithm 1, V_1 and V_2 are initially placed at distance equal to OH+S. OH be amount of horizontal overlap and S be amount of separation required between nodes. Here, OH=W. Thus,

$$\text{Initial distance between } V_1 \text{ and } V_2 = W+S \quad (2)$$

For next (n−2) vertices, every time we add a vertex, V_2 is shifted right by distance OL+OR+2S. OL be amount of horizontal overlap with C_p and OR be amount of horizontal overlap with C_q. These overlaps are maximum when C_p and C_q are closest to each other, i.e. at distance W+S. In this case, values of OL and OR will be $\frac{W}{2}-\frac{S}{2}$. Thus,

Each time V_2 is shifted right by distance = W+S …(3)
Total displacement of V_2 from V_1 is given by (2) + (3) as

$$(n-1)(W+S)$$

And width of graph becomes,

$$(n-1)(W+S) + W \ldots(4)$$

Also, height of the graph is worst when every vertex is placed vertically one above other. In this case, vertical

distance between vertices is OV+S. OV be amount of vertical overlap with its lower vertex. Here OV=H. After adding (n−2) vertices, Distance between lowest and topmost vertex becomes,

$$(n-2)(H+S)$$

And height of graph becomes,

$$(n-2)(H+S) + H \quad \ldots(4)$$

Thus, the resulting laid out graph is bounded by area,

$$((n-1)(W+S) + W) \times ((n-2)(H+S) + H) \quad \ldots(5)$$

If you are using *Word*, use either the Microsoft Equation Editor or the *MathType* add-on (http://www.mathtype.com) for equations in your paper (Insert | Object | Create New | Microsoft Equation *or* MathType Equation). "Float over text" should not be selected.

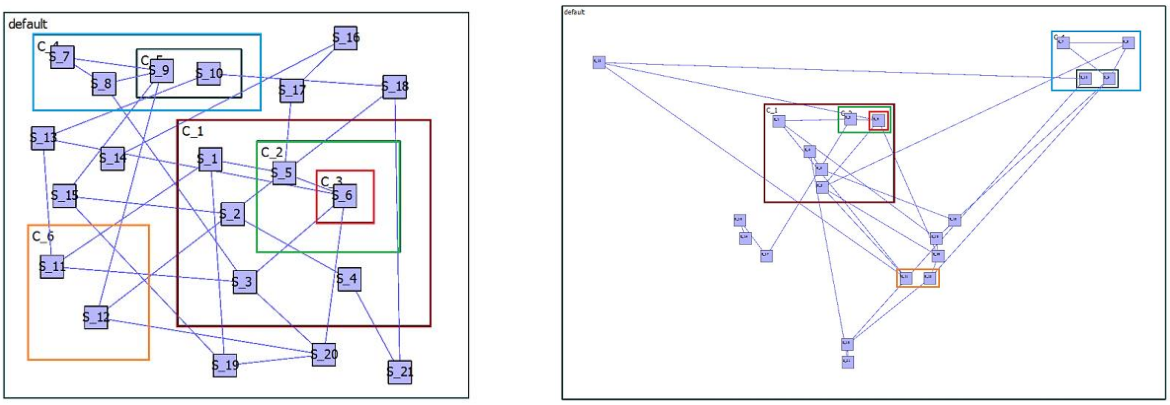
5. Results and Conclusions

We perform the tests on a data set consisting of biological pathways. The graphs are clustered in nature with multiple clusters at a level and multiple levels of nested clusters. We analyze the laid out graphs for number of edge crossings and area of graph.

The experimental results show average reduction in number of edge crossings to 42% of original (Fig 5). Also the nodes in the laid out graphs maintain correspondence with their parent compartments. The proposed algorithm plots the graph in area bounded by $((n-1)(W+S) + W) \times ((n-2)(H+S) + H)$. The algorithm finds the layout in the time in $O(E^2+EV^2 \log V)$.

5.1 Future Scope

In implementation of Wolf's thesis, we consider the problem as VIP-FIX. Also we use basic heuristics approach for sake of simplicity. The number of crossings can be reduced by considering the problem as VIP-VAR and using permutation heuristics. Also we observe that there is scope to reduce the edge crossings among edges between vertices in different clusters by rearranging the internal layout of that cluster. This can be achieved by rotating the layout of cluster or choosing different canonical order.



(a) **(b)**

Figure 4: (a) A clustered input graph, **(b)** Grid Based Straight-Line Layout of graph in (a)

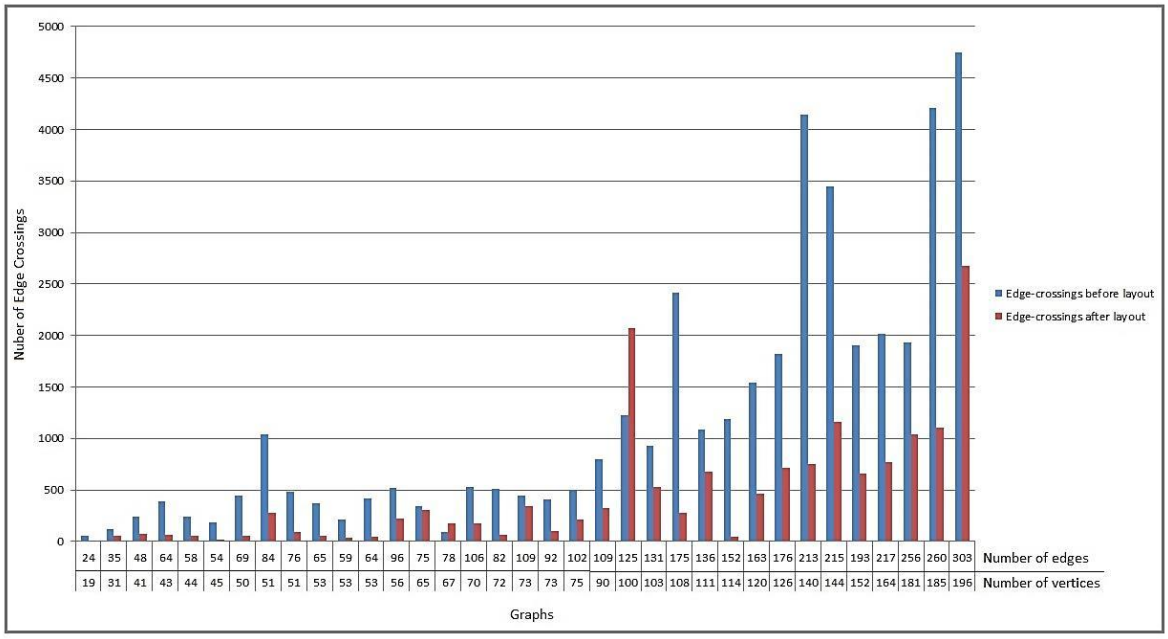


Figure 5: Edge-crossings in graphs before and after applying Grid Based Straight-Line Layout

References

[1] C. Bachmaier, U. Brandes, F. Schreiber, 2013, "Biological Networks", Chapter 20, Handbook of Graph Drawing and Visualization, CRC Press, pp. 621-651.
[2] W.T. Tutte, 1963, "How to draw a graph", Proc. London Math. Soc. 13, pp. 743-768.
[3] C. Thomassen, 1980, "Planarity and Duality of Finite and Infinite Planar Graphs," J. Combinatorial Theory, Series B, vol. 29, pp. 244-271.
[4] N. Chiba, T. Yamanouchi, T. Nishizeki, 1984, "Linear algorithms for convex drawings of planar graphs, in Progress in Graph Theory", J.A. Bondy and U.S.R. Murty (eds.), Academic Press, pp.153-173.
[5] I. Fary, 1948, "On straight line representation of planar graphs", Acta. Sci. Math. Szeged 11, pp. 229-233.
[6] S. K. Stein, 1951, "Convex maps", Proc. Amer. Math Soc. 2, pp. 464-466.
[7] P. Rosentiehl, R. E. Tarjan, 1986, "Rectilinear planar layouts and bipolar orientations of planar graphs", Discrete Computational Geometry 1, pp. 343-353.
[8] H.deFraysseix, J.Pach, R.Pollack, 1990, "How to draw a planar graph on a grid", Combinatorica 10, pp. 41-51.
[9] H.deFraysseix, J.Pach, R.Pollack, 1988, "Small sets supporting Straight-Line Embeddings of planar graphs", Proc. 20th Annual Symposium on Theory of Computing, pp.426-433.
[10] Chrobak M., Payne T. H., 1995, "A Linear-Time Algorithm for Drawing Planar Graph on a Grid", Information Processing Letters, Volume 54, pp. 241-246.
[11] G.Kant, 1993, "Algorithms for Drawing Planar Graphs", Ph.D. Dissertation, Department of Computer Science, University of Utrecht.
[12] Christian Wolf, 2008, "Inserting a Vertex into a Planar Graph", Diploma Thesis, Dortmund University.
[13] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, 1994, "Algorithms for Drawing Graphs: an Annotated Bibliography", Computational Geometry: Theory and Applications, vol. 4, no. 5, pp. 235-282.
[14] Vismara L., 2013, "Planar straight-line drawing algorithms", Chapter 6, Handbook of Graph Drawing and Visualization, CRC Press, June 24, pp. 193-221.

Author Profile

Noopur Landge received B.E. degree in Computer Science Engineering from Amravati University in 2012. He is currently pursuing M.Tech. in Computer Science Engineering from Vishwakarma Institute of Technology, Pune. He is also working on graph layout algorithms in association with Persistent Systems Ltd. Pune.