Force Directed Graph Drawing Algorithms

Naeem Sajjad[1]

Abstract— **The ability to represent relational data in the form of drawings or graphs is a powerful tool which allows us to perform analysis through visual exploration. Several data presentation problems require the drawing or display of graphs; examples include circuit schematics and software engineering diagrams. Force-directed graph drawing algorithms are a special class of graph drawing algorithms used to draw the simple undirected graphs in plane. Graphs drawn with these algorithms tend to be aesthetically pleasing, exhibit symmetries, and tend to produce crossing-free layouts for planar graphs. In this paper we are going to introduce the basic force-directed graph drawing algorithms, we present the critical analysis of these algorithms. At the end of this paper a section is devoted for comparison of all these algortihms based upon our knowledge and study of this specific domain.**

I. INTRODUCTION

To display the relational data in a meaningful way is always been a problem of graph representation. For example consider a scenario in which you need to calculate the optimal placement of electrical components inside the electronic application which contains a large number of small connected components. These components have to be placed on the chip such that the number of crossings is as small as possible, and the required area of the chip must not become too large. The problem become even more complex when several different constraints have to be satisfied as well, for example the number of bends and total length of the connections must be minimized as in design of Very Large Scale Integration (VLSI) chips[1]. A wide variety of fields each with their own requirements utilize automatic graph drawing algorithms to clarify or to display the structure of the information in a compact and relatively small space. As a result, graph drawing algorithms have been a focus of research from the past couple of decades providing a better quality of drawing and higher drawing performance. Several classes of graph drawing algorithms with different aesthetic criterias have been evolved to address the problem of planar drawings. Some of the most flexible algorithms for calculating layouts of simple undirected graphs belong to a class known as force-directed algorithms. Also known as spring embedders, such algorithms calculate the layout of a graph using only information contained within the structure of the graph itself, rather than relying on domain-specific knowledge.

An algorithm designed by Tutte [2] in 1963 was the first one of this class for obtaining a straight line crossing free drawing of 3-connected planar graphs. In this algorithm Tutte constructed an embedding using *barycentric representations*, whose results is guaranteed to be a planar drawing of the graph. The idea behind Tuttes algorithm, is that if a face of the planar graph is fixed in the plane, then suitable positions for the remaining vertices can be found by solving a system of linear equations.

Another algorithm designed by M.J. Fruchterman and M. Reingold[3] model the vertices behavior as atomic particles or celestial bodies, exerting attractive and repulsive forces on each other. This algorithm is actually a further modification of the Eades[4] algorithm with a concentration on two-dimensional, aesthetically-pleasing pictures of graphs by doing simplified simulations of physical systems. It perform well at distributing vertices evenly, making edge lengths uniform, and reflecting symmetry with additional speed and simplicity.

In 1984 Eades[4] proposed a mechanical model of steel rings and springs to produce aesthetically pleasing 2D layouts for plotters and CRT screens. The basic idea of the model proposed is; *To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system. The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state.*

The 1989 algorithm of Kamada and Kawai[5] introduce a dynamic system in which nodes of a graph are mutually connected by springs. This algorithm relate the balanced layout of vertices to the dynamically balanced spring system. As a result, the degree of imbalance can be formulated as the total energy of springs. Pleasing layouts can be obtained by decreasing the total energy of the system and the best results are produced with the minimum total energy of the system. As described in their algorithm *We regard the desirable geometric (Euclidean) distance between two vertices in the drawing as the graph theoretic distance between them in the corresponding graph*.

In 1999 Bor Plestenjak presented *An algorithm for drawing planar graphs*[6]. This algorithm was derived from the Fruchterman and Reingolds spring embedding algorithm. It is as simple and efficient for the Convex Drawing[7] of 3-connected planar graphs which needs only one face of the graph to draw it planarly. It takes the graph as a mechanical model having vertices as metal rings and edges between vertices as elastic bands of zero length. Under the influence of attractive forces of elastic bands, the free vertices will move until the final layout is planar. At the end all vertices will lie inside or on the border of the initial polygon and what is more important, it turns out there are no edge intersections.

This paper is organized in a way, first to explain the basic

[1]Naeem Sajjad is the student of BEng. Software at University of Bradford, UK. This work was done in order to fulfill the requirements of Term Paper for the module of Research Methods, taught by Mr. Sarfraz Raza.

force-directed algorithms then to present a knowledge based analysis and comparison of all presented algorithms.

II. Applications of Graph Drawing Algorithms

Work in this area was done mainly in response to requirements of data visualization techniques and interactive computer systems. Many fields in computer science, such as software engineering, electronic circuit design and database design, have found it useful to represent data as graphs, with vertices denoting elements and edges denoting relations between them. These graphs are normally generated by software tools based on information in the system. As the size of a graph generated from data or constraints grew, so has the sophistication of embedding algorithms. In software engineering, the architecture of a large software system can be visualized as a directed graph with vertices representing modules and edges denoting various relationships between them. These systems are often hierarchical in nature and their drawings should reflect this. Furthermore, this information can be used to make the graph drawing task easier [8]. Computer hardware and microchips are now sufficiently complex that they are designed using CAD tools. It is then the responsibility of the tool to create a layout of the logical gates and the connections between them on microchips and circuit boards. This layout should be a grid drawing. An orthogonal drawing is one in which an edge is a chain of horizontal and vertical segments. A grid drawing is an orthogonal drawing in which all the vertices and bends of edges have integer coordinates [9]. There are many other examples of applications which use graph drawing algorithms. Entity-relationship diagrams in database design can have a visual representation or an algebraic one. One of the requirements of systems analysis and design tools is that a database description need only be entered once in either format and the alternate format will be generated. There is a project management technique that uses PERT charts (Project Evaluation and Review Technique) to track dependencies among tasks. These dependencies form a directed graph from which other information can be derived, such as a project critical path. One technique used by the Human Genome Project analyses the gene structure by representing raw data as a directed graph [10].

III. Graph drawing Algorithms

A. *Tuttes Algorithm of linear equations*

Going back to 1963, the graph drawing algorithm of Tutte [6] was one of the first force directed graph drawing methods based on barycentric representations for obtaining a straight-line, crossings free drawing for a given 3-connected planar graph. A graph G is 3connected (nodally 3-connected) if it is simple and non-separable and satisfies the following condition; if G is the union of two proper subgraphs H and K such that H intersection K consists solely of two vertices u and v, then one of H and K is a link-graph (arc-graph) with ends u and v. Unlike all other graph drawing techniques Tutte guarantees that the resulting graph is crossing free and all faces of the graph are convex. The main theme of the Tuttes Algorithm is that if one of the faces of planar graph is fixed in the plane, then suitable positions for the remaining vertices, which are represented as a convex combination of the positions of its neighbors, can be found by solving a system of linear equations.

The basic Algorithm of Tutte is described below; in this algorithm the force because of an edge (u, v) incident on the vertices is proportional to the distance between vertices u and v.

Input: *G = (V, E) with V = V0 intersection V1, with fixed vertices V0 and free vertices V1; a strictly convex polygon P with V0 vertices.*

Output: *A position pv for each vertex of V, such that the fixed vertices form a convex polygon P.*

Initialize V0: *Place fixed vertices u V0 at corners of P;*

Initialize V1: *Place free vertices v in V1 at the origin;*

repeat
foreach *free vertex v in V1*
do
xv (1 deg(v)) (u,v)E xu;
yv (1 deg(v)) (u,v)E yu;
until xv and yv converge for all free vertices V;

In this model the force due to an edge (u, v) is proportional to the distance between vertices u and v and the springs have ideal length of zero; there are no explicit repulsive forces. The vertex set is partitioned into fixed and free vertices. Setting the partial derivatives of the force function to zero results in independent systems of linear equations for the x-coordinate and for the y-coordinate.

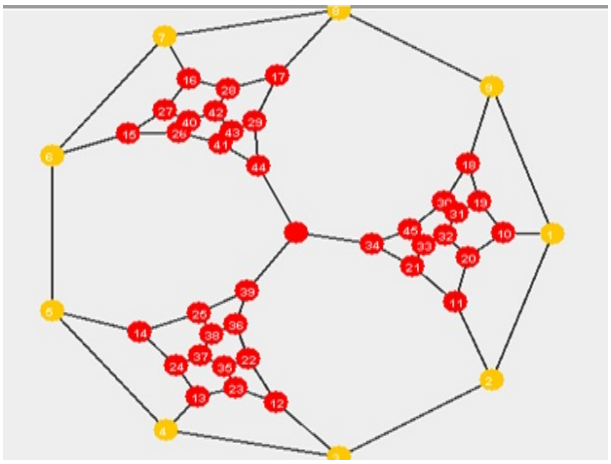


Fig. 1: Tutte's graph

The equations in the for-loop are linear and the number of equations is equal to the number of the unknowns, which

in turn is equal to the number of free vertices. Solving these equations results in placing each free vertex at the barycenter of its neighbors. An example of the tutte's algorithm drawing is given in Fig.1. The basic drawback of this approach is that it only applies to 3-connected planar graphs. The resulting drawing is not always *aesthetically pleasing* and may have poor vertex resolution. It works only for smaller graphs e.g. a graph with less than hundered nodes.

B. Algorithm of Eades

The algorithm of Eades was presented in 1984 [4] which can target up to 30 vertices and uses a mechanical model to embed a graph that replaces the vertices by steel rings and replace each edge with a spring to form a mechanical system. After the initial placement of vertices is done, let them go freely under the logarithmic strength forces of spring, these forces on the rings moves the whole system into a minimal energy state. Logarithmic strength force exerted by a spring is given as;

$$c1 * log\ (d/c2) \tag{1}$$

Where d is the distance between given two vertices, C1 and C2 are constants. *Logarithmic* strength springs are used instead of Linear springs because the springs between vertices those are far apart become too strong, although edges exert no force when distance d is equal to constant C2. Non-adjacent vertices in this algorithm repels each other by a force that is equal to third order law, given as;

$$c3\ /\ (d)2 \tag{2}$$

Where C3 is the constant and d is the distance between vertices. The overall structure or pseudo code of the Eades Algorithm is shown below.

algorithm*SPRING(G:graph);*

place vertices of G in random locations;
repeat *M times*
calculate the force on each vertex;
*move the vertex c4 * (force on vertex)*
draw graph on CRT or plotter.

This excellent description of the algorithm proves the simple and conceptual nature of *Eades* algorithm in particular and same behaviour of all *spring embedder* algorithms in particularly.

Another survey[11] of graph drawing algorithms has shown, using Eades Algorithm; *The values c1 = 2, c2 = 1, c3 = 1, c4 = 0.1, are appropriate for most graphs. Almost all graphs achieve a minimal energy state after the simulation step is run 100 times, that is, M = 100.*

C. Graph Drawing by Force-directed Placement

Going back to 1991 the Fruchterman and Reingolds Algorithm for Graph Drawing by Force-directed placement[3] added even vertex distribution criteria to the early two methods of Tutte[2] and Eades[4]. It treats the vertices of the graph as atomic particles or celestial bodies, exerting attractive and repulsive force form one another. Two main functions of the algorithm are:
(i) keeping vertices closely connected by an edge and
(ii) preventing vertices from being drawn too close to each other.

The attractive and repulsive forces between vertices are modeled as,

$$fa(d) = d2 \tag{3}$$

$$fr(d) = k2/d \tag{4}$$

where d is the distance between two vertices and K is the optimal distance between vertices given as,

$$k = Carea/numberofvertices \tag{5}$$

The value of constant C can be found experimentally. The cooling function *cool(i)* is a major function responsible of providing aesthetically pleasing drawing with this algorithm. At each step the displacement of a vertex is limited to some maximum value *cool(i)*. The temperature starts from some initial high value, allowing more displacements and decreases to zero in linear fashion allowing smaller and smaller displacements. Due to decrease in temperature after some number of iterations the displacements become so smaller that the layout freezes. An example drawing of Icosahedron by Fruchterman and Reingolds Algorithm is shown in Fig.2

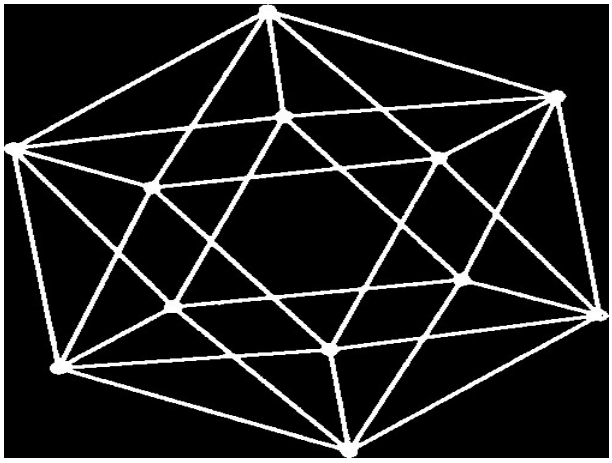


Fig. 2: Icosahedron Drawn by Fruchterman and Reingolds Algorithm

At each step the basic algorithm computes O (E) attractive forces: because vertices are only attracted to their neighbors and O (V square) repulsive forces: because we have to compute the repulsion of every vertex from every other. To reduce the quadratic complexity of the repulsive forces, Fruchterman and Reingold suggest neglecting the contribution of the more distant vertices in repulsive forces, because the repulsive force decreases as the inverse square of the distance.

About placement of vertices authors in[3] said that; *it depends on how many vertices there are and how much space is available. Some graphs are too complicated to draw attractively at all. Our vague guidelines recall a result from particle physics: At a distance of about 1 fm [femto-meter] the strong nuclear force is attractive and about 10 times the electric force between two protons. The force decreases rapidly with increasing distance, becoming completely negligible at about 15 times this separation. When two nucleons are within about 04 fm of each other, the strong nuclear forces become repulsive. Thus nuclei do not collapse.* Satisfying the neglection of repulsive forces between distant verices Speeds up factors used by authors are Grid Variant, simulated sintering and quenching.

D. An Algorithm for Drawing General Undirected Graphs

The 1989 algorithm of Kamada and Kawai introduced a different way of thinking about good graph layouts. The previous algorithms of Eades[4] and Fruchterman[3] keeps the vertices closer to each other but making sure that vertices are not too close to each other. This algorithm of Kamada and Kawai[5] takes a different approach described as;

We regard the desirable geometric (Euclidean) distance between two vertices in the drawing as the graph theoretic distance between them in the corresponding graph.

In this graph drawing algorithm, first the distance d_{ij} (between the vertices) must be computed for all pairs of vertices in a given graph. Then the length of the shortest path between two vertices i anj j is computed as l_{ij} and the strength of the spring between these two vertices is calculated as k_{ij}. Although initial positions of the vertices do not have a great influence on the resulting drawing, we must specify the initial positions of vertices for some special cases; e.g., all the vertices hold a single line. After initialization with having some initial position the total energy E of the system decreases, step by step, by moving a vertex to a stable position.

The algorithm is described as follows.

compute d_{ij} *for* $1 \leq i \neq j \leq n$;
compute l_{ij} *for* $1 \leq i \neq j \leq n$;
compute k_{ij} *for* $1 \leq i \neq j \leq n$;
initialize P1 ,P2,.....,Pn;
While ($max_i \triangle_i \geq \emptyset$)
let P_m *be the particle satisfying* $\triangle_m = max_i \triangle_i$;
while ($\triangle_m \geq \emptyset$)
compute δx *and* δy
$x_m := x_m + \delta x$;
$y_m := y_m + \delta y$;

In this model, the perfect drawing of a graph would be one in which the pair-wise geometric distances between the drawn vertices match the graph theoretic pairwise distances, as computed by an All-Pairs-Shortest-Path computation. As this goal cannot always be achieved for arbitrary graphs in 2D or 3D Euclidean spaces, the approach relies on setting up a spring system in such a way that minimizing the energy of the system corresponds to minimizing the difference between the geometric and graph distances. In this model there are no separate attractive and repulsive forces between pairs of vertices, but instead if a pair of vertices is (geometrically) closer/farther than their corresponding graph distance the vertices repel/attract each other. A triangulated triangle graph drawn by Kamada and Kawai's Algorithm is shown in Fig.3.

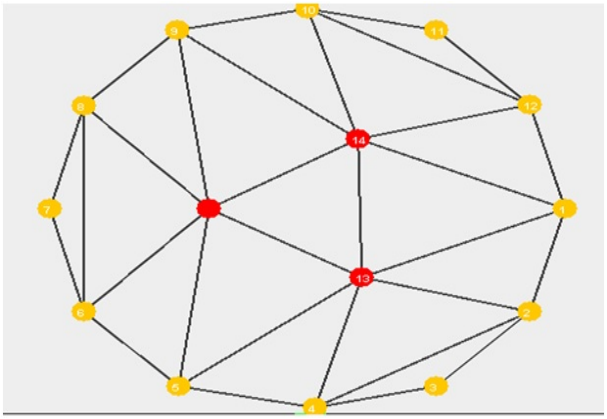


Fig. 3: Triangulated Triangle Drawn by Kamada and Kawai's Algorithm

The algorithm of Kamada and Kawai is computationally expensive, requiring an All-Pair- Shortest-Path computation which can be done in O(V^3)time. Furthermore, the algorithm requires O(V^2) storage for the pairwise vertex distances. Despite the higher time and space complexity, the algorithm contributes a simple and intuitive definition of a good graph layout: A graph layout is good if the geometric distances between vertices closely correspond to the underlying graph distances.

E. An Algorithm for Drawing Schlegel Diagrams

The algorithm described over here was presented by Bor Plestenjak[6] in 1990 and was derived from the Fruchterman and Reingolds spring embedding algorithm. It is a simple and efficient algorithm for the Convex Drawing [1] of 3-connected planar graphs which needs only one face of the graph to draw it planarly. It takes the graph as a mechanical model having vertices as metal rings and edges between vertices as elastic bands of zero length. A further modification of this algorithm is that it fixes some vertices on the outer face of a regular polygon while sets the other vertices free and assumes that edges are not interlaced. Under the influence of attractive forces of elastic bands, the free vertices will move until the final layout is planar. At the end all vertices will lie inside or on the border of the initial polygon and what is more important, it turns out there are no edge intersections.

In each step the basic algorithm computes only the attractive forces acting on each vertex, since it deletes all repulsive forces, and moves the vertex in direction of resulting force. Temperature is used to limit the maximum displacement in each step, called cooling function Cool(i) which satisfies,

$$\lim_{i \to \infty} cool(i) = 0$$

The temperature starts from some initial high value, allowing more displacements and decrease to zero in linear fashion allowing smaller and smaller displacements. Due to decrease in temperature after some number of iterations the displacements become so smaller that the layout freezes. If the forces and cooling function cool (i) are chosen properly, the layout produces is so near to equilibrium that one can say it is planar.

To calculate the attractive force between two adjacent nodes x and y, the algorithm uses the third order law,

$$F_{xy} = Cd^3 \tag{6}$$

Here C is the constant and d is the distance between adjacent vertices. The basic algorithm consists of the following steps.

- *Position all vertices of an outer face W in vertices of a regular polygon of size k inscribed into the unit circle and put all other vertices into the origin.*
- *For i :=1to iterations:*
  - *For all vertices y ∈ V: set the resultant force F_y to zero. F_y = 0.*
  - *For all edges (x,y) ∈ E: calculate the attractive force F_{xy} and update the resultant forces F_x and F_y. accordingly.*
  - *For all vertices y ∈ (V-W): move vertex y in the direction of resultant force F_y for the size of the force, but not more than for the value of Cool(i).*

It is obvious that a particular choice of functions C will not give suitable results for all planar graphs. Constant C has to be in harmony with maximum displacement. If C is too large then every vertex is shifted for maximum displacement irrespective of force acting on that vertex and vice versa. A suggested value of constant C is given below.

$$C = \sqrt{n}/\pi \tag{7}$$

This is not the only limitation of the algorithm. Namely, the algorithm can draw correctly only those graphs that are 2-connected, otherwise some vertices and edges may overlap. This means that using this algorithm, planar graphs like trees cannot be drawn automatically. In some cases this problem can be effectively solved by adding new temporary edges before running the algorithm and deleting them after the figure without overlapping is drawn.

The cooling function Cool(i), determines the maximum allowed displacement of vertices in step i of the algorithm. Since the algorithm is based on the simulated annealing, the function Cool(i) must satisfy the condition.

$$\lim_{i\to\infty} cool(i) = 0$$

Choosing a suitable value of Cool(i) is a critical part of the algorithm. If the temperature decreases too slowly then the total displacements resulting in a specific step are too large and more steps will be required to reach the equilibrium. On the other hand, if temperature decreases too fast then the layout freezes in a non-planar layout. A suggested value of the Cool(i) is given below.

$$\text{cool(i)} = \tfrac{\pi}{n}/(1 + (\pi/n)i^{3/2})$$

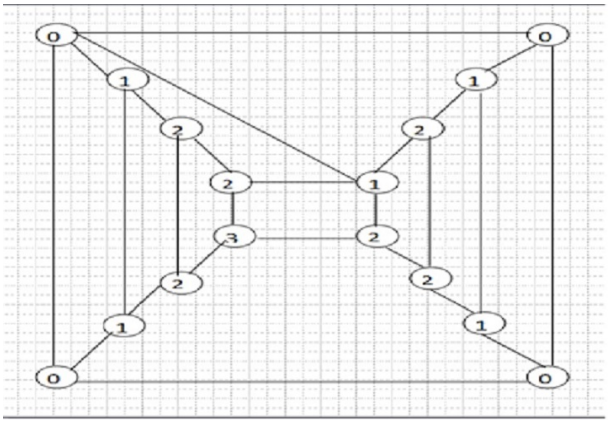


Fig. 4: Peripheriсity of Vertices

A further modification of the algorithm is to introduce the concept of Peripheriсity of a vertex to the algorithm described earlier. If the previous algorithm is applied to a graph having a large number of faces, e.g. a fullerene on hundred or more vertices, it produces a graph with large faces on the border of figure and large number of crowded small faces at the middle of figure. To overcome this layout problem the algorithm uses the concept of Peripheriсity assuming that the bands holding vertices are not equally strong. This assumption is necessary for the algorithm to terminate with approximately equally arranged faces.

Peripheriсity of a vertex is defined as the length of the shortest path from the vertex to the outer cycle. Concept of the Peripheriсity is more evident in the Fig.4, where vertices of a graph are labeled according to their shortest path to the outer cycle. The coefficient of the band strength, denoted as, C (x.per, y.per) between vertices x and y is a function of peripheriсities x.per and y.per. If the maxim Peripheriсity of a given graph is maxper, then the bands with the x.per + y.per = 0 should be the strongest and the bands with x.per+y.per=2maxper should be weakest.

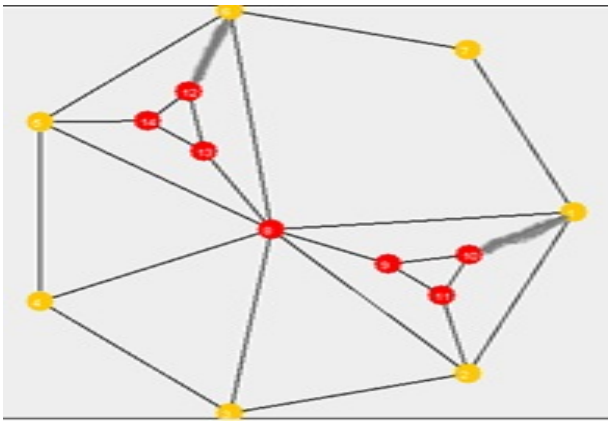


Fig. 5: Planar Layout after adding temporary edges

The bands strength coefficient is chosen experimentally and can be supposed as a free parameter of the algorithm. Author suggests that the best results are obtained when the coefficient C (x.per, y.per) forms a geometric progression

form.

$$C(x.per, y.per) = \sqrt{\frac{n}{\pi}} exp(A * \frac{2maxper - x.per - y.per}{maxper}) \quad (8)$$

For constant A, the author suggest a value of A=2.5 to obtain better results.

To improve the efficiency of earlier described algorithm this approach applies the concept of Oscillation of a vertex. When the algorithm is applied to a given graph, after some initial number of steps, the vertices begin to oscillate between two states regularly. If we compare the coordinates of each vertex with its coordinates resulting from two steps ago, the algorithm can be exited if the difference between the coordinates calculation is below the prescribed constant e. this approach reduces the number of steps of algorithm without affecting the quality of produced layout.

This algorithm was actually derived for 3-connected planar graphs; it can be applied to other non-planar graphs as well. For every given graph we can set an arbitrary set of vertices on a regular polygon and set other vertices free inside the polygon. By applying a mechanical model to the graph we can search for the equilibrium and after adding new temporary edges, colored as gray, in Fig.5 we got a planar layout of the graph. Once figure has been drawn planarly we can simply delete those temporary edges. By deleting some edges of a planar graph it remains planar, although the final layout produced by such a graphs needs not to be in plane, but it may contain a useful information about structure and symmetry of the graph.

The main advantage of Bor Plestenjaks Algorithm over other force-directed graph drawing algorithms is improvement in running time, regardless of filtration techniques used in other algorithms. This requires only one face of the graph to draw it planarly while other algorithms require all faces of a graph to draw it planarly. The algorithm of Fruchterman and Reingold reduces the time complexity by computing repulsive forces between every pair of vertices, but attractive forces are calculated only between neighbors. This reduces the time complexity because calculating the attractive forces between neighbors is thus O(E), although the repulsive force calculation is still O(V^2).

The algorithm proposed by Bor Plestenjak deletes all repulsive forces between vertices and calculates only attractive forces between neighbors, which consumes O(E+V) time to complete one step of the algorithm.

The example drawing in Fig.1 is a Tutte graph, a graph that is three-regular, planar, non-hamiltonian and three-connected. There exists an ordering of the vertices so that each two vertices in the sequence are connected and the first vertex is connected to the last vertex in this ordering. Removing any two vertices leaves the graph connected, i.e., there remains a path between each two vertices. But removing three vertices may result in a disconnected graph.

IV. CONCLUSION

The concluding remark of research study on these five planar graph drawing algorithms is that for particular 3-connected graphs with finite number of Vertives, the algorithm proposed by Dr. Bor Plestenjak[6] perform better in terms of simplicity and time complexity. Algorithm of Kamada and Kawai[5] guarantees planar output but resultant drawing is poor for large number of vertices, similarly algorithms described in [2],[3],[4] also have more limitations than the benifits over the algorithm of Dr. Bor Plestenjak.

ACKNOWLEDGMENT

We would like to thank our research supervisor Mr. Sarfraz Raza for his continues guidance and suggestions to compile and complete this research work at The University of Bradford, UK.

REFERENCES

[1] Baker, R. Jacob (2010). CMOS: Circuit Design, Layout, and Simulation, Third Edition. Wiley-IEEE. p. 1174. ISBN 978-0-470-88132-3.

[2] William T. Tutte. How to draw a graph. Proc. London Math. Society, 13(52):743768, 1963.

[3] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. Software Practice and Experience, 21(11): 1129-1164, 1991.

[4] Peter Eades. A heuristic for graph drawing. Congressus Numerantium,42:149160, 1984.

[5] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. Inform. Process. Lett., 31:715, 1989.

[6] B. Plestenjak: An Algorithm for Drawing Planar Graphs, Softw. Pract. Exper.Volume 29, Issue 11, pages 973984, September 1999.

[7] W.T. Tutte, Convex Representation of Graphs, Proc. London Math. Soc., 10, 304-320 (1960).

[8] H.A. Mller, M.A. Orgun, S.R. Tilley and J.S. Uhl. A Reverse-engineering approach to subsystem structure identification. Software Maintenance: Research and Practice, 5:181-204, 1993.

[9] G. Di Battista, P. Eades, R. Tamassia, And I. G. Tollis. Algorithms for Drawing Graphs: an Annotated Bibliography. June, 1994.

[10] E. Harley and A.J Bonner. A Flexible Approach to Genome Map Assembly. Proceedings of the International Symposium on Intelligent Systems for Molecular Biology, pp. 161-169, 1994.

[11] C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.